Consistency Maintenance in Concurrent Representations

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science

at the

University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy (Computer Science)

by

Anand Natrajan

© Copyright by

Anand Natrajan

All Rights Reserved

January 2000

APPROVAL SHEET

This dissertation is submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy (Computer Science)

Anand Natrajan

This dissertation has been read and approved by the Examining Committee:

Paul F. Reynolds, Jr. (Thesis Advisor)

Gabriel Robins (Committee Chairman)

Anita K. Jones

Worthy N. Martin

Kevin J. Sullivan

Ronald D. Williams

Accepted for the School of Engineering and Applied Science:

Dean Richard W. Miksad
School of Engineering and Applied Science

January 2000

One often hears of writers that rise and swell with their subject, though it may seem but an ordinary one. How, then, with me, writing of this Leviathan? Unconsciously my chirography expands into placard capitals. Give me a condor's quill! Give me Vesuvius' crater for an inkstand! Friends, hold my arms! For in the mere act of penning my thoughts of this Leviathan, they weary me, and make me faint with their out-reaching comprehensiveness of sweep, as if to include the whole circle of the sciences, and all the generations of whales, and men, and mastodons, past, present, and to come, with all the revolving panoramas of empire on earth, and throughout the whole universe, not excluding its suburbs. Such, and so magnifying, is the virtue of a large and liberal theme! We expand to its bulk. To produce a mighty book, you must choose a mighty theme. No great and enduring volume can ever be written on the flea, though many there be who have tried it.

— *Herman Melville,* Moby-Dick

Abstract

Multi-Representation Modeling (MRM) involves executing multiple models of the same phenomenon jointly. MRM is a technique in modeling and simulation for capturing the combined semantics of multiple models. Previous MRM approaches, such as selective viewing and aggregation-disaggregation, have encountered problems such as chain disaggregation, temporal inconsistency and mapping inconsistency. Eliminating these problems has been a difficult task for MRM designers. We eliminate these problems by showing how to achieve MRM effectively, i.e., correctly, consistently and inexpensively. Our thesis is that MRM can be effective. Maintaining consistency among the concurrent representations of jointly-executing models is our approach for effective MRM.

We developed a framework, UNIFY, to achieve effective MRM. UNIFY satisfies three MRM requirements: multi-representation interaction, multi-representation consistency and cost-effectiveness. It enables designers to construct solutions for application-specific multiple models. UNIFY is based on four fundamental observations that reduce the problem of joint execution to the problem of maintaining consistency among the representations of multiple models when dependent concurrent interactions occur. UNIFY consists of processes and techniques such as Multiple Representation Entities (MREs), Attribute Dependency Graphs (ADGs) and a taxonomy of interactions. An MRE maintains concurrent representations. An ADG captures relationships among attributes in concurrent representations. An ADG and application-specific mapping functions that translate attributes across representations constitute a Consistency Enforcer that maintains internal consistency within an MRE. Our taxonomy of interactions provides a way to

classify interactions based on their semantic characteristics. This classification presents policies that can be encoded in an Interaction Resolver for resolving the effects of dependent concurrent interactions on an MRE.

UNIFY contributes to the practice of modeling and simulation. We show how designers can apply techniques in UNIFY. We present guidelines for maintaining consistency among concurrent representations. UNIFY is the first known general framework for achieving effective MRM.

Acknowledgements

It is a myth that a dissertation is the soul-wrenching creation solely of its author's time, toil and tenacity. Many people conspired to drag this author kicking and screaming towards his goal. I thank these people for conspiring to do so.

—— ~~~ ——

I am thankful to Paul Reynolds, my advisor and friend, for giving me guidance and counsel, and for having faith and confidence in me. His patience in reading draft after draft of every paper, proposal and idea I wrote up continues to amaze me. No one should be subjected to the torture of reading my early attempts at technical writing, and thanks to Paul, no one will. I appreciate Paul's fine balance between giving me the freedom to pursue what fired me and reining in my imagination when it got the better of me. I thank him for always being willing to meet me whenever I barged into his office.

I am grateful to my committee members for their comments and suggestions. I have benefitted greatly from their advice. I thank Worthy Martin and James French for lending a sympathetic ear and putting my toils in perspective. It has been a pleasure working with my colleagues, in particular, Anh Nguyen-Tuong, Rashmi Srinivasa, Sudhir Srinivasan and Glenn Wasson. Many of the ideas in my work originated in discussions with them. I am deeply grateful to them for investing time and energy discussing ideas with me and tolerating my many opinionated digressions. Gabriel Ferrer, John Karro, Allison Powell and Rashmi Srinivasa deserve credit for reading sections of my work. Their incisive comments made me re-think how I presented my ideas. Any errors that remain in this presentation are attributable to my negligence or stubbornness.

I thank the Defense Modeling and Simulation Office, US Army SIMTECH and Janet Morrow for making it possible for me to do my research.

—— ~~~ ——

This dissertation would not have been possible without Rashmi Srinivasa, my wife and friend. Her support and encouragement has seen me through tumultuous times. I thank her for simultaneously brandishing a sword to quell the demons of my insecurities, a spoon to bake delectable desserts and a wand to bring joy to my life in so many different ways.

I thank my parents, Subramanian and Shanta Natrajan, and brother, Arvind Natrajan, for their unflagging belief that despite their incomprehension about what I do, I must be saving the world. I am indebted to my parents for inculcating in me the dedication and discipline to do whatever I undertake well. I cannot thank my brother enough for showing me what it is to be a free spirit.

I have been fortunate to have many friends who cherish me despite my eccentricities. I risk doing them a disservice by not mentioning all of them here, but plead paucity of space. I thank Glenn Wasson for goading me through weight-lifting, imploring me to spike the volleyball, teaching me about baseball and being a sink for my bile during our coffee *klatsches*. I thank Anh Nguyen-Tuong for letting me ramble about programming languages at three in the morning, putting me in my place at racquetball and being a friend to me in my early days at UVa. I thank Suresh Balasubramaniam, Karine Boulle, Aaron Cass, John Jones, Gopal Kumar, Sally McKee, Venkataraman Pallassana, John Regehr, Prakash Vachaspati, Ravichandran Vancheeswaran, Murtuza Vasowalla, Aruna Viswadoss, Soumya Viswanathan, Chenxi Wang and Jennifer Wong for many good times.

Finally, I am thankful for the many diversions I have enjoyed during my sojourn here. Without them, crossing over to the realms of insanity would have been entirely within reach. My various hobbies, my books, my cooking, and beautiful, beautiful Charlottesville have connived to ensure that the road to my goal was not as bumpy as it could have been.

There is a time for some things, and a time for all things;
a time for great things, and a time for small things. …
But all in good time.
— *Miguel de Cervantes,* Don Quixote

Table of Contents

Abstract i

Acknowledgements iii

Table of Contents vi

List of Figures xi

List of Tables xiv

List of Symbols xvi

Chapter 1 Introduction 1

1.1 Background 3

1.2 `UNIFY` — An Overview 5

1.3 Requirements for Effective MRM 8

1.4 Claims and Contributions 9

1.5 Evaluation 10

1.6 Outline 12

Chapter 2 Related Work 14

2.1 MRM Applications 14

2.1.1 Multi-Resolution Graphical Modelling 15

2.1.2 Hierarchical Autonomous Agents 15

2.1.3 Blackboard Systems 16

2.1.4 Cache Coherence 16

2.1.5 Abstract Data Types and Object Inheritance 16

2.1.6 Views in Databases and Integrated Environments 17

2.1.7 Nested Climate Modelling . . . 17
2.1.8 Integrated Molecular Modelling . . . 18
2.1.9 Multi-Level Computer Games . . . 18
2.1.10 Battlefield Simulations . . . 19
2.1.11 MRM Applications Summary . . . 19
2.2 Multi-Model Execution . . . 20
2.2.1 Selective Viewing . . . 21
2.2.2 Aggregation-Disaggregation . . . 22
2.2.2.1 Full Disaggregation . . . 23
2.2.2.2 Partial Disaggregation . . . 23
2.2.2.3 Playboxes . . . 24
2.2.2.4 Pseudo-Disaggregation . . . 25
2.2.3 Variable Resolution Modelling . . . 26
2.3 Maintaining Consistency among Concurrent Representations . . . 27
2.4 Chapter Summary . . . 31
Chapter 3 Foundation . . . 32
3.1 Model . . . 32
3.2 Interactions . . . 37
3.3 Multi-models . . . 41
3.3.1 Cross-model Relationships . . . 43
3.3.2 Mapping Functions . . . 43
3.3.3 Time-Steps . . . 44
3.4 Evaluation . . . 46
3.5 Assumptions and Rationale . . . 49
3.6 Chapter Summary . . . 52
Chapter 4 Fundamental Observations . . . 53
4.1 Problems with Aggregation-Disaggregation . . . 54
4.1.1 Mapping Inconsistency . . . 55
4.1.2 Chain Disaggregation . . . 56
4.1.3 Transition Latency . . . 56
4.1.4 Thrashing . . . 57
4.1.5 Network Flooding . . . 57
4.1.6 Cross-Level Interactions . . . 58
4.1.7 Summary of Problems with Aggregation-Disaggregation . . . 58
4.2 Fundamental Observations . . . 59
4.2.1 Fundamental Observation 1 . . . 59
4.2.2 Fundamental Observation 2 . . . 62
4.2.3 Fundamental Observation 3 . . . 64

4.2.4 Fundamental Observation 4 65
4.3 Chapter Summary 69
Chapter 5 Multiple Representation Entities. 70
5.1 Description of an MRE 72
5.2 Challenges. 73
5.3 Rationale 75
5.4 Execution of an MRE 77
5.4.1 Maintaining Consistency 78
5.4.1.1 Temporal Consistency. 78
5.4.1.2 Mapping Consistency 80
5.4.2 Resolving Concurrent Interactions 81
5.4.3 Storing Attributes in a Core 81
5.4.4 Comparing against Alternative Approaches 84
5.4.4.1 Comparing against aggregation-disaggregation 84
5.4.4.2 Comparing against selective viewing 85
5.5 Benefits of MREs 86
5.6 Limitations of MREs 88
5.7 Chapter Summary 92
Chapter 6 Consistency Enforcers 95
6.1 Constructing an Attribute Dependency Graph 96
6.1.1 Assigning Nodes to Attributes 99
6.1.2 Assigning Arcs to Dependencies 100
6.1.3 Assigning Semantics to Dependencies 101
6.1.3.1 Cumulative and Distributive Dependencies 102
6.1.3.2 Interaction and Modelling Dependencies 103
6.1.3.3 Selecting Dependencies 103
6.1.3.4 Properties of Dependency Classes 104
6.1.3.5 Examples of Dependency Classes 105
6.1.3.6 Dependency Weights. 106
6.1.3.7 Interaction Semantics 109
6.1.4 Summary of Attribute Dependency Graphs 109
6.2 Selecting Mapping Functions 110
6.3 Traversing an ADG. 111
6.3.1 Algorithm for Traversing an ADG 112
6.3.2 Cyclic Dependencies. 114
6.3.3 Unplanned Dependencies 116
6.3.4 Traversal Path 116
6.4 Possible Implementations of a Consistency Enforcer. 118
6.4.1 As-Is 118

6.4.2 Spreadsheets . . . 119
6.4.3 Attribute Grammars . . . 120
6.4.4 Mediators . . . 122
6.4.5 Constraint Solvers . . . 122
6.5 Chapter Summary . . . 124
Chapter 7 Interaction Resolvers . . . 125
7.1 Interactions . . . 126
7.2 Serialization . . . 127
7.3 Abandoning Isolation . . . 130
7.4 Switches — A Simple System . . . 133
7.4.1 Unconstrained System . . . 133
7.4.2 Constrained System . . . 134
7.4.3 Dependent Concurrent Interactions . . . 136
7.4.4 Complexity . . . 137
7.5 A Taxonomy of Interactions . . . 139
7.5.1 Properties of a Taxonomy of Interactions . . . 139
7.5.2 Interaction Characteristics and Classes . . . 141
7.5.2.1 Request and Response . . . 141
7.5.2.2 Certain and Uncertain . . . 142
7.5.2.3 Combining Characteristics . . . 143
7.5.3 Evaluating the Taxonomy . . . 143
7.5.4 Resolving Effects of Concurrent Interactions . . . 144
7.5.5 Policies for Resolving Effects of Interactions . . . 148
7.6 Constructing an Interaction Resolver . . . 150
7.6.1 Operation of an IR . . . 150
7.6.2 An Example IR . . . 154
7.7 Chapter Summary . . . 159
Chapter 8 Applying `UNIFY`: A Process . . . 160
8.1 Guidelines for MRM Designers . . . 161
8.2 Using `UNIFY` with a Specification Methodology . . . 163
8.3 Process for Effective MRM . . . 166
Chapter 9 Evaluation . . . 170
9.1 Evaluating `UNIFY` in terms of MRM Requirements . . . 170
9.1.1 Multi-Representation Interaction . . . 171
9.1.2 Multi-Representation Consistency . . . 172
9.1.3 Cost-Effectiveness . . . 173
9.1.3.1 Assumptions . . . 174
9.1.3.2 Consistency Cost . . . 177

9.1.3.3 Simulation cost 179
9.1.3.4 Expected Costs 180
9.1.3.5 Experimental Costs 180
9.1.3.6 Summary of Cost-Effectiveness 187
9.1.4 Summary of Evaluation in Terms of MRM Requirements 187
9.2 Applying UNIFY to Existing Models 187
9.2.1 Military Models 188
9.2.2 Autonomous Agent Model 189
9.3 Limitations 191
9.4 Chapter Summary 193
Chapter 10 Conclusions 194
10.1 Contributions 195
10.2 Future Work 199
Appendix A Examples of Multiple Representations 201
Appendix B Joint Task Force Prototype 216
Appendix C Joint Precision Strike Demonstration 241
Appendix D Real-time Platform Reference 264
Appendix E Hierarchical Autonomous Agents 287
Indexed Glossary 300
References 310

A foolish consistency is the hobgoblin of little minds ...
— *Ralph Waldo Emerson,* Self-Reliance

List of Figures

Figure 1: Our Approach to MRM . . . 6
Figure 2: Full Disaggregation . . . 23
Figure 3: Partial Disaggregation . . . 24
Figure 4: Playbox . . . 25
Figure 5: Pseudo-disaggregation . . . 26
Figure 6: Possible compatible time-steps . . . 45
Figure 7: Mapping Inconsistency . . . 55
Figure 8: Chain Disaggregation . . . 56
Figure 9: Fundamental Observation 1 . . . 60
Figure 10: Reducing transition overheads by limiting propagation of transitions . . . 63
Figure 12: Dependency considerations . . . 64
Figure 11: Concurrent multi-level interactions . . . 64
Figure 13: Time-steps — Equal and In-phase . . . 66
Figure 14: Time-steps — Equal but not In-phase . . . 67
Figure 15: Time-steps — Unequal and not In-phase . . . 67
Figure 16: Compatible Time-steps . . . 68
Figure 17: Eliminating time-step differentials . . . 68
Figure 18: An MRE . . . 72
Figure 19: Multi-representation Interaction . . . 72
Figure 20: Execution of an MRE . . . 78
Figure 21: T-joint entity . . . 79
Figure 22: Core attributes . . . 82

Figure 23: Eliminating Chain Disaggregation 86
Figure 24: Reducing Network Flooding 87
Figure 25: Simple ADG 96
Figure 26: Platoon-Tanks MRE 98
Figure 27: Nodes in the ADG for the Platoon-Tanks MRE 100
Figure 28: Dependencies in the ADG for the Platoon-Tanks MRE 101
Figure 29: Dependency Classes in the ADG for the Platoon-Tanks MRE 106
Figure 30: Cumulative Weights 106
Figure 31: Distributive Weights 107
Figure 32: Mapping Value Spaces 110
Figure 33: Mapping Changes in Values 111
Figure 34: Algorithm for ADG Traversal 113
Figure 35: Applying the Effects of an Interaction 114
Figure 36: Propagation of Interaction Effects 115
Figure 37: Clients and Server 127
Figure 38: Switches 133
Figure 39: State Transition Diagram 134
Figure 40: Constrained Switches 134
Figure 41: New States 134
Figure 42: Constrained State Transition Diagram 135
Figure 43: Transitions on Concurrent Interactions 137
Figure 44: Classes of Interactions 143
Figure 45: Concurrent Interactions Affecting Sets of Attributes 146
Figure 46: Independent Concurrent Response and Request Interactions 146
Figure 47: Algorithm for Resolving Interactions 153
Figure 48: Process for Effective MRM 168
Figure 49: Entity in Synthetic Application 177
Figure 50: AD — Consistency Cost 178
Figure 51: SV — Consistency Cost 178
Figure 52: `UNIFY` — Consistency Cost 179
Figure 53: (Left to Right) AD, SV and `UNIFY` — Simulation Cost 179
Figure 54: Expected Costs 180
Figure 55: Simulation Cost varying with Number of Interactions 184
Figure 56: Consistency Cost varying with Number of Interactions 184
Figure 57: Simulation Cost varying with Rate of Simulation 184
Figure 58: Consistency Cost varying with Rate of Simulation 185

Figure 59: Simulation Cost varying with Number of Sub-entities 185
Figure 60: Consistency Cost varying with Number of Sub-entities 185
Figure 61: Simulation Cost varying with Number of Levels 186
Figure 62: Consistency Cost varying with Number of Levels 186
Figure 63: AD, SV and `UNIFY` — Cost Comparison 186
Figure 64: Marcus and Archway 190
Figure 65: MRE for planner and PA system representations 190
Figure 66: Platoon-Tanks MRE 217
Figure 67: ADG for the JTFp Platoon-Tanks MRE 227
Figure 68: JTFp Platoon-Tanks MRE. 239
Figure 69: Platoon-Tanks MRE 242
Figure 70: ADG for the JPSD Platoon-Tanks MRE. 251
Figure 71: JPSD Platoon-Tanks MRE 262
Figure 72: Platoon-Tanks MRE 265
Figure 73: ADG for the RPR Platoon-Tanks MRE 275
Figure 74: RPR Platoon-Tanks MRE 285
Figure 75: Marcus MRE. 288
Figure 76: ADG for the Marcus MRE 292
Figure 77: Marcus MRE. 298

There is no excellent beauty that hath not
some strangeness in the proportion.
— *Francis Bacon,* Of Beauty

List of Tables

Table 1: Evaluation of Domains employing MRM. 20
Table 2: Summary of Assumptions made by MRM approaches 76
Table 3: Comparison among MRM approaches 85
Table 4: Summary of Benefits of MREs. 88
Table 5: Summary of Limitations of MREs 92
Table 6: Comparison among MRM approaches 94
Table 7: Assigning Cumulative and Distributive Dependencies. 104
Table 8: Effects of an Interaction 117
Table 9: Example Concurrent Interactions 154
Table 10: Effects of Concurrent Interactions 155
Table 11: Example Attribute Relationship Table 165
Table 12: Example Concurrent Interactions Table 166
Table 13: Cost Comparison among MRM approaches. 180
Table 14: Object Class Structure Table for JTFp 219
Table 15: Attribute/Parameter Table for JTFp 220
Table 16: Attributes of Platoon, Tank_1 and Tank_2 (JTFp) 222
Table 17: Attribute Relationship Table for Platoon-Tanks MRE in JTFp 225
Table 18: Mapping Functions for JTFp Platoon-Tanks MRE. 228
Table 19: Object Interaction Table for JTFp. 230
Table 20: Effects of Interactions for JTFp Platoon-Tanks MRE 232
Table 21: Concurrent Interactions Table for JTFp Platoon-Tanks MRE 237
Table 22: Object Class Structure Table for JPSD 244

Table 23: Attribute/Parameter Table for JPSD 245
Table 24: Attributes of Platoon, $Tank_1$ and $Tank_2$ (JPSD) 248
Table 25: Attribute Relationship Table for Platoon-Tanks MRE in JPSD 252
Table 26: Mapping Functions for JPSD Platoon-Tanks MRE 253
Table 27: Object Interaction Table for JPSD 256
Table 28: Effects of Interactions for JPSD Platoon-Tanks MRE 257
Table 29: Concurrent Interactions Table for JPSD Platoon-Tanks MRE 261
Table 30: Object Class Structure Table for RPR 267
Table 31: Attribute/Parameter Table for RPR. 268
Table 32: Attributes of Platoon, $Tank_1$ and $Tank_2$ (RPR). 270
Table 33: Attribute Relationship Table for Platoon-Tanks MRE in RPR. 273
Table 34: Mapping Functions for RPR Platoon-Tanks MRE 276
Table 35: Object Interaction Table for RPR 278
Table 36: Effects of Interactions for RPR Platoon-Tanks MRE. 279
Table 37: Concurrent Interactions Table for RPR Platoon-Tanks MRE. 284
Table 38: Attributes of planner and PA (Marcus). 290
Table 39: Attribute Relationship Table for Marcus MRE. 291
Table 40: Mapping Functions for Marcus MRE. 294
Table 41: Interactions sent and received by the Marcus MRE 296
Table 42: Concurrent Interactions Table for Marcus MRE. 297

Education that consists in learning things and not the meaning of them is feeding upon the husks and not the corn. — Mark Twain

… the learned have the right and the duty to use an obscure language that is comprehensible only to their fellows. — Umberto Eco, The Name of the Rose

List of Symbols

a, b, v	Attribute
v_0, v_1, …	Value for attribute
δa, δv_1, δv_2	Change to an attribute
$<a, \delta a>$	Tuple of attribute a and change δa to a
P, Q, T, E_1, E_2, T_1, T_2, S_1, S_2, …	Entity
P, Q, R	Set of attributes
r	Relationship
$P \rightarrow Q$	Relationship between attribute sets P and Q
f, g	Mapping function
I, J, I_1, I_2, …	Interaction
$I.affects$	Effects determined by semantics of interaction I
$I.affects^+$	Effects of interaction I determined by dependencies
$I.affects^*$	Effects of interaction I
$E(I)$	Effects of interaction I
$I \bullet J$	Concurrent occurrence of interactions I and J
$E(I) \lozenge E(J)$	Applying $E(I)$ and $E(J)$ in some sequential order
$E(I)$, $E(J)$	Applying $E(I)$ before $E(J)$
t, t', t_i, t_{i+1}, t_j, t_{j+1}, t_0, t_1, …	Observation time, simulation time
$[t_i, t_{i+1}]$	Time-step
τ	Duration of time-step
$\Delta P(t_i)$	Changes to attribute set P during time-step [ti-1, t]
T, T^A, T^B, T^M	Sequence of observation times
Rep, Rep^A, Rep^B	Set of all attributes in a model
Rel, Rel^A, Rel^B	Set of all relationships in a model
Int, Int^A, Int^B	Set of all interactions in a model
$Model$, $Model^A$, $Model^B$	Model
$Level^A$, $Level^B$	Representation level
$Rep(t)$	State of a representation Rep at time t

$Rel(t)$	Relationships in *Rel* that hold at time t
$Int(t)$	Set of interactions in *Int* sent or received at time t
$Int(t)_k$	k^{th} interaction in $Int(t)$
$Model(t)$	*Model* at time t
$F(Rep(t), E(I))$	Function that applies $E(I)$ to state $Rep(t)$
RepSeq	Sequence of representation states
RelSeq	Sequence of relationships that hold
IntSeq	Sequence of set of interactions sent or received
Rep^M	Set of all attributes in a multi-model
Rel^M	Set of all relationships in a multi-model
$Rel^{cross\text{-}model}$	Set of cross-model relationships in a multi-model
Int^M	Set of all interactions in a multi-model
$Model^M$	Multi-model
$\oplus$	Vector sum
$\rightarrow$	Transition in a state diagram
`R(`a`)`	Read operation on attribute/variable a
`W(`a`)`	Write operation on attribute/variable a
Z_1, Z_2, Z_3	Schedule of operations

There are more things in heaven and earth, Horatio,
than are dreamt of in your philosophy.
— *William Shakespeare,* Hamlet

Chapter 1

Introduction

Integrating multiple, independently-developed models of overlapping phenomena is the crux of multi-model design. Experience shows that this integration is a hard problem because of semantic mismatches between the models, and because current approaches make sub-optimal trade-offs between run-time performance and consistency of the models. Current practice involves employing one of two basic approaches: selective viewing, which may compromise performance for consistency, and aggregation-disaggregation, which may compromise consistency for run-time performance. Often, the precise conditions under which integration can be done effectively are not entirely clear. This dissertation addresses the integration of independently-developed models in a manner that reconciles demands for run-time performance and consistency of the models when concurrent interactions occur.

In this dissertation, we present a new approach, `UNIFY`, for integrating multiple models. The main contributions of this work are two-fold. First, we show that `UNIFY` improves on the run-time performance of selective viewing while simultaneously

improving on the consistency provided by aggregation-disaggregation. In other words, it achieves a better balance of these competing concerns than either of the two approaches in use today. Our thesis is that multiple models can be integrated such that their joint execution is as consistent as selective viewing but with lower costs than selective viewing or aggregation-disaggregation. An approach that satisfies our thesis achieves "effective" joint execution. In this dissertation, we present requirements for effective joint execution, and show how current approaches fail to satisfy them. We define concurrent representations as the representations of jointly-executing models. We define maintaining consistency as ensuring that the states of representations do not conflict. Effective joint execution of multiple models can be achieved by maintaining consistency among their representations. Consistency maintenance among concurrent representations is the cornerstone of `UNIFY`.

The second contribution of this work is identifying the conditions under which multiple, independently-developed models can be integrated. This knowledge enables designers to produce models that can be composed later into a multi-model. Our work shows that existing models can be integrated only to the extent that they satisfy these conditions. The extent to which existing models do meet these conditions is not known, but the difficulty that designers experience in practice suggests that there is room for improvement. We present four fundamental observations about jointly-executing models, which form the basis of the techniques and processes that are part of `UNIFY`. We apply `UNIFY` to existing models and present guidelines for multi-model designers.

1.1 Background

Modelling is a method to study a phenomenon without involving the phenomenon itself. A model captures essential parts of a phenomenon, such as its constituent processes and interacting objects, which are called entities. Typically, models have representation, which is a means of describing objects and processes within a model. Simulation is a technique to execute models, typically on a computer. Modelling and simulation provide the opportunity to study a phenomenon relatively inexpensively, reproducibly and, by reducing the number of controlling factors, at a convenient level of abstraction.

Multiple models executing jointly may capture combined semantics that cannot be captured by any one model alone. Multiple models may be constructed in order to study different parts of a phenomenon. The multiple models together constitute a multi-model. When the multiple models execute at overlapping times and exchange information with one another, they are said to execute jointly. Davis and other researchers advocate jointly executing multiple models of a phenomenon [DAVIS93]. Constructing and maintaining a new model for every combination of semantics may involve high cost and effort on the part of designers. In contrast, simple and well-designed models executing jointly may be able to capture such semantics. Effective joint execution of well-designed models can lead to a multi-model that satisfies its users' requirements. Constructing well-designed models is an important task; however, we restrict our work to the effectiveness of joint execution. Multi-representation modelling (MRM) is the joint execution of multiple models of the same phenomenon.

Currently, two basic approaches exist for executing multiple models jointly: selective viewing and aggregation-disaggregation. We explain these approaches with a brief

example. Consider a chemical reaction for which two models exist: a molecular model, $Model^A$, and an atomic model, $Model^B$. Since the reaction can be studied from both perspectives, it may be required for the models to execute jointly. In the selective viewing approach, the more detailed model of the two, in this case $Model^B$, is executed alone. The execution of $Model^A$ is emulated by selecting a view, i.e., filtering information, from the state of $Model^B$ when necessary. Since only $Model^B$ is executed, maintaining consistency between the two models is straightforward. However, since $Model^B$ is more detailed, executing it entails higher resource consumption than $Model^A$. Moreover, since $Model^A$ is not executed, selective viewing does not capture the combined semantics of the two models. In the aggregation-disaggregation approach, as far as possible, the less detailed model of the two, in this case $Model^A$, is executed alone. When more detail is required, the execution of $Model^A$ is suspended and $Model^B$ is executed. Subsequently, when detail is not required, the execution of $Model^B$ may be suspended and the execution of $Model^A$ resumed. Since $Model^A$ is less detailed, executing it entails lower resource consumption cost than $Model^B$. Since the currently-executing model may change at different times, maintaining consistency between the models is important for maintaining semantic continuity. However, for reasons that we will explain in §4.1, maintaining consistency in this manner is hard, i.e., it can be error-prone and resource-intensive. Moreover, aggregation-disaggregation does not capture the combined semantics of the two models because at any given time only one of the models is executed.

Both these approaches are based on sound principles; however, we show that they can fail to achieve effective MRM in many instances [REYN97]. Selective viewing is based on the principle that high detail is more important than performance or abstraction.

Aggregation-disaggregation is based on the principle that performance and abstraction can be gained by providing high detail infrequently. Rigidly adhering to one principle or the other has caused MRM to become ineffective in many instances. With our approach, UNIFY, we show how to balance these competing principles, thus eliminating problems inherent in alternative approaches.

1.2 UNIFY — An Overview

UNIFY is a framework for maintaining consistency among representations of jointly-executing models. It is based on four fundamental observations that capture general characteristics of jointly-executing models [REYN97]. Current MRM approaches encounter a number of problems because they have failed to appreciate these characteristics. The fundamental observations indicate that consistent MRM can be achieved at a lower cost than other approaches by maintaining consistency among multiple representations when concurrent interactions occur.

When multiple models of the same phenomenon execute jointly, significant problems can arise if the models conflict. Eliminating or avoiding these conflicts has been a hard problem for MRM designers. Some multi-models may satisfy their users' requirements despite such conflicts. However, we believe that maintaining consistency among the representations of a multi-model is a systematic and disciplined approach for constructing multi-models that satisfy their users' requirements. This approach benefits multi-models that require consistency, as we will show in the rest of this dissertation. This approach benefits multi-models that tolerate relaxed or no consistency as well, because it shows how other requirements for effective joint execution can be satisfied.

We avoid problems encountered in current approaches by making multiple representations of an entity co-exist at all times within a Multiple Representation Entity (MRE). For example, in Figure 1, E_1 is an MRE for an entity in multiple models, $Model^A$ and $Model^B$. An MRE is a contrast to selective viewing and aggregation-disaggregation, wherein either the entity in $Model^A$ or the entity in $Model^B$, but not both, would exist at any given time. As we will show in §2.1, designers in many domains, such as multi-resolution graphical models, hierarchical autonomous agents and molecular modelling have adopted approaches similar to creating MREs. MREs, a part of UNIFY, maintain the representations of multiple models at all times.

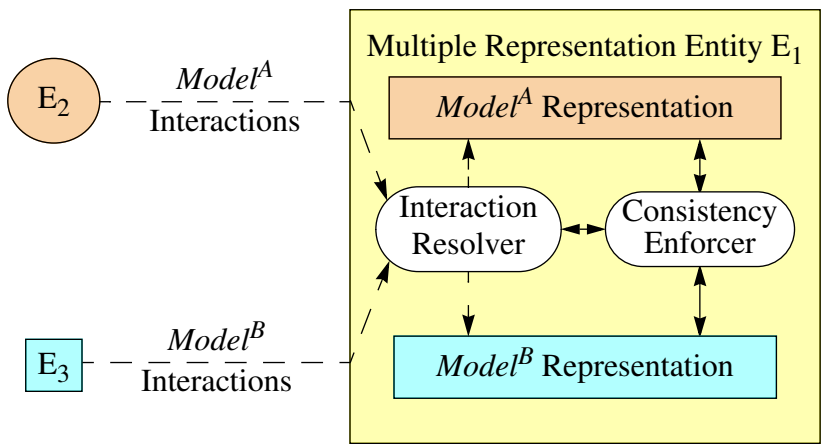


FIGURE 1: Our Approach to MRM

An MRE permits concurrent changes to any of its representations. Changes to states of representations occur as a result of interactions among objects and processes. Interactions are means by which objects and processes communicate or try to influence the behaviour of one another. Interactions may change multiple representations of objects or processes.

One challenge in UNIFY is maintaining consistency among multiple representations when the state of any representation changes. In Figure 1, when E_1 interacts with either E_2 or E_3, the multiple representations within E_1 must be consistent. A Consistency Enforcer

(CE) maintains consistency among the multiple representations within an MRE by capturing relationships among parts of the representations. A CE consists of an Attribute Dependency Graph (ADG) and mapping functions. An ADG captures dependencies among representations. When the state of a representation changes, a CE traverses an ADG to determine how the state of other representations must change. The CE performs the actual changes by invoking application-specific mapping functions that translate changes in one attribute to changes in others. As part of `UNIFY`, we show how to construct an ADG, select mapping functions and construct a CE for an MRE.

Another challenge in `UNIFY` is ensuring that the effects of interactions that occur at overlapping simulation times are applied correctly. For example, in Figure 1, E_1's behaviour must be meaningful even when E_2 and E_3 interact with E_1 *concurrently.* Interactions occurring at overlapping times are called concurrent interactions. Concurrent interactions may be dependent, i.e., have related effects, for example, precluding or enhancing one another. Traditionally, concurrent interactions have been serialized, i.e., applied one after another in some arbitrary order. However, serialization can cause incorrect behaviour because the effects of dependent concurrent interactions are not reflected meaningfully. For example, in a model of a chemical reaction, E_1 may represent some quantity of an acid, E_2 may represent a reagent and E_3 a catalyst. Adding a reagent or catalyst is an interaction in this model. When both E_2 and E_3 are added to E_1, the rate of the reaction may increase more than the sum of the increases caused by adding either E_2 or E_3 alone. Serialization can capture the sum of the increases, but not the increase caused when both E_2 and E_3 are added. Therefore, in this model, serialization produces incorrect results. As part of `UNIFY`, we present a taxonomy of interactions that captures the

semantic relations among concurrent interactions and presents mechanisms to resolve them. Also, we show how to construct an Interaction Resolver (IR) for an MRE in order to resolve the effects of dependent concurrent interactions.

MREs, ADGs, a taxonomy of interactions, and processes for constructing a CE and an IR are part of `UNIFY`. Multi-model designers can achieve consistent MRM at lower cost than other approaches by applying these techniques and processes.

1.3 Requirements for Effective MRM

In jointly-executing models, entities in all models must interact consistently and cost-effectively. Although these requirements seem self-evident, alternative techniques for MRM often fail to satisfy them. We measure the success of an MRM approach by analysing whether the approach satisfies these requirements.

Often, multi-models are unsatisfactory because they become inconsistent or expensive. An effective MRM approach must satisfy at least the following reasonable requirements:

R1: **Multi-representation Interaction**: Entities in each model may initiate and receive interactions that may cause changes to the entities concurrently. Dependent concurrent interactions must be permitted.

R2: **Multi-representation Consistency**: The representations of jointly-executing models must be consistent with one another. *Temporal consistency* requires that two entities interacting with a third entity at overlapping simulation times have consistent views of the third entity. *Mapping consistency* requires that entity properties common to different models be translated such that repeated translations in a given period do not cause abnormal behaviour in the entity

during that period. Multi-representation consistency is interesting only if the multiple models are related to one another.

R3: **Cost-effectiveness**: The costs of simulating multiple models and maintaining consistency among them should be lower than alternative approaches.

These requirements represent the conditions under which multiple models can be integrated effectively. We will evaluate `UNIFY` and alternative MRM approaches such as aggregation-disaggregation and selective viewing with regard to these requirements. We will consider an MRM approach sufficient only if it satisfies all three requirements.

1.4 Claims and Contributions

`UNIFY` benefits the practice of modelling and simulation because it enables designers to build consistent multi-models with lower run-time costs than other approaches. We have examined the problem of joint execution of multiple models in detail, and created general and useful techniques for consistency maintenance in concurrent representations. Rather than conceiving a detailed solution for every application we analysed, we concentrated on a process that MRM designers may modify for their applications.

We present a sufficient and practical framework for MRM. Our framework, `UNIFY`, is a sufficient approach to MRM because it satisfies the three requirements for MRM: R1, R2 and R3. Moreover, `UNIFY` is a practical approach to MRM because it can be applied in conjunction with a methodology for specifying models.

The major contributions of our work are the fundamental observations, MREs, ADGs, the taxonomy of interactions, a cost study and the guidelines for designers. All of these contributions further the existing practice in modelling and simulation. The taxonomy of

interactions offers a spectrum of solution choices for resolving concurrent interactions in any domain. We expect UNIFY to be useful in a variety of domains, such as hierarchical autonomous agents, climate modelling and graphical modelling.

A substantial benefit of our work to multi-model designers is a set of guidelines for consistency maintenance. The guidelines lead designers from their joint execution requirements to the design of consistent MREs. We augment Object Model Templates — a methodology for specifying objects and the interactions among them — with specifications for concurrent interactions. Incorporating UNIFY into an existing specification methodology enables designers to understand our work in terms of techniques already familiar to them. The guidelines provide designers with an easy reference for incorporating consistency maintenance in their models.

Our work will benefit many modellers. Designers may incorporate consistency in their applications by following our guidelines. Analysts may examine the justification behind the guidelines. Our analyses of MRM approaches cautions modellers entering the field of MRM: joint execution is neither trivial nor easy. Finally, we have laid the foundation for future explorations and refinements.

1.5 Evaluation

In this dissertation, we show how multi-model designers can achieve consistency similar to selective viewing but at a lower cost than either selective viewing or aggregation-disaggregation. We show how designers can employ an ADG, mapping functions, a taxonomy of interactions and policies for concurrent interactions in order to maintain consistency within an MRE when concurrent interactions occur. We measure the

costs involved in executing multiple models jointly and show that these costs are lower in UNIFY than in selective viewing and aggregation-disaggregation. We show how UNIFY satisfies the requirements for effective MRM, R1, R2 and R3, while other approaches do not even if they make similar assumptions as we do. We apply the techniques and processes in UNIFY to four multi-models in order to provide empirical evidence that UNIFY is a practical framework. Finally, we show how UNIFY can be applied in conjunction with Object Model Template, a methodology for specifying multi-models.

The three MRM requirements, R1, R2 and R3, capture desirable goals for the joint execution of multiple models. Satisfying these requirements supports our thesis that a multi-model can be constructed in a consistent manner and with reduced run-time costs. The requirements themselves do not outline an approach for effective MRM. In other words, approaches other than UNIFY for achieving effective MRM are possible. Finally, the requirements may be part of a larger set of requirements for the joint execution of multiple models. Identifying the members of the larger set is a topic for future work.

Our work presents a general approach for effective MRM. We do not address how effective MRM can be achieved for specific models. However, we do present the conditions under which such models may be executed jointly. We provide techniques and processes that designers can employ to satisfy most of these conditions. We have been unable to provide techniques for achieving compatible time-steps (discussed in §3.3.3). Although this inability is a limitation of our work, we show how compatible time-steps eliminate inconsistencies caused by time-step differentials, thus benefiting multi-model designers. We regard our work as a preliminary step towards a detailed framework that

guides designers in the design of their multi-models. We expect that UNIFY, with future additions, will be that framework.

We envision designers routinely constructing simple models that can be integrated and jointly executed as a multi-model. A number of issues must be resolved before this vision becomes reality. For example, constructing models can be complex, verifying them can be difficult and reconciling the semantic differences between them can cause problems. Our work addresses only one of these important issues: the joint execution of the models. Our work focusses on consistency maintenance in concurrent representations. We show that maintaining consistent representations for multiple models that execute jointly leads to effective MRM.

1.6 Outline

In Chapter 2, we briefly present applications that adopt the approach of maintaining concurrent representations. Detailed discussions of these applications are in Appendix A. We present alternative MRM approaches wherein concurrent representations are not maintained. Also, we present work related to key concepts in UNIFY. In Chapter 3, we lay the foundation for UNIFY by introducing and defining terms that we will use throughout this dissertation. Also, we discuss the criteria that we will use to evaluate MRM approaches. In Chapter 4, we present and justify some fundamental observations about MRM. These observations arise from empirical studies of many MRM applications. Any solution to the MRM problem must incorporate these observations. Our approach recommends maintaining consistency among concurrent representations of multiple models. We present our framework-based approach to MRM, UNIFY, in Chapter 5 and

discuss the technical challenges with such an approach. In Chapter 6, we address the first challenge — keeping multiple representations consistent when interactions change any representation. In Chapter 7, we address the second challenge — resolving the effects of concurrent interactions. In Chapter 8, we present a process for applying the techniques that are part of `UNIFY` and present guidelines for designers of multi-model applications. In Chapter 9, we evaluate `UNIFY` and briefly present case studies of applying it. We present the case studies in detail in Appendices B, C, D and E. We conclude in Chapter 10 by discussing the contributions of our work to the practice of modelling and simulation and presenting some areas for future work.

The vision of one man lends not its wings to another man.
— *Kahlil Gibran,* The Prophet

Chapter 2

Related Work

Multi-Representation Modelling (MRM) — the joint execution of different models of the same phenomenon — has been explored in applications in a number of domains, from multi-resolution graphics and battlefield simulations to climate models and molecular models. In most of these domains, MRM has proven beneficial for some applications no matter what MRM approach has been used. In §2.1, we present example applications that employ multi-models. In §2.2, using examples from battlefield simulations, we describe alternative MRM approaches, wherein all but one model may suspend execution. In §2.3, we describe work that has influenced our approach.

2.1 MRM Applications

We present a sampling of domains in which MRM in some form has been employed. For these domains, MRM has been considered beneficial for many applications. A detailed discussion of domains employing MRM is in Appendix A along with evaluations of whether the MRM approaches satisfy R1, R2 and R3.

2.1.1 Multi-Resolution Graphical Modelling

In multi-resolution graphical modelling, the system maintains multiple representations, or *levels of detail*, of an object and renders the appropriate representation depending on the object's distance from the viewer [CLARK76]. Coarser levels of detail for an object employ fewer polygons, thus reducing the time required to render the object. Moreover, coarse levels of detail depict the object satisfactorily when the perceived size of the object relative to the viewing area is small, for example, when the object is distant from the viewer. In multi-resolution graphical models, researchers concentrate on generating levels of detail automatically before run-time; at run-time, an appropriate level is selected for visually-appealing rendering [GAR95] [HECK94] [HECK97] [LUEBKE97] [PUPPO97]. A few applications permit a user to change a level of detail at run-time, thus requiring re-generation of other levels of detail [BERM94] [LEE98] [ZORIN97].

2.1.2 Hierarchical Autonomous Agents

Hierarchical autonomous agents jointly execute multiple layers (e.g., a deliberative layer [SACER74] and a reactive layer [AGRE87]) in order to utilise the capabilities of each layer [ALBUS97] [BON97] [FIRBY87] [GAT92] [HANKS90] [LAIRD91] [SIM94] [WAS98A]. Multiple layers enable an agent to pre-plan some of the steps required to fulfill its goal yet exhibit robust behaviour when unexpected or urgent situations occur. Usually, each layer maintains representation about the agent's goal or surroundings [BROOKS86] [BRILL98]. Eliminating inconsistencies among dependent parts of the representations for multiple layers is an open issue.

2.1.3 Blackboard Systems

In blackboard systems such as `Hearsay-II`, many processes write to and read from a single data structure, called a blackboard [ERMAN80]. `Hearsay-II` translates spoken sentences into the corresponding alphabetic representation. `Hearsay-II`'s blackboard is a multi-model; each layer is a different model of a spoken sentence. Layers corresponding to sentence fragments such as phonemes, words and phrases execute jointly to produce multiple interpretations of one sentence. Each interpretation is a consistent view of the sentence. Multiple interpretations are ranked by a credibility metric; the most credible interpretation is the best translation of the spoken sentence. However, maintaining multiple interpretations of a sentence is resource-intensive.

2.1.4 Cache Coherence

In a multi-processor configuration, each processor may access a fast local cache in order to reduce accesses to slow main memory. Processors may read and modify copies of main memory data stored in their caches. Ensuring that processors access correct versions of cached data is the cache coherence problem [HENN96] [ARCH86]. The main memory copy and each cache copy of a datum are concurrent representations of a variable. Processes issue interactions in the form of read and write operations to any copy. Caches and main memory copies bear simple relationships, such as equality, with one another.

2.1.5 Abstract Data Types and Object Inheritance

In polymorphic languages, data may have multiple types [CARD85]. Some languages support *ad hoc* polymorphism, wherein a datum may be defined multiply, e.g., a union

[KERN88] [STROU91] or a perspective [GOLD80] [STEFIK86]. Unions and perspectives permit one representation of a datum to be viewed in different ways. Unions and perspectives are not expressive enough to capture relationships among multiple representations. Object-oriented languages such as Smalltalk-80 [BORN82], Simula-67 [DAHL66] [BIRT73] and C++ [STROU91] support inclusion polymorphism, wherein a datum may belong to different classes. The languages rely on the typing mechanism and the contexts in which the datum is used to determine its class. Object-oriented languages capture limited relationships, such as inheritance, among parts of representations.

2.1.6 Views in Databases and Integrated Environments

In relational database applications, data are abstracted into relations, which essentially are tables whose rows are tuples and columns are values for members of tuples [CODD70] [ASTRA76] [STONE76] [LINTON84]. In object-oriented databases, data are abstracted as relationships among entities [CHEN76] [BALZER85]. A view in a database is derivative, i.e., the view is a set of relations derived from existing relations or relationships [CHAM75]. A view in an integrated environment is constructive, i.e., the database is constructed from individual views [GAR87]. Changes to a view must be translated to changes in the database [BAN81] [HOR86].

2.1.7 Nested Climate Modelling

In nested climate modelling, Limited Area Models (LAMs), which predict regional climate, execute jointly with Global Circulation Models (GCMs), which predict wide-ranging climate changes. The joint execution produces more accurate predictions of the weather than either alone [GIORGI90] [GIORGI91] [RISBEY96]. Typically, GCM data for

large geographic areas are translated to LAM input. LAMs supplied with this input data perform further computations to predict weather for small geographic areas. Ideally, LAM data should be translated to GCM input as well in order to account for local factors that may influence global climate. However, translating GCM data for LAM input is common, but the reverse translation is an open problem.

2.1.8 Integrated Molecular Modelling

When theoretical studies on the potential energy surfaces for chemical reactions of a large system are carried out, low-computation low-detail models, such as molecular mechanics models, are used initially for most of the system, and high-computation high-detail models, such as molecular orbital methods, are used subsequently for a small part of the system [MATSU96] [SVEN96A] [HUMBEL96] [SVEN96B]. Such integrated models enable researchers to study interesting aspects of a reaction in detail without incurring the cost of modelling the entire reaction in detail. Integrated molecular models permit interactions at multiple levels and are remarkably consistent with one another. Also, reported resource consumption is low.

2.1.9 Multi-Level Computer Games

In a number of commercial computer games, players control characters inhabiting a world displayed at multiple resolutions. Usually, a player interacts at the most detailed resolution level, with the other resolution levels existing solely to provide the player with a wider or less-cluttered view of the game world. In a few games, the player may transition to less-detailed resolution levels and interact at those resolution levels. Typically, players

can interact at only one resolution level at a time. In most games, all processing takes place at the most detailed resolution level.

2.1.10 Battlefield Simulations

A number of battlefield simulations require the joint execution of multiple models, for example, training models and analysis models [AMG95] [DAVIS93] [DAVIS98] [DIS93] [DOD94] [REYN94]. Typically, battlefield simulations employ an approach called aggregation-disaggregation to ensure that entities interact at the same representation level. Aggregation-disaggregation enables many independently-designed simulations to execute jointly. However, aggregation-disaggregation scales poorly with large numbers of jointly-executing models or interacting entities; it can preclude concurrent multi-representation interactions, give rise to inconsistencies among the multiple representations, and increase resource consumption.

2.1.11 MRM Applications Summary

In Table 1, we evaluate the MRM approaches employed in the above domains with regard to our MRM requirements of multi-representation interactions (R1), multi-representation consistency (R2) and cost-effectiveness (R3). The evaluation here is intentionally brief; it is meant to highlight shortcomings of previous work. Detailed evaluations of these domains are in Appendix A. In Table 1, darkly-shaded cells signify that a domain satisfies a requirement. Lightly-shaded cells signify that a domain satisfies a requirement poorly. Unshaded cells signify that a domain does not satisfy a requirement. An ideal MRM approach for each domain will have all three cells shaded darkly.

TABLE 1: Evaluation of Domains employing MRM

Domain	R1	R2	R3
Multi-Resolution Graphical Modelling			
Hierarchical Autonomous Agents			
Blackboard Systems			
Cache Coherence			
Abstract Data Types and Object Inheritance			
Views in Databases and Integrated Environments			
Nested Climate Modelling			
Integrated Molecular Modelling			
Multi-Level Computer Games			
Battlefield Simulations			

2.2 Multi-Model Execution

MRM approaches such as selective viewing and aggregation-disaggregation execute only one model at a time. In *selective viewing*, only the most detailed model is executed. In *aggregation-disaggregation*, at any given time, only one model is executed; depending on the interactions among entities, the system may change the currently-executing model by transitioning among models. In *Variable Resolution Modelling*, processes are modelled at different resolution levels. At any time, a user may choose to model processes or sub-processes at higher or lower detail. The system transitions among multiple process models in order to satisfy the user's request. In the following sections, we critique each approach briefly. Most of the examples in these sections are from battlefield simulations because of our experience and familiarity with that domain.

2.2.1 Selective Viewing

With selective viewing, only the most detailed model is executed, and all other models are emulated by selecting information, or views, from the representation of the most detailed model [DAVIS93]. Selective viewing is employed when modelling a phenomenon in detail at all times is considered necessary. Low-resolution views of a multi-model are generated from the most detailed model. While this approach may be suitable for games because available processing resources can execute the most detailed model at near-real-time, for more complex models, selective viewing has many disadvantages.

First, executing the most detailed model incurs the highest resource usage cost. Proponents of selective viewing may argue that the smallest detail can affect the execution of the complete model (e.g., a butterfly flapping its wings in Columbia can affect the weather of Western Europe). While this argument may be valid in some cases, for most models, most of the details can be abstracted reasonably in order to conserve resources.

Second, the most detailed model is likely to be the most complex model. One of the main benefits of modelling is to make reasonable simplifications in order to study a phenomenon efficiently. Executing the most detailed model adds complexity instead of reducing it.

Third, executing the most detailed model may limit the opportunities for performing some types of analyses. Abstract models enable a user to make high-level decisions regarding the multi-model. These high-level decisions are likely to change the behaviour of many entities, thus enabling broad analyses of the multi-model. Enabling equivalent analyses in a detailed model requires making corresponding low-level decisions. These

low-level decisions may not exist or may be difficult to make. Thus, the equivalent analyses in a detailed model may be impossible or infeasible.

Fourth, some multiple models may not bear hierarchical relationships with one another, i.e., none of them is the most detailed model. Selective viewing implies that the most detailed model is a monolithic model. For non-hierarchical models, the monolithic model must be created by capturing all the details of all the models. Such a monolithic model requires additional design effort and is likely to be very complex.

The philosophical question of what is the most detailed model can entrap designers into adding ever-increasing detail to a model by refining entities in the model increasingly. However, even assuming a designer can escape this trap eventually, selective viewing is not suitable for the execution of a multi-model because of the above disadvantages.

2.2.2 Aggregation-Disaggregation

Inconsistencies can arise in a multi-model when a low resolution entity (LRE), e.g., corp, interacts with a high resolution entity (HRE), e.g., tank. A common MRM approach is to change the resolution of an entity dynamically to match the resolution of other interacting entities. This dynamic change is called *aggregation* (HREs $\rightarrow$ LRE) or *disaggregation* (LRE $\rightarrow$ HREs). Aggregation-disaggregation ensures that entities interact with one another at the same level by forcibly changing their representation levels [SMITH94]. Typically, if an LRE interacts with an HRE, the LRE is disaggregated into its constituents, which interact at the HRE level. LRE-LRE interactions would be at the LRE level. A disaggregated LRE may be re-aggregated so that it can interact subsequently at the LRE level. We critique the variations on aggregation-disaggregation [NAT96].

2.2.2.1 Full Disaggregation

Full disaggregation involves disaggregating an LRE into its constituent HREs. In Figure 2, LREs L_1 and L_2 are disaggregated when they interact with an HRE. Typically, full disaggregation occurs when an LRE establishes contact (e.g., sensor, line-of-sight) with an HRE. Full disaggregation ensures that all entities interact at the same representation levels. However, full disaggregation is often too aggressive — although only some HREs that constitute an LRE may be involved in a particular interaction, all the constituent HREs will be disaggregated. Moreover, full disaggregation leads to chain disaggregation — cascading disaggregation of interacting LREs when one of them interacts with an HRE (e.g., the disaggregation of LRE L_3). The large number of entities instantiated in full disaggregation may place a high demand on system resources. Accordingly, full disaggregation is restricted to small-scale multi-models [CALD95A].

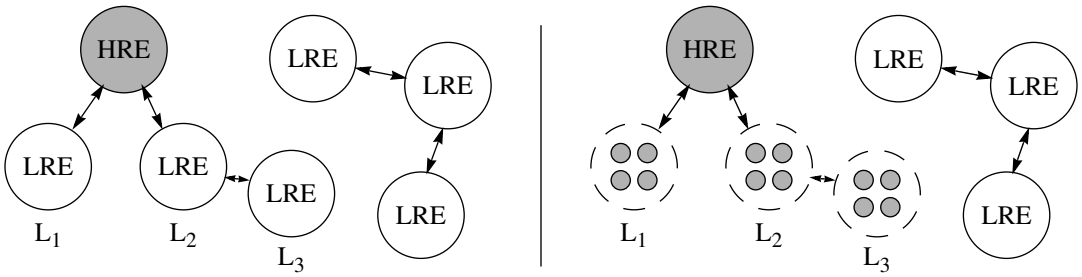


FIGURE 2: Full Disaggregation

2.2.2.2 Partial Disaggregation

Partial disaggregation attempts to overcome the main limitations of full disaggregation by disaggregating an LRE partly rather than entirely. As seen in Figure 3, a partition is created inside LRE L_2 such that only a part of L_2 is disaggregated into HREs that interact with the disaggregated constituents of LRE L_1; the remaining part of L_2 is left as an LRE to interact with LRE L_3. For example, in the `BBS/SIMNET` [HARDY94] [BURD95]

linkage, a BBS entity that engages a SIMNET entity is partitioned such that one part disaggregates and fights a disaggregate-level battle in the SIMNET world, while the other part remains aggregated and fights aggregate-level battles in the BBS world.

As seen in Figure 3, partial disaggregation has the potential to control chain disaggregation. This potential depends on how easily a partition can be constructed inside an LRE. The criteria for constructing the partition must be chosen carefully to prevent partial disaggregation from degenerating into full disaggregation.

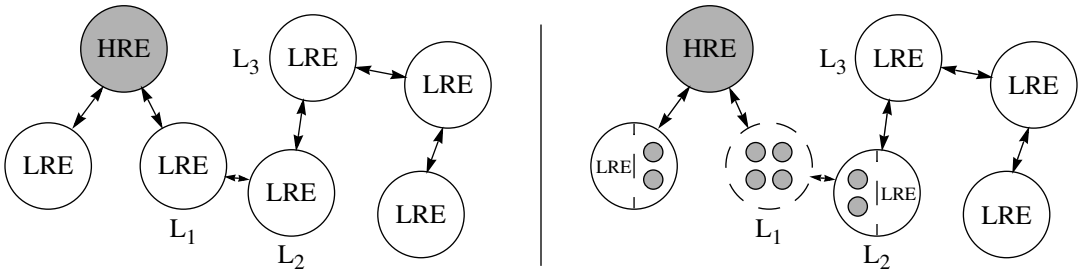


FIGURE 3: Partial Disaggregation

2.2.2.3 Playboxes

A common aggregation-disaggregation variant is to demarcate a pre-determined region of the simulated domain, called a *playbox*, within which only HREs can participate [KARR94]. Conceptually, the playbox may be defined in any domain, for example, a spatial domain such as a simulated battlefield. Entities inside the playbox are disaggregated while those outside remain aggregated. An LRE that crosses into the playbox must be disaggregated; likewise, when all the disaggregated constituent entities of an LRE leave the playbox, they are aggregated into the LRE. The playbox is typically static in terms of location and boundaries, although it can be dynamic.

Playboxes may force entities to disaggregate unnecessarily, for example, when an entity enters a playbox but does not interact with others in the playbox (e.g., LRE L_2 in

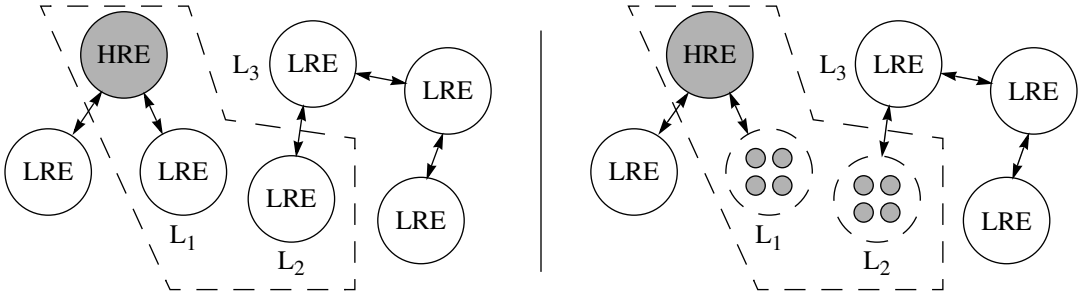


FIGURE 4: Playbox

Figure 4). Furthermore, thrashing can occur when the trajectory of an entity causes it to enter and leave the playbox rapidly. Cross-level interactions across the boundary of the playbox (e.g., interactions between the disaggregated L_2 and LRE L_3 in Figure 4) must be addressed separately. Additionally, static playboxes artificially constrain the region in which LREs and HREs may interact meaningfully. Projects that use playboxes are `Eagle/BDS-D` [STOBER95], `Abacus/ModSAF` [COX95] and `AIM` [SEIDEL95].

2.2.2.4 Pseudo-Disaggregation

Consider a situation where an HRE requires the attributes of the constituent HREs of an LRE but does not interact with them. For example, an Unmanned Airborne Vehicle (UAV) may obtain aerial pictures that are processed for details of entities observed in an area. Since LREs are a modelling abstraction, any LRE in the UAV picture must be depicted as its constituent HREs. In this case, disaggregating the LRE is wasteful since only a perception of the constituent HREs is required. In pseudo-disaggregation, an HRE receives low-resolution information from LREs and *internally* disaggregates the information to obtain high-resolution information. For example, in Figure 5, the UAV is an HRE that pseudo-disaggregates LREs L_1 and L_2. Pseudo-disaggregation is applicable when the interaction is unidirectional, i.e., L_1 and L_2 do not interact with the UAV. The algorithms used by the UAV to disaggregate L_1 and L_2 locally must be similar to the ones

L_1 and L_2 would use to disaggregate themselves, if required. Each HRE must incorporate rules to disaggregate every LRE in the simulation. Pseudo-disaggregation is employed by `JPSD CLCGF` [CALD95B], `TACSIM/CBS` [SMITH95], `Eagle/BDS-D` [STOBER95], `ALSP` [WEAT93] and others [ALLEN96].

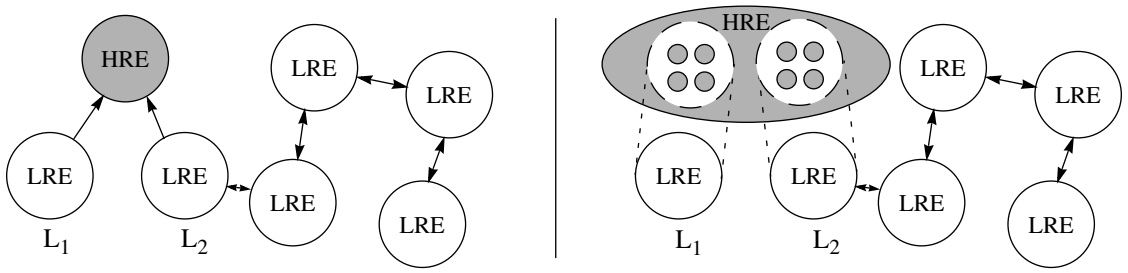


FIGURE 5: Pseudo-disaggregation

2.2.3 Variable Resolution Modelling

In Davis's Variable Resolution Modelling (VRM), designers construct families of models that support dynamic changes in resolution [DAVIS92] [DAVIS93]. For example, a coarse model of weather prediction may include season and geographical location. A model at a finer resolution may include temperature variations, cloud patterns and wind directions. A model at yet finer resolution may include rates of temperature changes, range of temperatures and so on. Designing with VRM in mind facilitates the construction of models that can execute at any desired level of resolution.

VRM involves building tunable process hierarchies, while MRM involves making multiple models execute jointly. It is possible for a simulation to incorporate both philosophies. For example, in a multi-resolution simulation, various aggregate-level and disaggregate-level entities may interact with one another. Users may vary the resolution at which the simulation proceeds. There are two aspects to this variability: one, the interactions among entities, which is our focus, and two, the resolution of simulation

processes, which is Davis's focus. We address issues that arise when aggregate-level entities interact with disaggregate-level entities. Davis addresses issues that arise when one wishes to observe phenomena such as invasions or stratagems at variable resolution. Designers may describe the movement of a single tank either by a very high-level process or by low-level sub-processes that involve factors like fine-grained terrain conditions and availability of fuel. Here, the motion of the tank is a VRM process, but the interaction of the tank with other tanks or platoons is an MRM issue.

VRM is related to MRM because a process at multiple resolution levels is likely to require multiple representations. Many VRM researchers argue for the existence of multiple resolutions [DAVIS98] [HARSH92] [HILL92A] [HILL92B] [HORR92]. However, in VRM, users are expected to transition among models during execution rather than execute multiple models concurrently. VRM complements MRM; the relationships among hierarchical resolution levels for a process are mapping functions that translate attributes among multiple representations.

2.3 Maintaining Consistency among Concurrent Representations

We present UNIFY briefly in order to discuss work that has influenced our approach to MRM. UNIFY includes the concept of a Multiple Representation Entity (MRE) which is a technique to maintain concurrent representations based on four fundamental observations about MRM [REYN97]. MREs are internally consistent and interact at multiple representation levels concurrently. A Consistency Enforcer (CE) consisting of an Attribute Dependency Graph (ADG) and application-specific mapping functions maintains

consistency among multiple representations in an MRE. An Interaction Resolver (IR) based on our taxonomy of interactions resolves the effects of dependent concurrent interactions [NAT99]. MREs reduce simulation and consistency costs [NAT97].

Determining whether a multi-model is satisfactory is ultimately a form of the Turing test [TURING50] because only end-users can determine whether the multi-model meets their requirements. Crucial to a multi-model is the effective joint execution of its constituent models. We believe effective joint execution can be achieved by maintaining consistency among concurrent representations. Consistent concurrent representations enable consistent concurrent behaviour since behaviour is influenced by state [HOP79]. Approaches like Temporal Logic of Actions support the notion that behaviour is influenced by state [LAM94] [ABADI95]. The definition of consistency is application-dependent. For some applications, consistency may be bi-modal (i.e., the representations are consistent or inconsistent), whereas for others it may be multi-modal (i.e., the representations are consistent to some degree). For yet other applications, consistency may be similar to determining the effectiveness of a real-time system that schedules tasks according to their deadlines and their expected values [BURNS98].

Dependency graphs similar to our ADGs have been used to capture cause-effect relationships in Petri Nets [PETER77] [PETRI62], dataflow models [DENNIS80] [ACK82] [DAVIS82] [GAJSKI82] [GRIM93], object-oriented design [RUM91] [SHLAER92] and logical time systems [LAM78]. Since attribute relationships can be viewed as constraints [ALLEN92] [HILL92A] [HORR92], a CE may be implemented as a constraint solver. Typically, a constraint solver operates in the Herbrand universe [JAFFAR94] [SARAS91]. Although constraint solving in the Herbrand universe can be complex [FRÜH92A]

[FRÜH92B] [VAN96], constraint solving in other domains can be simplified [MARR93] [GARCÍA93] [FREE90] [JAFFAR92] [CORMEN89]. A CE may be implemented as a set of mediators. The relationships among attributes at multiple representation levels may be realized by mediators, which capture behavioral relationships in complex systems [SULL94]. A CE may be implemented as an attribute grammar, which is a means of propagating changes among dependent attributes [KNUTH68] [KNUTH71] [REPS84] [BESH85] [DEMERS85] [REPS86] [HOR86].

Interactions are common in many domains, for example, database transactions and operations [ESWA76]; processor interrupts; cache operations [HENN96]; reads and writes to shared memory in parallel processing systems; operations, events and actions in object-oriented and process modelling [RUM91] [SHLAER92] [ALHIR98]; method invocations and function calls in object-oriented systems; messages in distributed processing systems and logical time systems [LAM78]; accesses to a blackboard [ERMAN80]; and exceptions in programming languages [GOOD75] [BARNES80] [LISKOV79] [STROU91] [YEMINI85]. Resolving the effects of interactions, transactions, events or operations that overlap in time is a well-known problem. The effects of concurrent interactions in MRM are similar to race conditions. In both cases a *laissez-faire* approach can lead to unpredictable, and often incorrect, effects. Many synchronisation primitives have been proposed to eliminate race conditions, such as locks, semaphores, barriers and monitors [MAD74] [SILB91] [TANEN92] [BRINCH78]. These primitives lead to policies that resolve concurrence by curbing it, i.e., concurrent operations are transformed into non-concurrent operations even if they should not be transformed this way.

A traditional policy for resolving concurrent events, operations, transactions or interactions is serialization — imposing an order on them [ESWA76] [HAER83]. Serialization is often a valid policy when the concurrent events or transactions are logically independent. Traditionally, database systems serialize independent transactions [BERN81] [PAPA86] [BRAHMA90]. Cache coherence models also serialize independent operations on cache blocks [HENN96] [ARCH86]. Object and process modelling techniques either require that one action execute in a state at a time or recommend partitioning the states in which concurrent events can occur and then reflecting the effects of those events simultaneously [ALHIR98] [RUM91] [SHLAER92]. Either approach assumes the concurrent events are independent. In logical time systems such as Lamport time [LAM78], virtual time [JEFF85], vector clocks [MATT89], PDES [FUJI90] and isotach systems [WILL93], independence is tied to a notion of concurrence, i.e., two events are assumed independent if it cannot be determined that there exists a cause-effect relationship among them.

The effects of some concurrent interactions may not be captured by any serial order. For example, the semantics of one interaction may interfere with the semantics of another interaction such that one or the other or both may be fully or partially excluded, ignored, delayed or even enhanced. Some database schemes utilise semantic information about transactions to reorder concurrent transactions, possibly non-serializably [BADRI92] [BARG91] [GARCIA83] [KORTH88] [LYNCH83] [MUNSON96] [WEIHL88] [THOM98]. However, even these approaches assume that the interactions are logically independent. Some concurrent interactions may be logically dependent, i.e, their *concurrent* occurrence

is a factor in determining their effects. We classify such interactions and evaluate our approach based on criteria for a good taxonomy [AMO94] [HOW97].

After considering specification methodologies such as DFDs, PERT charts, IDEF0-3, UML [ALHIR98] [FOWLER97] [TEXEL97], OOA [SHLAER92] and Rumbaugh's Object Modelling Techniques [RUM91], we chose the High Level Architecture's Object Model Template (OMT) [OMT98] as a base for presenting our techniques in a manner useful to designers of multi-models. OMT permits designers to specify object classes and interactions [JPSD97] [JTFP97] [RPR97].

2.4 Chapter Summary

A number of domains employ some form of multi-representation modelling (MRM) with varying degrees of success. We presented some MRM applications and summarised their strengths and deficiencies using the metrics of multi-representation interactions (R1), multi-representation consistency (R2) and cost-effectiveness (R3). Common approaches for MRM involve executing the most detailed model or transitioning from one model to another. These approaches can make the multiple models inconsistent and incur high costs. Maintaining consistent representations of multiple models can be more effective than alternative approaches to MRM. We explore that thought in subsequent chapters.

If you have built castles in the air, your work need not be lost;
that is where they should be. Now put foundations under them.
— Henry David Thoreau

Chapter 3

Foundation

Multi-Representation Modelling (MRM) is a means of capturing the combined semantics of jointly-executing models. The joint execution of multiple models brings up issues of conceptual and representational differences among the models. MRM involves the resolution of such differences. MRM includes but is not restricted to models that are executed as computer programs, called simulations. In this chapter, we lay the foundation for discussing our framework, `UNIFY`, by defining key concepts such as model, representation and interactions. We state and justify assumptions we make in our work and describe our evaluation strategy.

3.1 Model

Modelling is a way to study a phenomenon without undertaking the phenomenon itself. A *model* captures the semantics of selected concepts, objects and processes of a phenomenon in terms of other well-defined concepts, objects and processes. Objects and processes in a phenomenon are called *entities* in a model. The *representation* of an entity

is a means of describing the entity and its properties. The representation of a model is the union of the representations of entities. Anything that is not part of the model is part of the model's *environment*. An *attribute* is an element of the representation of an entity that captures a property of the entity. A *relationship* between two attributes indicates how the value of one attribute changes when the value of the other attribute changes. In a valid or consistent model, the relationships among attributes *hold*, i.e, the values of attributes change in accordance with the relationships among them. Therefore, for each relationship, there must exist functions that translate changes in one attribute to changes in other related attributes. At a given instant of time, the values of the attributes and the relationships among the attributes reflect the phenomenon being modelled.

A model may change over time when the phenomenon it models changes. The state of a model is a set of values such that each member is a well-defined value assigned to an attribute. When the state of a model changes, the values assigned to its attributes may change, although the relationships among attributes continue to hold. Changes in attribute values are caused by interactions. An *interaction* is a communication between entities. An interaction is initiated by an entity called the *sender*, and directed towards an entity called the *receiver*. The sender and receiver may be part of the same model, in which case the interaction is internal. Either the sender or the receiver may be part of the environment or another model, in which case the interaction is external. We do not need to differentiate between internal and external interactions. The *effects* of an interaction are the changes caused by the interaction to the sender and receiver — typically, to their attributes. We define interactions more rigorously in §3.2.

The preceding informal notions are characteristic of what model designers routinely assume. Now, we take a more formal view based on Object Modelling Technique [RUM91], Object Oriented Analysis [SHLAER92], Object Model Template [OMT98] and Unified Modelling Language [ALHIR98] [FOWLER97]. Let *Rep* be the set of all attributes of all entities in a model. Let *Rel* be the set of all relationships that hold in the model. Each relationship $r \in Rel$ is a mapping between sets of attributes belonging to *Rep*, i.e., $r\colon P \rightarrow Q$, where $P, Q \subseteq Rep$. Let *Int* be the set of interactions whose sender, receiver or both are part of the model. We define a model as a tuple of representations, relationships and interactions.

$$Model = \langle Rep, Rel, Int \rangle$$

Our model is similar to an object model in Object Modelling Technique (OMT_R)*. In OMT_R, the object model describes the structure of objects in the system: their identity, attributes, mutual relationships and operations. *Rep* corresponds to the set of identities and attributes of OMT_R objects. *Rel* corresponds to the set of relationships among OMT_R objects. *Int* corresponds to the union of operations, events and actions as defined in OMT_R. An OMT_R object operation refers to an interaction for which the receiver is the same OMT_R object that defines the operation. In OMT_R, a dynamic model is a state diagram describing those aspects of the system concerned with time and the sequencing of operations. External stimuli that may change the model are called events in OMT_R. In other words, OMT_R events are interactions for which either the sender or receiver is

* Rumbaugh *et al* use the acronym OMT for Object Modelling Technique. To resolve a name conflict with the High Level Architecture Object Model Template, we refer to Rumbaugh's Object Modelling Technique as OMT_R and the HLA Object Model Template as OMT.

outside the model. Finally, in OMT_R, a functional model describes changes within each state of the state diagram in the dynamic model. The changes within a state are called actions in OMT_R. OMT_R actions are interactions for which the sender and receiver may not be defined in terms of OMT_R objects; the sender and receiver both are the "system".

Our model is similar to an information model in Object Oriented Analysis (OOA). In OOA, the information model consists of objects, object attributes and relationships among objects. *Rep* corresponds to the set of OOA objects and their attributes. *Rel* corresponds to the set of relationships among OOA objects. An OOA state model is a state diagram in which a transition from one state to another is caused by an OOA event. A process model in OOA describes changes within each state of the state diagram in the state model. These changes are called actions, and are interactions for which the sender and receiver both are the "system". *Int* corresponds to the union of events and actions as defined in OOA.

In the High Level Architecture [AMG95], models are specified using the Object Model Template (OMT). OMT enables a designer to specify class hierarchies for objects, attributes of classes, interactions and parameters of interactions. *Rep* corresponds to the set of OMT object instances along with their attributes. OMT enables specifying interactions for which the sender and receiver are distinct OMT object instances, but not interactions for which either the sender or the receiver is outside the model or interactions for which the sender and receiver are the same OMT object instance. *Int* includes all these interactions, and hence is a superset of the set of OMT interactions. OMT does not include specifications for relationships among objects.

Our model is similar to a model in Unified Modelling Language (UML). In UML, a model consists of entities, relationships among entities and interactions among entities.

Rep corresponds to the set of UML objects and their attributes. *Rel* corresponds to the set of links and associations among UML objects. *Int* corresponds to the union of scenarios, interactions and object operations as defined in UML. A structural model in UML describes the static behaviour of a model, whereas a behavioral model describes the dynamic behaviour of the model.

All of the above models, including ours, assume that $Rep \neq \emptyset$. If $Rep = \emptyset$, then $Rel = \emptyset$ as well. $Rep \neq \emptyset$ indicates that representation exists for a model. If $Rep \neq \emptyset$, $Rel = \emptyset$ describes a model in which attributes are unrelated.

When a model *executes*, it simulates the progress of the phenomenon being modelled, implying the passage of time. Accordingly, when a model executes, time becomes an integral part of the model. We define a model at a particular time t as:

$$Model(t) = \langle Rep(t), Rel(t), Int(t) \rangle$$

As a model executes, its state and the relationships among attributes may change. These changes may happen continuously; however, for most practical executions of models, these changes happen at discrete times. Discretizing time is a common technique in model execution. Accordingly, there exists a sequence of times $T = (t_0, t_1, t_2, \ldots)$, such that at each t_i, the representation and relationships in *Model* are defined. At other times, i.e., $\forall t_j \notin T$, $Model(t_j)$ may be undefined or may be the same as $Model(t_i)$ where $t_i \in T$ and t_i is the largest instant in T such that $t_i < t_j$. The individual times in T may be regarded as observation times at which the model may be verified for consistency. T is monotonically increasing. The interval between two consecutive times is a *time-step*, denoted by $[t_i, t_{i+1}]$, where $t_i, t_{i+1} \in T$. The durations of time-steps in a particular model may vary, i.e., $\forall t_i, t_{i+1}, t_j, t_{j+1} \in T, i \neq j$, it is not guaranteed that $t_{i+1} - t_i = t_{j+1} - t_j$.

The execution of a model on a computer is called a *simulation*. A simulation is a tuple of the representation, relationships, interactions and observation times for that model.

$$Simulation = \langle Rep, Rel, Int, T \rangle$$

We define representation and relationships for model execution. *RepSeq* and *RelSeq* are the sequences of states and relationships that hold during model execution.

$$RepSeq = (Rep(t_0), Rep(t_1), Rep(t_2), \ldots)$$
$$RelSeq = (Rel(t_0), Rel(t_1), Rel(t_2), \ldots)$$

$Rep(t)$ is a set of values assigned to attributes in *Rep* at time t, i.e., $Rep(t)$ is the state of the model at time t. $Rel(t)$ is the set of relationships that hold at time t. A relationship $r \in Rel$, $r: P \rightarrow Q$, $P, Q \subseteq Rep$ holds at all observation times, i.e., $\forall t \in T$, $P(t) \rightarrow Q(t)$, where $P(t), Q(t) \subseteq Rep(t)$. A *dependency* is an indicator of a relationship between two attributes. The *behaviour of a model* is the sequence of states of that model [ABADI95] [LAM94] [HOP79]. Consider two models A and B that have the same representation, relationships and interactions. If attributes in A and B have different sequences of values or the same sequences of values but at different times, then A and B have different behaviours. The sequence of states for an entity is a subset of the sequence of states of a model, i.e., the *behaviour of an entity* is a subset of the behaviour of the model.

3.2 Interactions

In most models, entities and the environment exchange information with one another or influence one another. Models do not execute in isolation; typically, stimuli from the environment may influence behaviour of a model, and conversely, a model may generate stimuli that affect the environment. An *interaction* is a communication that causes a change in the behaviour of its sender or receiver or both.

Entities cause a change in the behaviour of one another by means of interactions. In other words, interactions cause a change in the sequence of states of entities. We regard interactions with one sender and multiple receivers as multiple instances of an interaction from one sender to one receiver. An interaction that causes a change in the state of its receiver changes the receiver's behaviour. Moreover, an interaction that does not change its receiver's state may well cause a change in behaviour. A receiver must evaluate whether the interaction affects it or not and apply the changes caused by the interaction if necessary. The evaluation and consequent action of the receiver take a finite, non-zero amount of time. Thus, the behaviour of the entity given the occurrence of an interaction is different from the behaviour of the entity if that interaction never occurred. An interaction that changes only the relationships in a model will cause the state of the model to change as well because of the changed relationships. We do not differentiate between interactions that change the state and interactions that change the relationships in a model.

Interactions may cause changes to the values of attributes. The semantics of an interaction and the dependencies among attributes determine the effects of an interaction. When the changes caused by an interaction are applied to individual attributes, the interaction takes effect. For an interaction I, $I.affects$ is the set of tuples of attributes and changes to values of attributes caused by the semantics of I. If I causes only a read to an attribute value, the attribute is not in $I.affects$. If I causes a write to an attribute value, the attribute and its changes are in $I.affects$. $I.affects^{+}$ is the set of tuples of attributes and changes to attributes dependent on the attributes in $I.affects$. $I.affects^{*}$ is the set of attributes transitively changed by I, i.e., $I.affects^{*} = I.affects \cup I.affects^{+}$.

Concurrent interactions are those interactions that occur during overlapping simulation time intervals. Interactions that occur one after another, i.e., do not overlap in time, are *sequential* interactions. In logical time systems, two interactions are concurrent if one does not "happen-before" the other [LAM78]. However, by this definition, interactions that occur at different times may be concurrent. In applications involving databases, caches and shared resources, two interactions are concurrent if they occur at overlapping times. We consider interactions as concurrent if they occur during the same time-step. Our definition of concurrence may exclude some concurrent interactions of logical time systems, but includes concurrent interactions in databases and caches. Interactions that occur at the same real time are simultaneous. In practical models, time is discrete, not continuous. Therefore, while real time-steps are of zero duration, time-steps in practical models are of non-zero duration. Thus, many interactions that are not simultaneous but happen to occur during the same time-step will be considered concurrent. The "false simultaneity" introduced by concurrent interactions may be reduced by a finer granularity of time within a model.

Concurrent interactions may be dependent. A *dependent interaction* is one whose effects are predicated on the occurrence of another interaction. An *independent interaction* is one that is not dependent on any other interaction. For example, two interactions may be related by cause and effect, i.e., one interaction causes the other. In such a case, the former interaction is independent of the latter, but the latter is dependent on the former. Concurrent interactions may be dependent solely on account of their concurrence, i.e., if the interactions were not concurrent, they would be independent.

With our definition of interactions, we define *IntSeq* as a sequence of sets of concurrent interactions. Each set contains interactions that occurred during one time-step. $Int(t_i)$ is the set of interactions that occur in time-step t_i. $Int(t_i)_k$ is the kth interaction that occurs in the time-step t_i. No ordering is implied by k; it is used solely to distinguish one interaction from another in that time-step. $\forall t_i \, \forall k, Int(t_i)_k \in Int$. $Int(t_i)$ consists of n_i+1 interactions, i.e., $|Int(t_i)| = n_i+1$. $I \bullet J$ indicates that I and J are concurrent interactions.

$$Int(t_i) = \{Int(t_i)_0 \bullet Int(t_i)_1 \bullet \ldots \bullet Int(t_i)_{n_i}\}$$

$$IntSeq = (Int(t_0), Int(t_1), Int(t_2), \ldots)$$

Concurrent interactions may cause concurrent changes to entities. Let the effect of an interaction $Int(t_i)_k$ on a state of a model be the change $E(Int(t_i)_k)$. $E(Int(t_i)_k)$ is the set of changes in $Int(t_i)_k.affects^*$. Applying the effect of an interaction is equivalent to computing changes to attribute values caused by the interaction, i.e., applying the effect of interaction $Int(t_i)_k$ on the representation $Rep(t_i)$ is equivalent to computing a function $F(Rep(t_i), E(Int(t_i)_k))$. Applying the combined effects of all the interactions in one time-step results in the state of the model at the next time-step.

$$Rep(t_{i+1}) = F(Rep(t_i), E(Int(t_i)))$$

Applying the effects of concurrent, possibly dependent, interactions is called *resolving* the effects of the interactions. Let $E(I \bullet J)$ denote the concurrent effects of interactions I and J, and $E(I) \lozenge E(J)$ denote their sequential effects. Concurrent interactions can be resolved in different ways including, but not limited to, applying the effects of interactions in an arbitrary order. When interactions are independent, their effects when concurrent are indistinguishable from their effects when sequential.

$$E(Int(t_i)_0 \bullet Int(t_i)_1 \bullet \ldots \bullet Int(t_i)_{n_i}) = E(Int(t_i)_0) \lozenge E(Int(t_i)_1) \lozenge \ldots \lozenge E(Int(t_i)_{n_i})$$

The effects of concurrent independent interactions can be resolved by applying the effects of individual interactions one after another. This policy for resolving the effects of concurrent interactions is called *serialization*. If it can be determined that at all time-steps, concurrent interactions are independent, then serialization is a valid policy for resolving the effects of concurrent interactions. When interactions are dependent, their effects when concurrent may not be the same as their effects when sequential. The effects of dependent concurrent interactions may be predicated on the occurrence of one another during the same time-step. In such cases, serialization may resolve the effects of such interactions incorrectly; other policies for resolving the effects are necessary.

3.3 Multi-models

Multiple models of the same phenomenon may execute jointly with one another. Simple, well-designed models executing jointly may capture all the facets required for a particular study of a phenomenon without a designer having to construct one model that captures exactly those facets. Given that the multiple models are simplifications of the same phenomenon, entities common to the models must be correlated or made consistent. However, correlating the entities can become a very significant problem if the models make different assumptions about the processes, objects, environment, the rate of progress of the phenomenon and the accuracy at which the phenomenon is modelled. Inconsistencies among models may undermine the reasons for executing them jointly.

We use the term *representation level* to describe the level of abstraction of a model. If some models are compositions/decompositions or abstractions/refinements of one another, their representation levels are also called *resolution levels* or *resolutions*. An aggregate

model is a relatively low-resolution (high-abstraction, low-decomposition) model, whereas a disaggregate model is a relatively high-resolution (low-abstraction, high-decomposition) model. A *High Resolution Entity* (HRE) is an entity at a low level of abstraction (high decomposition), and a *Low Resolution Entity* (LRE) is an entity at a high level of abstraction (low decomposition). Classification of an entity as an HRE or LRE depends on its resolution level relative to other relevant entities. The resolution levels form a hierarchy, with the highest level being the most abstract or most aggregate one, and the lowest level being the most refined or most disaggregate one. *Aggregation* is the composition of a collection of HREs into a single LRE, and *disaggregation* is the decomposition of an LRE into its constituent HREs.

Multi-representation modelling (MRM) is the joint execution of multiple models of the same phenomenon. We call the union of several models of the same phenomenon a *multi-model*. A multi-model may consist of several models; however, for ease of exposition, we will consider an example multi-model consisting of two models. If $Model^A$ and $Model^B$ are two models of the same phenomenon, then a multi-model $Model^M$ constructed from them is defined as:

$$\begin{aligned} Model^M &= \langle Rep^M, Rel^M, Int^M \rangle \\ Rep^M &= Rep^A \cup Rep^B \\ Rel^M &= Rel^A \cup Rel^B \cup Rel^{cross-model} \\ Int^M &= Int^A \cup Int^B \end{aligned}$$

Rep^A and Rep^B are called *concurrent representations*. We construct Rep^M by including all of the attributes in Rep^A and Rep^B, after disambiguating name conflicts. For an attribute a, $a \in Rep^A \vee a \in Rep^B \equiv a \in Rep^M$.

3.3.1 Cross-model Relationships

$Rel^{cross\text{-}model}$ is the set of relationships required in order to make the multiple models consistent with one another. Since $Model^A$ and $Model^B$ model the same phenomenon, they may represent overlapping sets of objects or processes. In such a case, Rep^A and Rep^B must be correlated. Correlating the representations in a multi-model is called *consistency maintenance*. If $Rel^{cross\text{-}model} = \emptyset$, then $Model^A$ and $Model^B$ are independent of each other because their representations are not related to each other. Then, consistency maintenance reduces to ensuring that the individual models are self-consistent. If $Rel^{cross\text{-}model} \neq \emptyset$, the representations of the models are related. A cross-model relationship $r \in Rel^{cross\text{-}model}$ is a mapping $r\colon P \rightarrow Q$ such that $P \subseteq Rep^A \wedge Q \subseteq Rep^B \vee P \subseteq Rep^B \wedge Q \subseteq Rep^A$. We construct Rel^M by including all of the relationships in Rel^A, Rel^B and $Rel^{cross\text{-}model}$, i.e., for a relationship r, $r \in Rel^A \vee r \in Rel^B \vee r \in Rel^{cross\text{-}model} \equiv r \in Rel^M$.

3.3.2 Mapping Functions

A *mapping function* associated with a relationship among attributes translates the changes in one attribute to changes in related attributes in such a manner that the relationship continues to hold. We assume that designers can construct appropriate mapping functions for each relationship in $Rel^{cross\text{-}model}$. Mapping functions encode application-specific semantics about the relationships among representations. Mapping functions are necessary for any MRM approach, including ours.

A requirement for mapping functions is that at every observation time, they must ensure that a relationship holds by translating value spaces or changes in values of attributes, as necessary. $\forall r \in Rel^{cross\text{-}model}$, $r\colon P \rightarrow Q$, $P, Q \subseteq Rep^M$, a mapping function f

may exist such that, if $P(t_i) \rightarrow Q(t_i)$ holds, then $P(t_{i+1}) \rightarrow Q(t_{i+1})$ holds. For example, f may be of the form $\forall t_i, t_{i+1}, Q(t_{i+1}) = f(Q(t_i), P(t_i), \Delta P(t_i))$, where $\Delta P(t_i)$ is the set of changes to values in $P(t_i)$. Since f ensures that r holds $\forall t_i \in T^M$, f must complete its computation within a time-step. In other words, a lower-bound value for an observation time t_{i+1} is the sum of the value of t_i and the time taken for f to complete.

All mapping functions must be *composable*. If mapping functions f and g translate attribute sets P to Q and Q to R respectively, invoking f and g in succession must translate P to R. Attribute relationships are transitive, i.e., $P \rightarrow Q \wedge Q \rightarrow R \Rightarrow P \rightarrow R$. Composable mapping functions capture transitive dependencies among attributes. If mapping functions are composable, the effects of an interaction propagate to all dependent attributes.

Mapping functions must be *reversible*. Consider mapping functions f and g: $\forall t_i, t_{i+1}, t_{i+2}, Q(t_{i+1}) = f(Q(t_i), P(t_i), \Delta P(t_i))$ and $P(t_{i+2}) = g(P(t_{i+1}), Q(t_{i+1}), \Delta Q(t_{i+1}))$. If no interactions occur during the time-steps $[t_i, t_{i+1}]$ and $[t_{i+1}, t_{i+2}]$, then invoking f and g in succession must result in $P(t_{i+2}) = P(t_i)$ within tolerable approximation. Reversibility is desirable for mapping functions because it ensures that a change does not propagate back to an attribute. Therefore, if Q changes as a result of a change to P, then reversible mapping functions ensure that P does not change again as a result of the change to Q.

3.3.3 Time-Steps

We assume that the time-steps of $Model^A$ and $Model^B$ are compatible. *Compatible time-steps* means that if T^A, T^B and T^M are the sequences of times associated with $Model^A$, $Model^B$ and $Model^M$ respectively, then $Model^A$ and $Model^B$ are defined for all times in T^M. T^M is constructed by interleaving T^A and T^B. Accordingly, times that are common to both

T^A and T^B (albeit labelled differently) are included in T^M only once. If $T^M = T^A \cup T^B$, then $Model^A$ must be defined for all times in T^B and $Model^B$ must be defined for all times in T^A. If $T^M = T^A \cap T^B$, then $Model^A$ and $Model^B$ are defined for all $t \in T^M$. Figure 6 shows two ways to construct T^M.

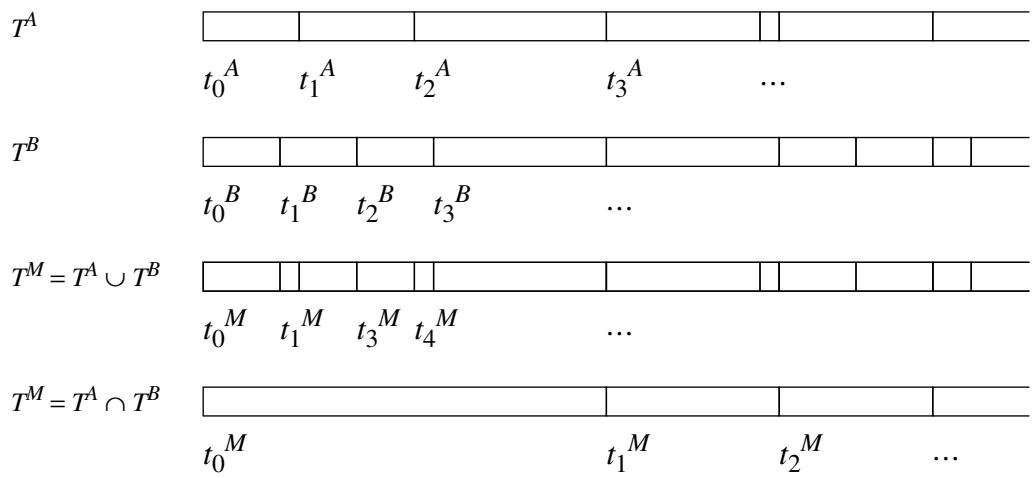


FIGURE 6: Possible compatible time-steps

No matter how T^M is constructed, some interactions in each of Int^A and Int^B must be re-organised as if occurring in time-steps defined by times in T^M. For example, let $t_0{}^A, t_1{}^A, t_2{}^A \in T^A$. If $t_1{}^A \notin T^M$, then interactions occurring in the time-step $[t_1{}^A, t_2{}^A]$ must be re-organised as if occurring in $[t_0{}^A, t_2{}^A]$. This re-organisation increases "false simultaneity". In like fashion, let $t_0{}^A, t_1{}^A \in T^A$, and $t_0{}^B \in T^B$ such that $t_0{}^A < t_0{}^B < t_1{}^A$. If $t_0{}^A, t_1{}^A, t_0{}^B \in T^M$, then interactions occurring in $[t_0{}^A, t_1{}^A]$ must be re-organised into two sets, one occurring in $[t_0{}^A, t_0{}^B]$, and the other occurring in $[t_0{}^B, t_1{}^A]$. This re-organisation decreases "false simultaneity". If $T^M = T^A = T^B$:

$$
\begin{aligned}
RepSeq^M &= (\forall t_i \in T^M \big| (Rep^A \cup Rep^B)(t_i)) \\
RelSeq^M &= (\forall t_i \in T^M \big| (Rel^A(t_i) \cup Rel^B(t_i))) \\
IntSeq^M &= (\forall t_i \in T^M \big| (Int^A(t_i) \bullet Int^B(t_i)))
\end{aligned}
$$

3.4 Evaluation

In this dissertation we will show how UNIFY, our approach for consistency maintenance among concurrent representations satisfies R1, R2 and R3, our requirements for effective MRM.

A model must satisfy its users' requirements. Examples of user requirements are the accuracy of the model, the detail captured by the model and the rate at which the model progresses. The most accurate model of a phenomenon is the phenomenon itself; practical models are simplifications that may fail to imitate the phenomenon in some respects. The Turing test [TURING50] for a model is whether end-users are satisfied that the model captures the facets required for study. Likewise, for a multi-model, end-users must determine whether it meets their requirements. A multi-model can satisfy its users' requirements if its constituent models satisfy the users' requirements and the joint execution of the multiple models is effective.

Satisfactory multi-model $\Rightarrow$ *Satisfactory models* + *Effective joint execution*

Our work concentrates on effective joint execution of multiple models. In contrast, OMT_R, OOA and UML guide a designer in constructing a model to meet users' requirements.

Requirements for models and multi-models must indicate how users can be satisfied. For training models, training experts may indicate satisfaction by assessing how well the model reflects reality. A term used often in the training community is *fair fight*, which signifies an engagement in which no party can deduce and utilize information about the training system (that they could not deduce in a real situation) to gain an unfair advantage. For example, due to an artifact of simulation, an aircraft may be perceived for some time after having been destroyed. This artifact could be employed to draw additional fire and

thus force consumption of ammunition without sustaining losses. Similarly, crews in tank simulators have been reported to identify other tanks as being controlled by computer-generated forces rather than by humans by tracking their movements. The fair-fight concept is relevant to modelling since models approximate reality and there is potential to exploit knowledge of these approximations. In MRM, where a basic theory is still developing, arbitrary design choices may violate the fair-fight concept.

It is important to understand the difference between an unfair fight and what military analysts call the *fog of war*. The fog of war refers to circumstances — typically large numbers of concurrent events — that make it difficult to maintain a coherent picture of the battle, leading to unexpected events. Unfair fights, on the other hand, result from shortcomings in the design of a system and have no counterparts in a real-life phenomenon. Often, inconsistencies in a model are assessed incorrectly as being a part of the fog of war. While creating simulations that pass the Turing test is difficult, an important goal of designers should be to reduce the discrepancies that cause a simulation to fail the test [PETTY94].

Our work concentrates on the effectiveness of joint execution of multiple models. Our approach, called `UNIFY`, is meant to guide designers towards effective MRM. Whether an MRM approach is effective or not can be evaluated on the basis of how well it meets three requirements, listed in §1.3 and below:

- **Multi-representation Interaction (R1)**: The multi-model must permit concurrent interactions at multiple representation levels.

The interactions that occur in $Model^M$ must be the interactions that could occur either in $Model^A$ or in $Model^B$, i.e., Int^M must be $Int^A \cup Int^B$. If Int^M meets this condition, it

means that the joint execution of both models does not restrict the execution of either model. Effective joint execution of multiple models requires that entities at different representation levels initiate and receive interactions that may cause their behaviour to change. Many MRM approaches do not satisfy R1. For example, in selective viewing, if $Model^B$ is the most detailed model, then the only interactions permitted in Int^M are the ones in Int^B. In aggregation-disaggregation, in each time-step of T^M, either interactions in Int^A or interactions in Int^B, but not both are permitted. In Chapter 7, we present a taxonomy for resolving the effects of concurrent interactions in order to accommodate multi-representation interaction.

- **Multi-representation Consistency (R2)**: The multiple representations must be consistent with one another.

Rel^M must hold at all observed times in the multi-model. Moreover, it must be the case that $Rel^{cross\text{-}model} \neq \emptyset$, or else consistency maintenance and joint execution are too trivial to be interesting. In Chapter 6, we present a technique for maintaining consistency among multi-models by showing how to construct $Rel^{cross\text{-}model}$. Application-specific mapping functions associated with each relationship in $Rel^{cross\text{-}model}$ must be supplied by the designer. The mapping functions are required for consistency among multiple representations. Consistent representations are necessary for the consistent behaviour of a multi-model since the state of an entity influences its behaviour [LAM94]. Motivating the choice of finite automata for designing systems, Hopcroft *et al*. say [HOP79]:

> The state of the system summarizes the information concerning past inputs that is needed to determine the behaviour of the system on subsequent inputs.

The inputs and state of a finite automaton are interactions and representation of a model. Since the multiple representations in a multi-model determine the behaviour of the multi-model, maintaining consistency among the representations is required for effective joint execution.

- **Cost-effectiveness (R3)**: The costs of simulation and consistency maintenance must be low.

Simulation costs and consistency costs tend to be trade-offs, as we will see in Chapter 9. Simulation cost is the expenditure of resources in order to simulate entities, possibly at multiple representation levels. Consistency cost is the expenditure of resources in order to ensure that the multiple models meet consistency requirements. The resources expended may be computational, network or memory. In selective viewing, simulation costs are high whereas consistency costs are low since only the most detailed model is executed at all times. In aggregation-disaggregation, simulation costs are relatively low whereas consistency costs are high since the representations must be kept consistent when transitioning among models. We measure simulation cost and consistency cost for `UNIFY`, selective viewing and aggregation-disaggregation, and show how `UNIFY` reduces the total cost of simulation and consistency maintenance.

3.5 Assumptions and Rationale

Our approach for effective MRM, `UNIFY`, makes some assumptions about jointly-executing models. We have presented these assumptions in context earlier in this chapter; we discuss them in detail here.

Existence of representations: A representation exists for an entity and can influence the behaviour of the entity.

Typical models have representations; most designers consider representing entities in a model natural and intuitive. In some contexts, researchers claim that entities must not have a representation at all. For example, Brooks's description of subsumptive behaviour in autonomous agents involves agents maintaining no representation [BROOKS86]. However, a representation is beneficial towards an agent's operation [BRILL96]. Generally, entity state influences entity behaviour [ABADI95] [LAM94] [HOP79]. Therefore, our assumption about the existence and influence of representation is reasonable. We have not investigated in any detail the consequences of eliminating this assumption. Davis's work on variable-resolution process models is closer to a non-representational approach than our work [DAVIS92] [DAVIS98].

Existence of satisfactory models: Individual models meet their users' requirements.

The problem of linking independently-designed components into a composite system is hard enough without the additional complexity of the components falling short of meeting their individual requirements [ALLEN98]. Simply put, a bad model cannot be improved by jointly executing it with other models. Accordingly, we limit the scope of our work to the joint execution of models that meet their users' requirements.

Existence of mapping functions: There exist mapping functions to translate the representation of one model to the representation of other models.

Mapping functions are application-specific methods that capture the semantics of relationships among representations. Since capturing these semantics is essential for consistency of a multi-model, mapping functions are necessary for any approach to MRM.

Since mapping functions are application-specific, instead of specifying their semantics, we derive requirements for their use from example multi-representation models. Mapping functions must translate attribute values and changes to attribute values from one representation to another. Additionally, they must complete their translations in a time-bound manner so that the multiple models appear consistent at all observed times. The specifications of consistency and observed times depend on the application.

Existence of policies for concurrent interactions: There exist policies for resolving the effects of dependent concurrent interactions.

Designers must resolve the intertwined semantics of interactions in order to be able to relate them to one another. Concurrent interactions that are dependent on one another may have effects that cannot be captured by serialization or any other straightforward policy. Designers must decide beforehand how the effects of dependent concurrent interactions must be resolved and subsequently applied. Without a clear understanding of the semantics of interactions, designers cannot expect any MRM approach to be effective. Therefore, similar policies are necessary for any approach to MRM.

Existence of compatible time-steps: The time-steps at which the models execute are compatible.

When multiple models execute jointly, the multiple simulation times must be compatible. Simulation time is a fundamental property of most models. Simulation time is tied to the progress of the phenomenon being modelled. Simulation time may or may not be real, logical, linearly-increasing, monotonic or uni-dimensional. If the multiple models adopt the same sequences of times, they are likely to be compatible and may be expected to execute jointly with few problems. However, the greater the variance between the

sequences of times among the multiple models, the greater the difficulty of ensuring effective joint execution.

Alternative approaches, such as selective viewing and aggregation-disaggregation, cannot guarantee effective MRM even if they make similar assumptions as above because they continue to violate R1, R2 or R3. Therefore, we believe that our assumptions are reasonable for a framework for effective MRM.

3.6 Chapter Summary

MRM, the joint execution of multi-models, presents users with combined semantics that may not be captured by the independent execution of the multiple models. MRM requires designers to invest effort to ensure that the combined semantics meet users' expectations. In particular, ensuring that representations of multi-models are consistent when concurrent interactions may occur is crucial for effective MRM.

`UNIFY` is a framework for designers who require multi-models for their applications. Even if designers are capable of constructing individual models that meet their users' requirements, they can find constructing multi-models difficult. Designers can construct multi-models by ensuring that the joint execution of the multiple models is effective. An approach for effective MRM must satisfy the requirements of multi-representation interaction, consistency and cost-effectiveness.

In the next chapter, we identify problems with aggregation-disaggregation, a popular approach to MRM. After analyzing why these problems occur, we make some general observations about MRM.

If a man will begin with certainties, he shall end in doubts;
but if he will be content to begin with doubts, he shall end in certainties.
— Francis Bacon

Chapter 4

Fundamental Observations

We present four fundamental observations regarding Multi-Representation Modelling (MRM). These fundamental observations are the first of their kind relating to MRM; they support a framework for addressing MRM issues.

Often, characteristics of models make joint execution difficult. One model may be at a lower resolution because its entities are very abstract, whereas another may be at a higher resolution because its entities are very refined. Assumptions about objects, events, interactions and environment may be different. The fundamental processes in the model may have different algorithms because of differences in resolution. The models may progress with different systems of simulation time: discrete-event, time-stepped or continuous. Also, the time-steps at which the models progress may be vastly different.

Often, current approaches to MRM either place too many restrictions on the models or introduce new problems. For example, selective viewing is too restrictive because it requires that all representation, relationships and interactions be expressed at the highest resolution level. Aggregation-disaggregation introduces many problems, as we see in §4.1.

In this chapter, we explore problems in current approaches, and present and substantiate four fundamental observations about MRM. The fundamental observations we present here are exactly that, *observations*. Although they are presented informally, we present strong arguments for their existence. We arrived at these observations after analysing the causes of ineffectiveness in many models. Our observations are fundamental because any general solution to the MRM problem *must* take them into account. They address the general ineffectiveness of joint execution of multiple models, the necessity of maintaining consistency among concurrent representations of the same entity, the dependence among concurrent interactions and temporal consistency. These observations focus the problem of joint execution to the core problem of how to maintain consistency in the multiple representation levels of a single entity. Our framework, `UNIFY`, is based on these fundamental observations.

4.1 Problems with Aggregation-Disaggregation

Aggregation-disaggregation, a common approach to MRM, ensures that entities interact with one another at the same representation level by forcing one entity to be transformed to the level of the other. Typically, if a Low Resolution Entity (LRE) interacts with a High Resolution Entity (HRE), the LRE is disaggregated, i.e., decomposed into its constituents. LRE-LRE interactions would be at the LRE level. A disaggregated LRE may be aggregated so that it can interact subsequently at the LRE level. Aggregation-disaggregation causes simulations to incur considerable resource costs, thus violating R3. Problems such as chain disaggregation, network flooding and transition latency put unacceptable burdens on the resources needed to run a simulation. Moreover, aggregation-

disaggregation can cause mapping inconsistencies between levels, thus violating R2 [NAT95] [NRC97]. Finally, in most variants of aggregation-disaggregation, the multiple models do not execute truly jointly since the system transitions among models as required. In the following sub-sections, we discuss problems with aggregation-disaggregation.

4.1.1 Mapping Inconsistency

Mapping inconsistency occurs when an entity undergoes a sequence of transitions between representation levels resulting in a state it could not have achieved in the simulated time spanned by that sequence. Any scheme in which entities transition between representation levels (e.g., aggregation-disaggregation) must translate attributes between levels consistently. The translation should not lead to incorrect or unintended changes in the attributes. Poor translation strategies cause discontinuities or "jumps" in the state of entities. In Figure 7, when entity L is aggregated to interact with an LRE, the positions of its constituent HREs are lost. Subsequently, when L is disaggregated to interact with an HRE, a standard algorithm or doctrine reconstructs the positions of the HREs [CLARK94] [FRANCE93] [DAVIS93]. However, the reconstructed positions may result in "jumps" in the constituents of L. In general, mapping inconsistencies arise if the translation strategies utilise outdated, inaccurate or insufficient attribute information.

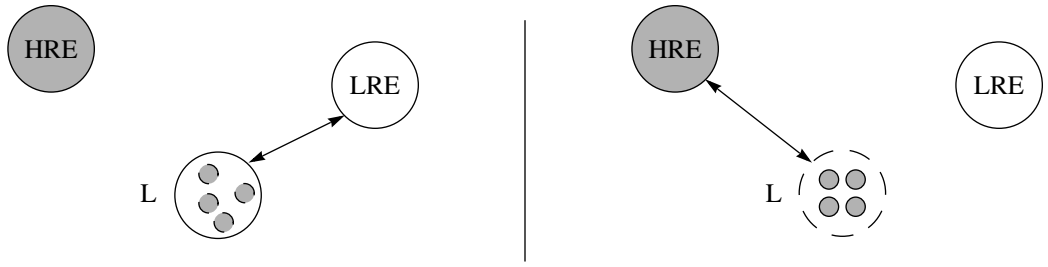


FIGURE 7: Mapping Inconsistency

4.1.2 Chain Disaggregation

Chain disaggregation occurs when a number of entities are forced to disaggregate because a disaggregate-level entity interacts with an aggregate-level entity. Consider an HRE H interacting with an LRE L. Typically, L would be disaggregated to interact with H at the disaggregate level. However, other LREs interacting with L may have to disaggregate, possibly leading to further disaggregations. Figure 8 illustrates the problem. The interaction between can H and L force all LREs to disaggregate in order to be able to interact at the same level. The forced disaggregation caused by the initial contact is called chain disaggregation or spreading disaggregation [ALLEN96] [CALD95B] [PETTY95] [STOBER95]. Chain disaggregation causes the number of simulated entities to increase rapidly. The increased cost of simulating these entities translates to increased load on processors and the network.

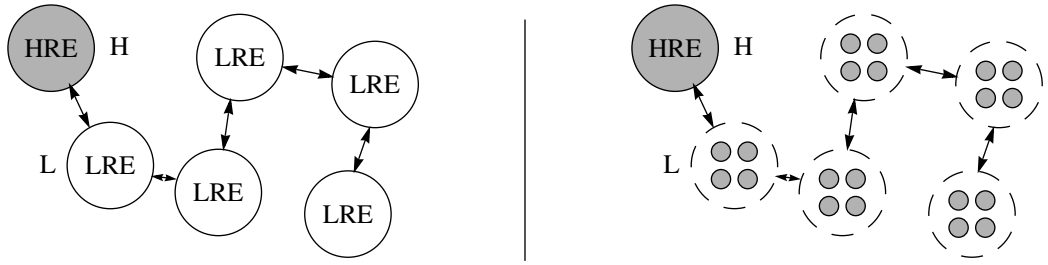


FIGURE 8: Chain Disaggregation

4.1.3 Transition Latency

Aggregation and disaggregation incur time overheads while performing the various steps involved when entities transition between levels. Examples of these steps are set-up, generation of disaggregate values from aggregate values and initiation of protocols to adjust disaggregate values for specific situations. Transition latency, the time taken to effect an aggregation or disaggregation, can be unacceptably high if these steps are

complex [ROBKIN92]. High transition latencies are incompatible with real-time constraints, for example, in human-in-the-loop simulations, because they may cause perceptual or conceptual inconsistencies. An entity that does not change position during a transition period, and then suddenly undergoes a large displacement at the end of the transition period causes a perceptual inconsistency. A conceptual inconsistency may be caused when it takes so long for an entity to disaggregate in order to comply with a request made by another entity that the request becomes obsolete.

4.1.4 Thrashing

When an entity undergoes rapid and repeated transitions from one level to another, it thrashes. For example, an LRE, L, may disaggregate on commencing interactions with an HRE, H. When H moves out of range, L may revert to the aggregate level. However, H's varying proximity to L may cause L to change levels frequently, thus incurring the overheads associated with making a level change and raising the costs of simulation and consistency maintenance. Thrashing depends on the policy that triggers a change of level. Thrashing must be addressed by any MRM approach. High transition latencies compound the problems caused by thrashing because they cause some entities to spend considerable amounts of time just changing levels.

4.1.5 Network Flooding

The network is projected to be a bottleneck in distributed simulations, especially when models consist of large numbers of entities [PULLEN95] [REDDY95] [HOFER95]. Network resources may be strained by aggregation and disaggregation. Each entity created during disaggregation could be a sender/receiver of messages, thus increasing network traffic.

Also, aggregation and disaggregation typically requires the exchange of many control messages — an overhead that must be incurred every time a change of level occurs. These messages can reduce the effective throughput of the network. Frequent changes of level and large numbers of entities may put an unacceptable burden on the network.

4.1.6 Cross-Level Interactions

In many systems, some interactions may span multiple representation levels. For example, two entities at different representation levels could engage in combat indirectly (as in long-range artillery fire). Disaggregation is not triggered because of the indirect nature of the engagement*. Therefore, the sender and receiver of the interaction are at different representation levels. We refer to such interactions as cross-level interactions. Since the participants in cross-level interactions are entities at different representation levels, it is difficult to reconcile the effects of such interactions. Cross-level interactions occur when requirement R1 is not satisfied.

4.1.7 Summary of Problems with Aggregation-Disaggregation

Often, problems with aggregation-disaggregation occur because designers make convenient rather than correct decisions about the joint execution of multiple models. Examples of such decisions are: permitting cross-level interactions, permitting interactions only within a playbox and pseudo-disaggregating. When a multi-model grows in terms of the number of its constituent models, the kinds of interactions that entities may receive, or the different scenarios under which the models execute, such decisions can lead

* Forcing a disaggregation could lead to chain disaggregation, and is therefore undesirable.

to ineffective joint execution. For example, cross-level interactions are difficult to reconcile, playboxes lead to thrashing and pseudo-disaggregation leads to a condition where entities must be able to disaggregate all entities in the model.

An approach for joint execution of multiple models based on correct decisions is necessary. Such an approach will avoid the pitfalls of merely convenient decisions, and satisfy three basic requirements for MRM: multi-representation interaction, multi-representation consistency and cost-effectiveness. This approach must be based on fundamental characteristics of joint execution. In §4.2, we present four fundamental observations about MRM. These observations highlight fundamental characteristics of joint execution. In Chapter 9, we show how our framework for MRM, `UNIFY`, satisfies the three basic requirements for MRM and avoids the pitfalls of other approaches.

4.2 Fundamental Observations

After analysing the causes for ineffectiveness in a number of multi-models, we made four fundamental observations about the joint execution of multiple models. These observations focus on entity interactions, effects of concurrent interactions, dependencies among concurrent interactions and time-step differentials. The fundamental observations influence our choice of the techniques that are part of `UNIFY`: Multiple Representation Entities, Attribute Dependency Graphs and a taxonomy of interactions.

4.2.1 Fundamental Observation 1

Two entities must interact at a representation level common to both so that the semantics of their interactions are meaningful to both. Therefore, the objects and

processes corresponding to each entity must be modelled at all the representation levels at which the entity can interact. When entities interact at common representation levels, they avoid cross-level interactions.

FO-1: For effective joint execution, objects or processes should be modelled at representation levels at which they can interact.

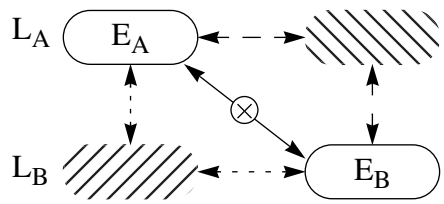


FIGURE 9: Fundamental Observation 1

Consider the joint execution of two models with entities, E_A and E_B, at different representation levels L_A and L_B respectively, as shown in Figure 9. Essentially, FO-1 states that for most applications, in order to interact with each other, either E_A must be represented at L_B or E_B must be represented at L_A. In other words, for effective joint execution, a combination of vertical and horizontal links must be followed.

To see why this observation is true, consider a military training simulation. Here, E_A may be a division of tanks being modelled in a low-resolution simulation while E_B may be a single, self-contained (manned) tank simulator. Typically, division-level engagements are simulated by equations that take the relative strengths of the engaging parties into account; actual firing of weapons and destruction of individual tanks are not simulated. In contrast, individual tank engagements are simulated on the basis of actions taken by the parties involved in the engagement (e.g., the human crew of the tank). These involve simulation of detailed actions such as sighting, target acquisition, firing, detonation and damage assessment.

In general, models at different representation levels are designed for different purposes and consequently, have different foci. What is relevant at one level may not be relevant at another, therefore may not be modelled there. The crew members inside an individual tank simulator expect to see individual targets through their sensors. Presenting them with an aggregated view of a tank division will be ineffective (if visual fidelity of the engagement is an effectiveness criterion).

Similar incompatibilities arise in other dimensions of resolution such as time and space. Time-steps vary from nanoseconds to minutes. When two models with disparate time-steps are executed jointly, the one with the smaller time-step may interpret a lack of response from the other as inaction when in fact, the other will report its action only at the end of its larger time-step. Likewise, terrain representation may vary between models. A simple mathematical mapping function may suffice to translate terrain coordinates between systems. However, sometimes such functions do not exist or are inadequate (e.g., when one model executes in two-dimensional space while the other executes in three-dimensional space). Further, the difference in resolution (e.g., meters versus kilometers) can lead to inconsistencies similar to those observed with time-step differentials.

A technique used to resolve these incompatibilities is to provide *bridges* between representation levels. In the two-level case of Figure 9, a bridge is a diagonal link. Such bridges are useful only in special cases; they are not general techniques for effective joint execution of multiple models. Pseudo-disaggregation can be such a bridge. For example, a perceiver of an aggregate entity could apply a local translation function to obtain a disaggregated view of the aggregate entity. This technique works well as long as perception is the only interaction — it fails if the perceiver also engages the perceived in

combat since the perceived units do not respond to events (e.g., attack, defend, retreat). To achieve a completely realistic engagement, the perceived units must respond as if they were being modelled as individual entities themselves. Thus, while bridges may suffice for joint execution in some cases, in general, entities must be modelled at the appropriate representation levels to achieve the required effectiveness.

Interactions may occur at any level at any time. In order to satisfy FO-1, entities must either (i) maintain representations at all levels at all times, or (ii) dynamically transition to the appropriate level as required. We take the first approach. The second approach, aggregation-disaggregation, has high associated overheads, as noted in §4.1.

4.2.2 Fundamental Observation 2

The high cost of dynamic transitions between representation levels can be reduced by reducing (i) the cost associated with a single transition, and (ii) the number of transitions. The cost associated with a single transition is application-specific. Here, we focus on reducing the number of transitions. Limiting the propagation of transitions, for example, by controlling chain disaggregation, results in significant reductions in overhead. Ideally, a transition should be restricted to a single entity and not propagate at all. Restricting transitions implies that entities must be able to resolve concurrent interactions (i.e., interactions occurring within simulated periods that overlap) at multiple levels. Resolving concurrent interactions means that the effects of these interactions must be combined without compromising effectiveness.

FO-2: The effects of concurrent interactions at multiple representation levels must be combined consistently.

In Figure 10, entity E_C must resolve concurrent interactions with entities E_B and E_D in order to limit the propagation of the transition. Concurrent interactions could be serialized, i.e., processed sequentially and atomically. This approach fails in the context of real-time interactions which *must* appear to take effect concurrently. Serializing the interactions removes the appearance of concurrence.

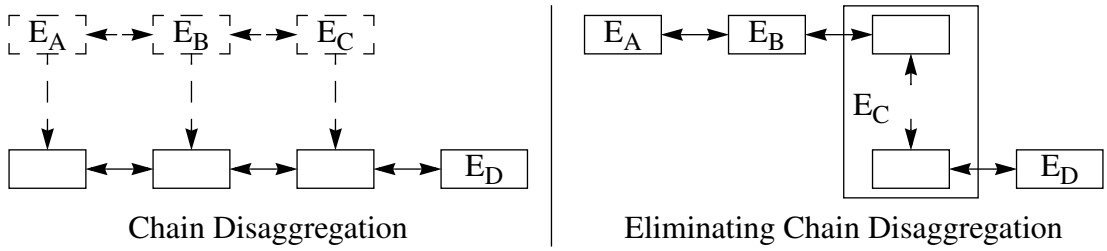


FIGURE 10: Reducing transition overheads by limiting propagation of transitions

Alternatively, interactions could be processed in parallel and their results combined. Although apparently reasonable, this approach has several pitfalls as well. The subtleties of these pitfalls are best explained by an example. Consider the following scenario (Figure 11): LRE_1 and LRE_2 are two platoons of tanks, engaged in battle. At the same time, LRE_2 is engaged by two individual tanks — HRE_1 and HRE_2. The battle between LRE_1 and LRE_2 is simulated at the aggregate level while the battle between LRE_1, HRE_1 and HRE_2 is simulated at the disaggregate level. During a particular time-step, LRE_1 inflicts 50% attrition on LRE_2. The 50% attrition may be interpreted as the destruction of two of the four tanks in LRE_2. During the same time-step, HRE_1 and HRE_2 destroy two tanks in LRE_2[†]. How should these two results be combined? Depending on the amount of overlap in the two interactions, the final result could be a reduction in LRE_2's strength by 50% (complete overlap), 75% (partial overlap) or 100% (no overlap). For the most part,

[†] Typically, platoon-level engagements are specified in terms of percentage attrition, whereas tank-level engagements are specified in number of tanks lost.

this choice must be made arbitrarily and the result assumed to be realistic. Unfortunately, apparently reasonable choices may lead to an unfair fight. The no-overlap choice does not account for the case where LRE_1, HRE_1 and HRE_2 may have fired at the same tanks in LRE_1, whereas the complete overlap choice penalises any co-ordination between LRE_1, HRE_1 and HRE_2 in picking targets from LRE_2. As another example, consider a time-step during which LRE_2 expends 75% of its ammunition fighting LRE_1. HRE_1 and HRE_2 also engage LRE_2 during this time-step, causing LRE_2 to expend 40% of its ammunition. At the end of the time-step, LRE_2 will have expended 115% of its ammunition!

The problems above occur because the effects of an interaction are computed assuming that the interaction is isolated, i.e., it is the only interaction that occurs in a time-step. For some concurrent interactions, assuming they occur in isolation causes their combined effects to be computed incorrectly, leading to ineffective joint execution.

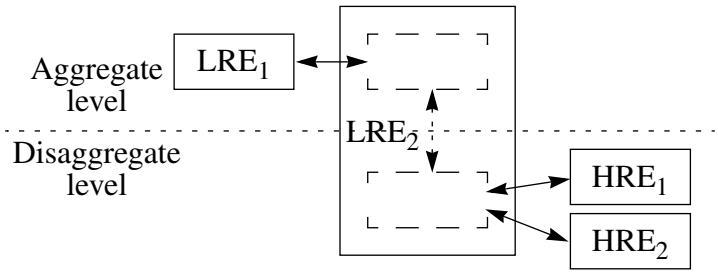


FIGURE 11: Concurrent multi-level interactions

4.2.3 Fundamental Observation 3

Often, consistency problems arise during joint execution because a key property of interactions is ignored when the interactions are isolated. That property is *interaction*

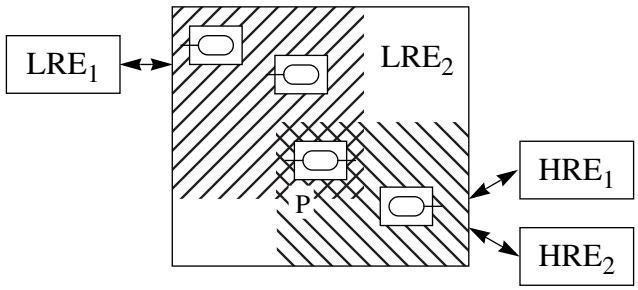


FIGURE 12: Dependency considerations

dependence — an interaction's existence or effects depend on another interaction. Consider the more detailed view of Figure 11 shown in Figure 12. In a time-step duration τ, LRE_2 interacts with LRE_1, reducing the ammunition of a constituent tank (P) by 25%. In effect, P fires at LRE_1 during τ. Also, in τ, LRE_2 interacts with HRE_1 because P fires at HRE_1. Both interactions involve the firing of a weapon by P *in the same time-step*. Clearly, this is physically impossible (indicated in Figure 12 by tank P having two turrets). By permitting such an outcome, the simulation permits an unfair engagement.

The problem arises because two interactions that occur at overlapping simulation times involve a common entity, thus affecting each other's outcome. The two interactions of interest, the aggregate-level interaction between LRE_1 and LRE_2, I_1, and the disaggregate-level interaction between tank P in LRE_2 and HRE_1, I_2, both involve tank P firing. Since P can fire only once, I_1 and I_2 are dependent. Therefore, the results generated by applying their effects independently are incorrect.

FO-3: Concurrent interactions may be dependent.

Interactions that overlap in (i) simulation time, and (ii) the set of interacting entities, may be dependent because they can affect the outcome of one another. For example in Figure 12, one interaction precludes the other. If two interactions that are dependent are executed independently, effectiveness will be compromised when the results of these interactions are combined.

4.2.4 Fundamental Observation 4

In §4.2.3, we have shown that the fundamental issue underlying consistent combination of concurrent interactions is dependence among interactions. Time-step

differentials aggravate the inconsistencies created due to dependency issues. Two interactions can be dependent if they overlap in time. The greater this overlap, the higher the potential for inconsistency.

FO-4: Time differentials may cause inconsistencies.

We elaborate on the problem of time differentials with a simple example. Let E_1 and E_2 be two entities that can change an attribute *v*. For this discussion it does not matter whether or not E_1 and E_2 are entities that describe the same object or process. During their time-steps, E_1 and E_2 send interactions that cause *v* to change; the changes may depend on the previous value of *v*. Thus, during each time-step, each entity reads *v*, performs some computation and writes to *v*.

Let the models for E_1 and E_2 both execute initially with time-steps of equal duration, i.e., $TS(E_1) = TS(E_2) = \tau$. Furthermore, we synchronise the executions of E_1 and E_2 so that all time-step boundaries for these entities occur at the same time. In Figure 13, each bar represents a time-line for one of the entities. Vertical breaks in the bar denote time-step boundaries. It is simple to ensure that E_1 and E_2 are temporally consistent, i.e., they have the same view of *v*. At the end of each time-step, we reconcile the changes to *v* by computing some function of the effects of E_1 and E_2. At the start of the next time-step, both E_1 and E_2 read the *same value* of *v*, no matter how we resolve the concurrent changes of the previous time-step.

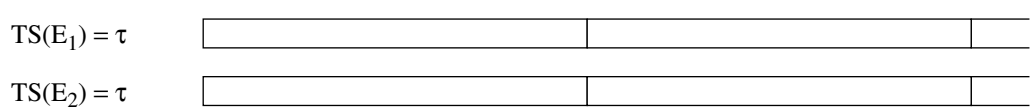


FIGURE 13: Time-steps — Equal and In-phase

Now let us assume that we neglected to synchronise the time-steps of E_1 and E_2. The shaded areas in Figure 14 denote times when E_1 and E_2 are temporally inconsistent. The

inconsistency arises because E_1 (which lags in terms of time-steps) continues to compute a change to v based on the value read at the *start* of E_1's time-step, whereas E_2 may have changed v at the end of E_2's time-step, which occurred before the end of E_1's time-step. The implications of temporal inconsistency can be different for different applications. E_1 may write a new value for v at the end of its time-step, thus causing E_2's computation to become "stale". E_1 may discard its computation and read the new value of v; however, E_1 may be forced to do so at the end of every time-step, thus rendering it redundant.

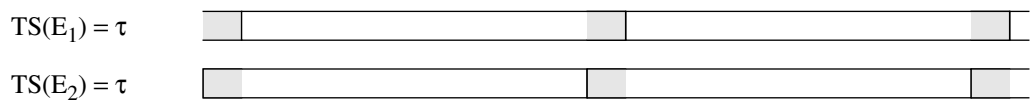


FIGURE 14: Time-steps — Equal but not In-phase

Temporal inconsistency is exacerbated if the durations of E_1 and E_2's time-steps are different. In Figure 15, E_2's time-step duration is $\tau/5$, whereas E_1's time-step duration remains τ. At the end of each of its time-steps, E_2 writes to v, therefore, for most of its time-step, E_1 uses outdated values of v. The increase in temporal inconsistency can be seen by the increase in the length of the shaded regions.

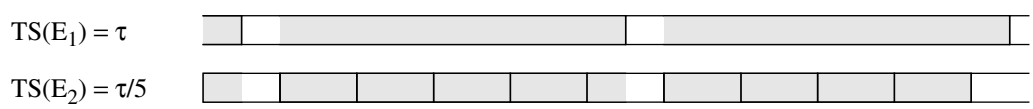


FIGURE 15: Time-steps — Unequal and not In-phase

If E_1 and E_2 have equal time-step durations, they can be temporally consistent. However, this requirement unnecessarily forces the time-step duration of E_2 to be τ, or the time-step duration of E_1 to be $\tau/5$. If a difference in E_1 and E_2's views of v at an observation time changes the behaviour of neither E_1 nor E_2, then the temporal inconsistency is *tolerable*. Let δv be a tolerable variance in the value of v during the time-step $[t_0, t_5]$ for E_1 (Figure 16). At the end of each time-step $[t_0, t_1]$, $[t_1, t_2]$, …, $[t_4, t_5]$ for

E_2, if the value of v changes by less than $\pm\delta v$, then E_1 and E_2 are temporally consistent with respect to v. If during all time-steps E_1 and E_2 are temporally consistent, then E_1 and E_2 execute at *compatible time-steps*.

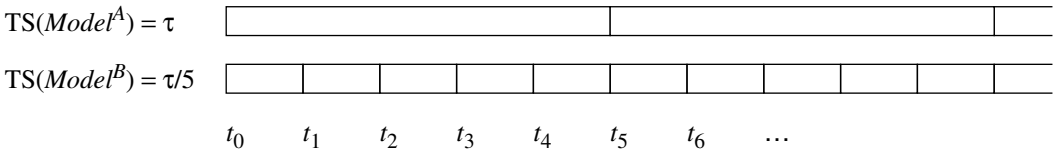


FIGURE 16: Compatible Time-steps

Even if time-steps are made equal, temporal inconsistency may arise if the entities do not read the same value of v at the start of each time-step. Consider Figure 17, in which some time-steps have been labelled. Suppose E_1 modifies v during the time-step between t_1 and t_2 without reading v beforehand. In effect, E_1 executes with the value of v read in the previous time-step. That value may have been changed by E_2 subsequently. Therefore, during the time-step between t_1 and t_2, E_1 and E_2 may be temporally inconsistent.

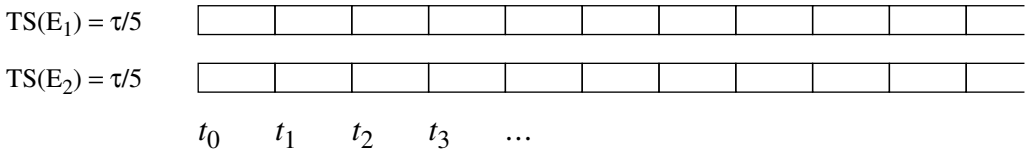


FIGURE 17: Eliminating time-step differentials

While proper design of models can remedy temporal inconsistency caused by cases such as the last one, temporal inconsistency caused by the previous cases may undermine the joint execution of multiple well-designed models. When executing legacy simulations such as `AWSIM/ModSAF`, `Eagle/BDS-D` and `BBS/SIMNET` jointly, time-step differentials are common. Low-resolution simulations typically use equations with coefficients derived from historical data aggregated over periods ranging from several minutes to days [KARR83] [EPST85]. Hence, time-steps of several minutes to a few hours are typical for such simulations. On the other hand, high-resolution simulations such as

CCTT/SIMNET tanks execute at the millisecond time-step level [MILLER95]. Resolving time-step differentials may be a very difficult problem, especially for legacy systems. FO-4 indicates that we must direct future simulation efforts towards solving this problem if we are to achieve effective multi-representation modelling.

4.3 Chapter Summary

The fundamental observations highlight the basic issues that must be addressed by any general, scalable approach to multi-representation modelling (MRM). These observations are a foundation for a successful approach to effective MRM. The fundamental observations address the issue of how models may interact, how dependent concurrent interactions may cause inconsistency and why resolving time differentials is important. These observations arise from the experience of analysing many models and determining why joint execution of these models becomes ineffective.

The key to multi-representation modelling is employing a holistic approach that is designed to solve issues of consistency. In the rest of this dissertation, we present one such approach, UNIFY, based on the fundamental observations.

All for one, one for all!
— *Alexandre Dumas,* The Three Musketeers

Chapter 5

Multiple Representation Entities

A *Multiple Representation Entity* (MRE) incorporates concurrent representations of multiple models. MREs are a part of `UNIFY`, our framework for effective MRM. The viability of an MRE rests on three key assumptions: (i) the presence of mapping functions that translate attributes from one representation to another, (ii) the presence of policies to resolve the effects of dependent concurrent interactions and (iii) compatible time-steps. Similar assumptions are not sufficient to make alternative approaches viable for effective MRM because the approaches continue to violate R1, R2 and R3. We believe that our assumptions are reasonable because without them the semantics of multi-models are not clear, and *no* MRM approach can be effective.

Our thesis is that MRM can be effective. Effective MRM can be achieved by maintaining consistency among concurrent representations. Traditional approaches to MRM, such as aggregation-disaggregation and selective viewing, violate R1 because they simulate only one model at any given time. Typically, attributes in the representation of the simulated model are updated as a result of interactions, but attributes in representations of

other models are *ghosted*, i.e., updated only in response to updates in the simulated model. Ghosting violates R1 because it constrains the kinds of interactions among entities within models by disallowing interactions with non-simulated models. Entities must be capable of interacting at multiple levels (R1), and must be represented at all levels at which they interact (FO-1). Therefore, for effective joint execution of multiple models, the representation of each model must exist at all times. We call representations that exist at all times and permit interactions at all levels concurrent representations. Maintaining concurrent representations means preserving the representations, as opposed to discarding or ghosting them. MREs are our technique for maintaining concurrent representations.

Maintaining *internal consistency* — consistency among concurrent representations — within an MRE when concurrent multi-representation interactions occur is a key challenge in `UNIFY`. For concurrent representations to be consistent with one another, changes to one representation must propagate to the other representations. We assume the presence of appropriate mapping functions to translate changes from one representation to another. The effects of concurrent multi-representation interactions must be resolved and applied to the representations. We assume that a designer can construct policies to resolve the intertwined semantics of such interactions. Lastly, we assume that the time-steps at which multiple models execute are compatible. Provided a designer can satisfy these assumptions, we show how to maintain internal consistency within an MRE.

In §5.1, we describe MREs. In §5.2, we present challenges with MREs. In §5.3, we discuss why our assumptions are necessary and sufficient for `UNIFY`, but insufficient for other MRM approaches. In §5.4, we describe the execution of an MRE broadly, deferring

detailed descriptions to Chapters 6 and 7. In §5.5 and §5.6, we present the benefits and limitations of MREs. We summarise in §5.7 with a table that compares MRM approaches.

5.1 Description of an MRE

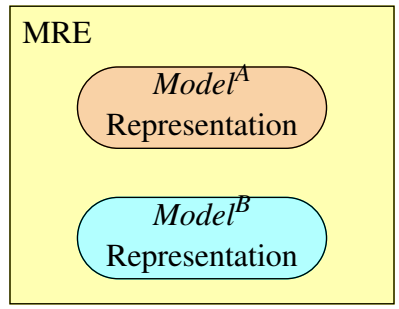


FIGURE 18: An MRE

A *Multiple Representation Entity* maintains concurrent representations. The representation of each model in a multi-model exists within an MRE at all times. Consider a multi-model, $Model^M$, consisting of two models, $Model^A$ and $Model^B$ (Figure 18). Rep^M is an MRE. $Rep^M(t_i)$ is the state of $Model^M$ at time t_i, i.e., it is a meaningful assignment of values to each attribute in Rep^A and Rep^B at time t_i. $RepSeq^M$ is a sequence of states for $Model^M$.

$$
\begin{aligned}
Rep^M &= Rep^A \cup Rep^B \\
\forall t_i \in T^M, Rep^M(t_i) &= (Rep^A \cup Rep^B)(t_i) \\
RepSeq^M &= (Rep^M(t_0), Rep^M(t_1), Rep^M(t_2), \ldots)
\end{aligned}
$$

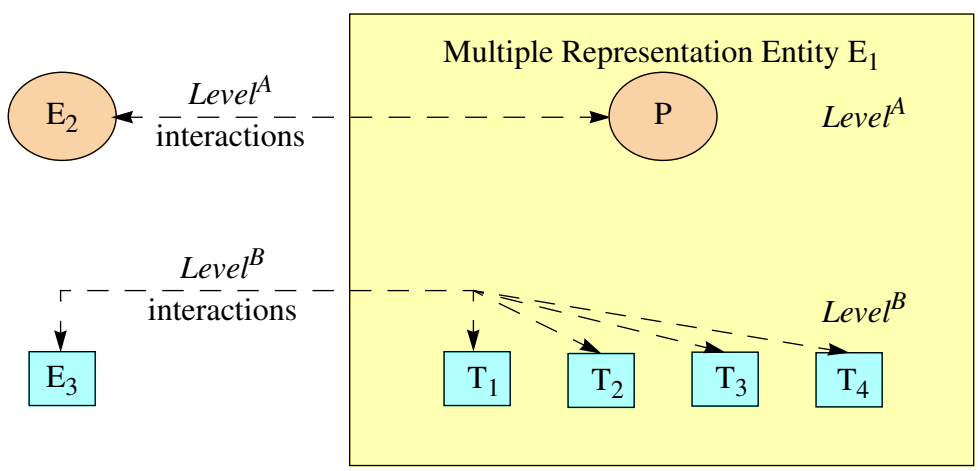


FIGURE 19: Multi-representation Interaction

Figure 18 shows an MRE for the representations of $Model^A$ and $Model^B$. Recall from §3.1 that the representation of an entity is a subset of the complete representation of a

model. An MRE may maintain a subset of Rep^A and Rep^B to describe one object or process present in both models. For example, in Figure 19, P is an entity that describes an object in $Model^A$ and $T_{1\text{-}4}$ are entities that describe the same object in $Model^B$. E_1 is an MRE consisting of the representations of P and $T_{1\text{-}4}$, thus describing the same object at multiple representation levels.

Each MRE either maintains or efficiently furnishes the state at all desired representation levels. Moreover, an MRE permits interactions at all representation levels at all times. By definition, an MRE satisfies R1. An entity in either model interacts with another entity at a representation level common to both. Let the representation levels for $Model^A$ and $Model^B$ be $Level^A$ and $Level^B$ respectively. Let E_2 be a $Level^A$ entity and E_3 be a $Level^B$ entity (see Figure 19). E_2 and E_1 interact at $Level^A$, which means that E_2 and P interact. Likewise, E_3 and E_1 interact at $Level^B$, which means that E_3 and $T_{1\text{-}4}$ interact. MREs disallow cross-level interactions (see §4.1.6). For example, E_2 cannot interact directly with $T_{1\text{-}4}$. Likewise, E_3 cannot interact directly with P.

5.2 Challenges

The challenge with MREs is maintaining consistency among representations when concurrent interactions occur (R2). This challenge can be divided into three issues:

1. How must internal consistency be maintained when a representation changes?
2. How must the changes caused by concurrent interactions be resolved?
3. How must time-step differentials be addressed?

The representations of jointly-executing models must be consistent at all observation times. In Figure 19, for E_1 to be internally consistent, any change to the representation of

P must affect the representations of $T_{1\text{-}4}$ as well and *vice versa*. An interaction between E_2 and E_1 may result in a change to the representation of P. This change must propagate to $T_{1\text{-}4}$, i.e., the interaction must affect the representations of $T_{1\text{-}4}$ as well. Likewise, an interaction between E_3 and E_1 may result in changes to the representations of $T_{1\text{-}4}$. These changes must propagate to P. Propagating changes requires a technique for capturing relations among attributes, and functions that translate changes to attributes.

An MRE must remain consistent at all observed times even when concurrent interactions occur. In Figure 19, if E_2 and E_3 interact with E_1 concurrently, the representations of P and $T_{1\text{-}4}$ may change concurrently. It may be extremely difficult to reconcile these concurrent changes when they propagate to the other representation level. For example, an interaction between E_2 and E_1 (or P) may preclude an interaction between E_3 and E_1 (or $T_{1\text{-}4}$). As another example, the effects of interactions between E_2 and E_1 (or P) and between E_3 and E_1 (or $T_{1\text{-}4}$) may be enhanced when the interactions occur concurrently. In both these cases, the naïve solution of "adding up" the effects of these interactions is incorrect because the interactions are dependent on one another.

Temporal inconsistency caused by time-step differentials must be eliminated. If the time-steps for $Model^A$ and $Model^B$ in Figure 19 are different, it becomes difficult to determine whether two interactions at different representation levels are concurrent or not. Consequently, the effects of these interactions are hard to resolve. While equal and in-phase time-steps may eliminate temporal inconsistency, requiring that all jointly-executing models progress at equal time-step durations is overly restrictive. Accordingly, we assume that the time-steps of multiple models are *compatible*, not necessarily *equal*.

5.3 Rationale

We made three assumptions in order to overcome the challenge of consistency maintenance among concurrent representations: (i) the presence of mapping functions, (ii) the presence of policies for concurrent interactions and (iii) the presence of compatible time-steps. As we show in §5.4, these assumptions are necessary and sufficient to maintain consistency within an MRE when concurrent interactions occur. We make two arguments for the reasonableness of these assumptions.

First, without any of these assumptions, the semantics of multi-models are not evident. These assumptions require designers to incorporate application-specific knowledge into the joint execution of multiple models. Alternative approaches to MRM make similar assumptions. For instance, selective viewing requires mapping functions to translate attributes from one representation to another. These mapping functions are invoked only once — when constructing the representation for the most detailed level. Likewise, aggregation-disaggregation requires mapping functions to translate attributes from one representation to another during aggregation and disaggregation. Concurrent interactions may be dependent whether they are at the same or different representation levels. Therefore, selective viewing and aggregation-disaggregation require policies for resolving the effects of dependent concurrent interactions. In selective viewing, since only the most detailed model is executed at all times, time-steps are trivially compatible. Similarly, in aggregation-disaggregation, only one model is executed at all times. Therefore, at any instant, time-steps are trivially compatible.

Second, alternative approaches cannot guarantee effective MRM despite making similar assumptions. For effective MRM, an approach must satisfy the requirements of

multi-representation interaction (R1), multi-representation consistency (R2) and cost-effectiveness (R3). Despite making assumptions about the presence of mapping functions, policies for resolving effects of interactions and compatible time-steps, selective viewing and aggregation-disaggregation cannot guarantee effective MRM. Since selective viewing and aggregation-disaggregation execute only one model at a time, they disallow multi-representation interactions, thus violating R1. Selective viewing satisfies R2 trivially because consistency must be maintained within the representation of only one model. Aggregation-disaggregation can violate R2 because of mapping inconsistencies among the representations of multiple models. In aggregation-disaggregation, when one model is executed, attributes in the representations of other models are lost or ghosted. If the attributes are lost, then transitioning representation levels may cause discontinuities in the values of attributes even if mapping functions exist. Finally, selective viewing and aggregation-disaggregation result in high costs. Since selective viewing involves simulating the most detailed model at all times, simulation cost is expectedly high. Aggregation-disaggregation reduces simulation costs by transitioning to a low-detail model whenever possible. However, aggregation-disaggregation incurs high consistency cost. The high costs in either case violate R3.

Table 2 summarises the assumptions made by various MRM approaches.

TABLE 2: Summary of Assumptions made by MRM approaches

Assumptions	Selective Viewing	Aggregation-Disaggregation	UNIFY
Mapping functions	Required initially	Required	Required
Policies for resolving concurrent interactions	Required	Required	Required

TABLE 2: Summary of Assumptions made by MRM approaches

Assumptions	Selective Viewing	Aggregation-Disaggregation	UNIFY
Compatible time-steps	Trivial	Trivial	Required

5.4 Execution of an MRE

An MRE permits concurrent interactions at multiple representation levels and maintains consistency among the multiple representations. Execution of the MRE entails applying the effects of any interaction consistently to attributes at all levels of the MRE. Therefore, during each time-step, the effects of interactions at multiple representation levels must be resolved and applied to the concurrent representations in an MRE. Recalling our definitions from Chapter 3 and §5.1:

$$Rep^M(t_{i+1}) = F(Rep^M(t_i), E(Int^M(t_i)))$$
$$\therefore Rep^M(t_{i+1}) = F((Rep^A \cup Rep^B)(t_i), E(Int^A(t_i) \bullet Int^B(t_i)))$$

A Consistency Enforcer and an Interaction Resolver are responsible for maintaining consistency among concurrent representations (Figure 20). An *Interaction Resolver* (IR) for an MRE is a module that determines $E(Int^A(t_i) \bullet Int^B(t_i))$, $\forall t_i \in T^M$, i.e., it resolves the effects of concurrent interactions. A *Consistency Enforcer* (CE) for an MRE is a module that generates $Rep^M(t_{i+1})$, $\forall t_i \in T^M$, i.e., it maps the effects of interactions from one level to another. For example, if E_1 receives concurrent interactions from E_2 and E_3, the IR resolves their effects. The resolved interactions may change the representation of P or $T_{1\text{-}4}$ or both subsequently. When an interaction changes attributes in one representation, the CE changes related attributes in the other representation appropriately. Subsequently, if E_2 and E_3 view E_1 concurrently, they receive consistent views of E_1 from the representations of P

and $T_{1\text{-}4}$. A CE and an IR have application-specific and application-independent components; in our work, we present the latter.

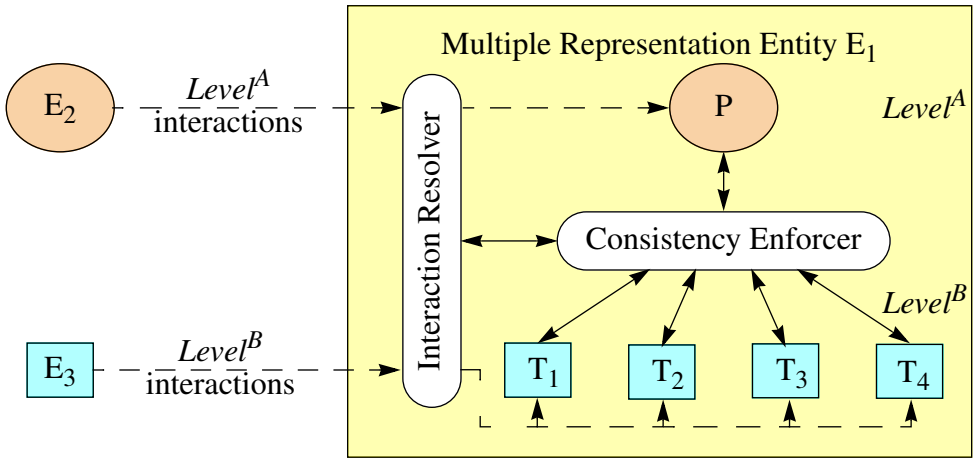


FIGURE 20: Execution of an MRE

5.4.1 Maintaining Consistency

A CE maintains internal consistency in an MRE. In effect, a CE ensures that an MRE exhibits temporal consistency and mapping consistency. In the following sub-sections, we show how an MRE exhibits consistency.

5.4.1.1 Temporal Consistency

An MRE exhibits temporal consistency if the changes caused by interactions are applied consistently to all representation levels. If the multiple representations within an MRE are mutually consistent, the MRE is temporally consistent. Entities viewing a temporally consistent MRE at overlapping times receive consistent views of the MRE.

For a valid and consistent model, $\forall t_i \in T^M$, $Rel^M(t_i)$ must hold. Let there be a relationship $r \in Rel^M(t_i)$ such that $r\colon P(t_i) \rightarrow Q(t_i)$, where $P(t_i), Q(t_i) \subseteq Rep(t_i)$. Let an interaction $Int^M(t_i)_k$ occur. Suppose $P(t_i) \subseteq$ attributes in $Int^M(t_i)_k.affects^*$ and $\Delta P(t_i) \subseteq$ changes in $Int^M(t_i)_k.affects^*$ such that applying $\Delta P(t_i)$ to $P(t_i)$ results in $P(t_{i+1})$.

For r to hold, mapping functions must generate $\Delta Q(t_i)$ such that applying $\Delta Q(t_i)$ results in $Q(t_{i+1})$ eventually. Consequently, r holds at observation time t_{i+1}, i.e., $r \in Rel^M(t_{i+1})$.

Mapping functions are necessary for translating the attributes in one representation to the attributes in another. Translating attributes means translating *value spaces*, *changes in values* or *types* of attributes from one representation to another. For example, consider the T-joint in Figure 21. One model may represent the T-joint with attributes such as connectedness, position and orientation. Another model may represent it as a pair of boards and a nail, each with attributes such as position and orientation. A mapping function must translate the positions of the boards to the position of the T-joint. Likewise, another mapping function must perform the reverse translation — from the position of the T-joint to the positions of the boards. Such mapping functions must take the values of some attributes and change them to the values of other attributes. Another pair of mapping functions must translate the orientation of the T-joint to the orientations of the boards and *vice versa*. These translations may be computationally less complex if the changes in orientations rather than the values of orientations are translated. Finally, consider the attribute of connectedness for a T-joint. Assume the system can infer that a T-joint is connected if the positions of two boards and a nail overlap*. A mapping function that translates the positions of the boards and nail to the connectedness of the T-joint must

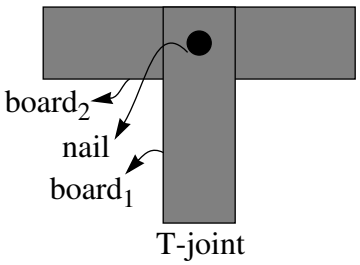


FIGURE 21: T-joint entity

* Naturally, if the boards and nail happen to lie in those positions without the boards having been nailed, the system may infer incorrectly that the T-joint is connected. Resolving this issue is out of the scope of our work, and for the purposes of this discussion, irrelevant.

translate the types of the attributes as well as the values. Finally, translations by mapping functions must complete before the time-step ends.

5.4.1.2 Mapping Consistency

An MRE exhibits mapping consistency if mapping functions are reversible (see §3.3.2). An interaction initiates translations caused by mapping functions. Sequences of interactions initiate repeated translations. Repeated translations must not cause discontinuities or "jumps" in concurrent representations (see §4.1.1). Reversible mapping functions ensure that repeated translations do not cause such discontinuities.

An MRE supports the design of reversible mapping functions. For the T-joint of Figure 21, let *f* translate the board positions to the T-joint position, and *g* translate the T-joint position to the board positions. Provided no interactions occur, if *f* translates the current values of the board positions to a value for the T-joint position, then *g* translates the value of the T-joint position to new values for the board positions, the new and previous values for the board positions must be within tolerable error. If either function could have generated a number of possible values for the resultant attributes, the previous values of the resultant attributes may be taken into account in order to generate the new values. For example, if the T-joint is rotated by 180°, invoking *f* on the values of the board positions may result in the original T-joint position. Subsequently, invoking *g* may result in board positions corresponding to no rotation, thus resulting in an intolerable change to the board positions. In contrast, if *g* took the orientation attribute or the previous values for the board positions into account, then the new positions would correspond correctly to the rotated T-joint position. Irrespective of the details, *f* and *g* must be reversible for the MRE to exhibit mapping consistency.

5.4.2 Resolving Concurrent Interactions

Dependent concurrent interactions may occur because an approach satisfies R1. An MRE permits concurrent interactions at multiple representation levels. Entities at any representation level may initiate and receive interactions that change the appropriate representation. If entities at different representation levels interact, the effects of concurrent interactions at multiple levels must be resolved, i.e., the effects of these interactions must be applied to all levels consistently (FO-2). However, concurrent interactions may be dependent (FO-3). The effects of dependent concurrent interactions must be resolved in a meaningful manner, i.e., in a manner consistent with requirements.

An IR is responsible for resolving the effects of concurrent interactions in an MRE. We assume that designers understand the semantics of interactions in their applications well enough to classify them and specify policies for resolving their dependent effects. Without such an understanding, arbitrary policies such as serialization must be chosen to resolve the effects of interactions. Arbitrary policies often fail to resolve the effects of dependent concurrent interactions meaningfully. In Chapter 7, we show how designers can construct an IR for an MRE.

5.4.3 Storing Attributes in a Core

Since an MRE incorporates concurrent representations, it makes a high demand on resources such as memory to store representations. Although conceptually it is straightforward to think of MREs as using memory for each representation, memory may be conserved by storing a small set of attributes at all times and generating other attributes on demand. In Figure 22, the MRE stores a set of attributes at all times from which it can

generate all attributes at all desired levels in a timely manner on demand. This set of attributes, the *core set* or *core*, may be updated on every interaction to keep the MRE internally consistent. Attributes in the core must be chosen such that they are sufficient for generating all the attributes in the MRE. The core set must be stored at all times in the simulation, but the other attributes may be discarded when they are no longer necessary.

For some applications, a core set of attributes that is smaller than the set of all attributes at all representations can exist. For example, if a molecular and atomic model of a compound execute jointly, the position and orientation attributes in the molecular model may determine the position attributes in the atomic model uniquely and *vice versa*. Therefore, storing either the molecular position and orientation or the atomic positions in a core may be sufficient to maintain internal consistency in an MRE for that compound. Since the core is a subset of all the attributes at all levels, we develop criteria that identify attributes that should be in the core. We have identified four such criteria: reversibility, decreasing validity with time, cost ratio and frequency of access. These criteria are independent but may conflict with one another. In such a case, appropriate weights must be assigned to the criteria to aid selection of the core attributes.

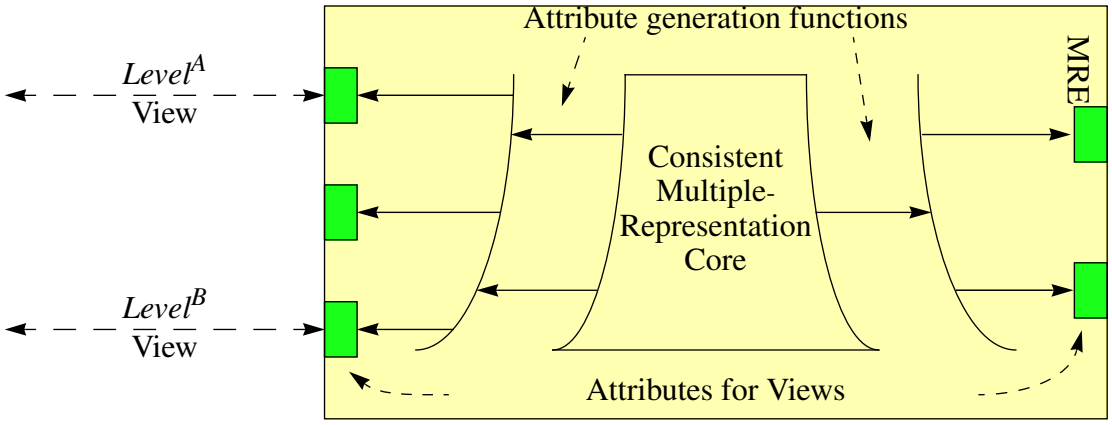


FIGURE 22: Core attributes

Reversibility: For many attributes, it is important that reversible mapping functions translate the values at one level to the values at another level. However, in many cases reversible mapping functions may be hard to find or encode. In such cases, when the attributes require reversibility but reversible mapping functions cannot be found, the attributes must be included at all representation levels in the core. Consider an application for which the position attribute requires reversibility but reversible mapping functions cannot be found. The position of the aggregate may be computed by averaging the position of the disaggregate entities. Likewise, a doctrine or template may be applied to the aggregate position to determine disaggregate positions. However, these translations are relevant only when the entities are not perturbed by other interactions. If the positions of the disaggregate entities change by small amounts because of disaggregate-level interactions, then it is not possible to generate those new positions from the aggregate position. Since perfectly reversible mapping functions cannot be found, the position attributes at both levels must be stored in the core.

Decreasing validity with time: Another criterion is whether the attribute's validity decreases or not with time. The attribute could be stored in the core when it is useful and when its validity goes below a threshold it could be removed from the core.

Cost ratio: Cost ratio is the ratio of the cost of maintaining the attribute to the cost of generating it. If the cost of maintaining the attribute is measured by the amount of memory it consumes and the cost of generating it is measured by the time it takes to generate it, then this criterion reduces to a space-time trade-off. If the cost of maintaining the attribute is measured by the amount of time required to change its value, the comparison lies between the time to effect a change and the time to generate the attribute. Whether the

attribute should be stored in the core or not depends on the cost ratio being larger than, smaller than or equal to one.

Frequency of access: Our fourth criterion is the frequency with which the attribute is accessed. If the frequency is high, then it may be judicious to store the attribute in the core.

5.4.4 Comparing against Alternative Approaches

We compare the execution of an MRE against alternative MRM approaches.

5.4.4.1 Comparing against aggregation-disaggregation

Are MREs a variant of aggregation-disaggregation? During aggregation, mapping functions translate disaggregate attributes to aggregate attributes. During disaggregation, the translation occurs in reverse, i.e., mapping functions translate aggregate attributes to disaggregate attributes. Similar translations occur in an MRE. The translation during aggregation loses information that must be re-generated during disaggregation. This re-generation is a common source of mapping inconsistency. The question is whether the translations in an MRE can cause mapping inconsistency similarly.

MREs maintain attributes at all representation levels at all times. In aggregation-disaggregation, attributes are either discarded or ghosted after a translation. In an MRE, attributes at *all* levels are retained after a translation. Consequently, mapping functions can utilise previous values of attributes in order to generate new values, thus avoiding mapping inconsistency. Moreover, an MRE permits interactions at all representation levels at all times, and incurs lower consistency costs than aggregation-disaggregation.

5.4.4.2 Comparing against selective viewing

Are MREs a variant of selective viewing? In selective viewing, only the most detailed model is executed at all times. Attributes in the multiple representations within an MRE may be construed as the attributes in the representation of the detailed model in selective viewing. The question is whether an MRE is a modular variant of selective viewing.

An MRE does not incur unnecessary simulation costs. For example, suppose a platoon model executes jointly with a model of its constituent tanks. In selective viewing, only the tank model executes. Platoon-level interactions must be translated to possibly many tank-level interactions, each possibly changing the representations of the corresponding tanks. In an MRE, platoon-level interactions change the representation of the platoon. Changes to the platoon representation propagate to the tank representations. Therefore, as compared to selective viewing, an MRE incurs a lower simulation cost at the expense of a higher consistency cost. In addition, an MRE permits interactions at all representation levels at all times.

TABLE 3: Comparison among MRM approaches

Requirements	Selective Viewing	Aggregation-Disaggregation	UNIFY
R1: Multi-representation Interaction	No	No	Yes
R2: Multi-representation Consistency	Trivially	Possible	Possible
R3: Cost-Effectiveness (see Chapter 9)	High Cost of Simulation	High Cost of Consistency	Low Costs

5.5 Benefits of MREs

Consistent concurrent representations can eliminate or reduce many of the problems with other MRM approaches. In §5.4.1, we showed how MREs eliminate temporal and mapping inconsistencies. Now, we show how MREs eliminate or reduce the remaining MRM problems discussed in §4.1. Recall that LRE stands for a Low Resolution Entity and HRE stands for High Resolution Entity.

Eliminating Chain Disaggregation: MREs eliminate chain disaggregation. An MRE does not disaggregate, and does not force other entities to disaggregate. Therefore, as Figure 23 shows, MREs do not cause chain disaggregation.

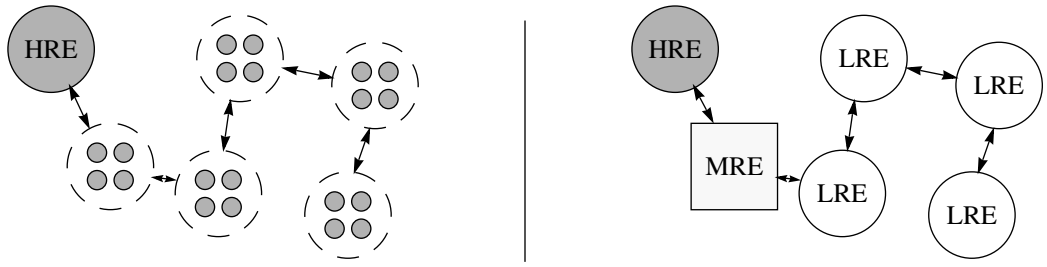


FIGURE 23: Eliminating Chain Disaggregation

Eliminating Transition Latency: MREs eliminate transition latencies encountered in aggregation-disaggregation. MREs do not transition among representation levels, i.e., they do not aggregate or disaggregate. Therefore, they do not require protocols for initiating aggregation or disaggregation. Consequently, transition latency is not an issue with MREs.

Eliminating Thrashing: MREs eliminate thrashing because they do not transition representation levels. Thrashing occurs when an entity aggregates and disaggregates repeatedly in a short period of time because it moves in and out of a playbox or interacts with entities at different representation levels. Thrashing causes the entity to consume

significant processing resources just transitioning levels. Since MREs interact at different representation levels without effecting a transition, MREs do not thrash.

Reducing Network Flooding: MREs reduce network flooding. Selective viewing introduces a large number of entities in the simulation. Likewise, a disaggregated LRE in aggregation-disaggregation introduces a large number of entities in the simulation. As Figure 24 shows, increasing the number of entities in the simulation increases the number of interactions among entities. Since interactions are implemented often as messages on a network, aggregation-disaggregation causes network flooding. MREs capture a benefit of aggregation — introducing fewer entities — thus reducing network flooding.

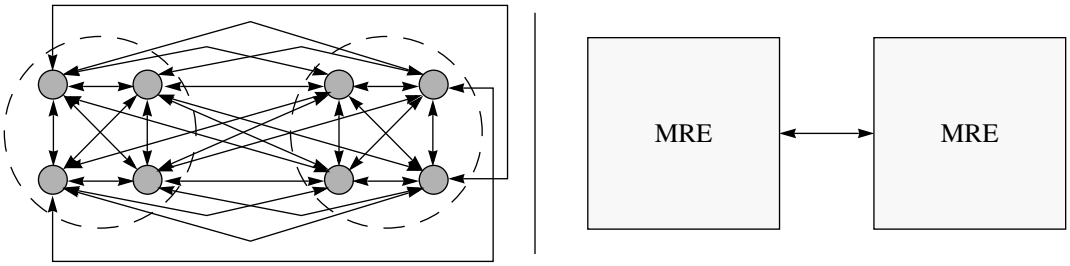


FIGURE 24: Reducing Network Flooding

Eliminating Cross-level Interactions: MREs eliminate cross-level interactions by permitting interactions among entities at all representation levels. With MREs, a $Level^A$ entity never interacts with a $Level^B$ entity; $Level^A$ entities interact with one another, and $Level^B$ entities interact with one another. Since entities interact at representation levels common to them, MREs eliminate cross-level interactions. Entities must negotiate the representation level at which they will interact beforehand. If entities interact at more than one level at a time, "double-interactions" can occur. For example, if an MRE A interacts with an MRE B at $Level^A$ as well as $Level^B$, then a double-interaction occurs when A sends two distinct sets of interactions, one at each level, for the same event. If A and B interact at one level but not both, double-interactions are prevented.

Summary of Benefits: Table 4 summarises the benefits of MREs by comparing how various MRM approaches address the above issues.

TABLE 4: Summary of Benefits of MREs

Benefits	Selective Viewing	Aggregation-Disaggregation	UNIFY
Temporal Inconsistency	Absent	Present	Eliminated
Mapping Inconsistency	Absent	Present	Eliminated
Chain Disaggregation	Inherent	Possible	Eliminated
Transition Latency	Non-existent	Possible	Eliminated
Thrashing	Non-existent	Possible	Eliminated
Network Flooding	High	Possibly high	Reduced
Cross-level Interactions	Non-existent	Possible	Eliminated

5.6 Limitations of MREs

MREs are a technique for capturing the combined semantics of jointly-executing models. An MRE does not show how to design a better model. In the context of an MRM approach, this limitation is not serious; we show that MREs are no worse than alternative approaches. However, without addressing this limitation, a multi-model cannot satisfy its users' requirements even if the MRM approach is effective. MREs can support solutions for many of the following issues; however, MREs do not inherently resolve these issues.

Identifying Representations and Relationships: An MRE does not identify the representation at any level nor relationships between representations. Identifying representations and relationships are the responsibility of a designer. No approach to MRM frees a designer of this responsibility.

Capturing Whole-Greater-than-the-Sum-of-Parts Relationships: Aggregate and disaggregate entities bear the relationship of being whole and parts of one another. The whole-and-parts relationship occurs frequently in battlefield simulations where a number of tanks may be considered as parts of a platoon, or a number of regiments may be considered as parts of a division. Likewise, in multi-resolution graphics, a number of triangles may be considered as parts of an entire surface, or in molecular models, a number of atoms may be considered as parts of a molecule.

A valid concern when aggregate and disaggregate models execute jointly is that the values of some aggregate attributes may be greater than the sum of the values of corresponding disaggregate attributes, i.e., the whole is greater than the sum of its parts. This concern has been called emergent behaviour problem [WIM86] or the configuration problem [HORR92]. For example, tanks may fight with greater strength when configured as a platoon. This increase in strength may be attributable to the presence of a commander who coordinates and guides activities (as is common in the case of military units) or any one of many other similar reasons. As another example, weak forces in atomic models may be ignored since their effect on the position of atoms may be negligible. However, in molecular models, these forces may add up to influence the positions of atoms significantly. The precise relationships between the platoon's strength and the tanks' strength and the atomic forces and the molecular forces must be captured by mapping functions that translate attributes among representations.

Selective viewing does not capture whole-greater-than-the-sum-of-parts relationships. In selective viewing, only the model for the parts is executed. Therefore, whole-greater-than-the-sum-of-parts relationships may not be captured unless information outside the

attributes of each part is present. Typically, an entity maintains attributes relevant only to its own execution. Therefore, the behaviour of an entity when it executes as part of a whole is not distinct from its behaviour when it executes individually. Consequently, information not present in the entity must be used to distinguish these behaviours. Maintaining such information is tantamount to executing multiple models.

Aggregation-disaggregation captures whole-greater-than-the-sum-of-parts relationships, but introduces mapping inconsistency because information is lost during transitions. For example, tanks in a platoon may have manœuvred into a favorable position, thus causing the strength of the platoon to be greater than the sum of the strengths of the tanks. At this point, transitioning to the platoon model and back to the tank model may cause the tanks to be placed in doctrinal formation (since the tanks' previous positions are lost). This placement may result in a platoon strength that is the sum of the strengths of the tanks. Thus, the transitions reduced the strength of the platoon.

MREs aid in the construction of mapping functions that capture whole-greater-than-the-sum-of-parts. Since an MRE incorporates concurrent representations, attributes at all levels are present for the design of mapping functions that avoid inconsistency. Although MREs can capture whole-greater-than-the-sum-of-parts relationships better than alternative approaches, MREs do not aid identification of attributes that bear such relationships. It is the responsibility of the designer to identify and encode such relationships within mapping functions.

Resolving Conflicting Results: The multiple models in a multi-model may employ different algorithms to compute similar effects at different representation levels. For example, in battlefield simulations, Lanchester equations are used to compute attrition or

loss of strength for aggregate-level forces. These equations are differential equations parameterised by coefficients based on historical data [KARR83]. Typically, Lanchester equations compute the results of battles involving large forces, such as divisions, brigades and corps. Also, the coefficients based on historical data are collected for battles lasting a few hours. The coefficients can be "smoothed" over small time-step granularities, say of ten minutes or so but not finer. As a result, models employing Lanchester equations must have time-steps of at least ten minutes, or else the attrition computed by the Lanchester equations cannot be claimed to be valid. In contrast, for battles involving disaggregate-level forces, such as tanks and artillery, attrition is computed by applying historical hit-kill probabilities for each engagement. Briefly, when a tank fires a shell at an enemy, the shell has a certain probability of hitting the target. Kill probabilities are conditioned on hit probabilities. Since attrition using hit-kill probabilities is computed on a per engagement basis, it can be applied to simulated battles with millisecond time-steps.

A multi-model that involves models employing different algorithms encounters two problems: temporal inconsistency and conflicting results. Temporal inconsistency may arise if the multiple algorithms make different assumptions about time at the multiple levels, as Lanchester equations and hit-kill probabilities do. Temporal consistency caused by time-step differentials must be eliminated; we do so by assuming compatible time-steps. Conflicting results arise if the algorithms predict different outcomes for the same set of inputs. Selective viewing avoids the problem of conflicting results by executing only the detailed model. Aggregation-disaggregation encounters the problem of conflicting results; depending on the level at which a multi-model is executed, the results of an outcome may vary [HILL92B].

MREs do not address the problem of conflicting results. Designers of multi-models must resolve conflicting results caused by different algorithms at multiple levels. Joint execution of multiple models captures the combined semantics of the models, no matter what the semantics of the individual models are.

Summary of Limitations: The limitations above are expected of any approach that focusses on MRM alone. Designers must address these limitations in order to construct useful multi-models. However, addressing these limitations is outside the scope of any MRM approach, including `UNIFY`. Table 5 summarises the limitations of MREs by comparing how various MRM approaches address the above issues.

TABLE 5: Summary of Limitations of MREs

Limitations	Selective Viewing	Aggregation-Disaggregation	`UNIFY`
Identifying Attributes and Dependencies in Representations	Not addressed	Not addressed	Not addressed
Capturing Whole-Greater-than-the-Sum-of-Parts Relationships	Not supported	Possible	Possible
Resolving Conflicting Results	Not necessary	Required	Required

5.7 Chapter Summary

A Multiple Representation Entity is a technique for maintaining concurrent representations in order to achieve effective MRM. A key challenge with an MRE is maintaining consistency among its concurrent representations in the presence of dependent concurrent interactions. We assume the existence of appropriate mapping

functions for translating attributes from one representation to another, policies for resolving the effects of dependent concurrent interactions and compatible time-steps. These assumptions do not make MRM trivial, because alternative approaches continue to exhibit problems even if they make similar assumptions.

MREs satisfy the MRM requirements of multi-representation interaction and consistency among the representations. MREs eliminate many problems with previous MRM approaches. We compared `UNIFY` with alternative approaches to MRM in terms of the requirements that each approach satisfies, the assumptions made towards satisfying those requirements, and the benefits and limitations of each approach. We depict these comparisons concisely in Table 6.

MREs and techniques for maintaining internal consistency among MREs constitute `UNIFY`. A Consistency Enforcer and an Interaction Resolver for an MRE maintain consistency among the concurrent representations and resolve the effects of concurrent interactions respectively. In Chapters 6 and 7, we describe a CE and an IR in detail.

TABLE 6: Comparison among MRM approaches

		Selective Viewing	Aggregation-Disaggregation	UNIFY
Requirements	R1: Multi-representation Interaction	No	No	Yes
	R2: Multi-representation Consistency	Trivially	Possible	Possible
	R3: Cost-Effectiveness (see Chapter 9)	High Cost of Simulation	High Cost of Consistency	Balanced Costs
Assumptions (see §5.3)	Mapping functions	Required initially	Required	Required
	Policies for resolving concurrent interactions	Required	Required	Required
	Compatible time-steps	Trivial	Trivial	Required
Benefits (see §5.5)	Temporal Inconsistency	Absent	Present	Eliminated
	Mapping Inconsistency	Absent	Present	Eliminated
	Chain Disaggregation	Inherent	Possible	Eliminated
	Transition Latency	Non-existent	Possible	Eliminated
	Thrashing	Non-existent	Possible	Eliminated
	Network Flooding	High	Possibly high	Reduced
	Cross-level Interactions	Non-existent	Possible	Eliminated
Limitations (see §5.6)	Identifying Attributes and Dependencies in Representations	Not addressed	Not addressed	Not addressed
	Capturing Whole-Greater-than-the-Sum-of-Parts Relationships	Not supported	Possible	Possible
	Resolving Conflicting Results	Not necessary	Required	Required

It is of course important to try to maintain consistency,
but when this effort forces you into a stupendously ugly theory,
you know something is wrong.
— *Douglas Hofstadter,* Gödel, Escher, Bach

Chapter 6

Consistency Enforcers

For effective MRM, jointly-executing multiple models must be consistent with one another (requirement R2). In Chapter 5, we presented Multiple Representation Entities (MREs) which incorporate concurrent representations of multiple models. A *Consistency Enforcer* (CE) is a component of an MRE that maintains consistency among concurrent representations. A CE consists of an Attribute Dependency Graph (ADG) that captures dependencies among representations, and application-specific mapping functions that translate attributes. An ADG and mapping functions ensure that the relationships in an MRE hold at all observation times. In this chapter, we present ADGs, discuss how mapping functions relate to them and demonstrate the construction of a CE.

When an interaction changes the value of attributes, a CE ensures that the concurrent representations in an MRE are consistent. The operation of a CE involves traversing an ADG and invoking mapping functions to compute the changes to relevant attributes. A CE maintains internal consistency within an MRE. Constructing a CE involves:

1. Constructing an Attribute Dependency Graph

a. Assigning Nodes to Attributes

b. Assigning Arcs to Dependencies

c. Assigning Semantics to Dependencies

2. Selecting Mapping Functions

In §6.1, we describe ADGs and introduce an example in order to demonstrate step 1. Also, we introduce four classes of dependencies: cumulative, distributive, modelling and interaction. In §6.2, we discuss the mapping functions that designers must provide for their multi-models (step 2). In §6.3, we describe how a CE can enforce consistency among multiple representations by traversing an ADG and propagating the effects of an interaction. In §6.4, we present various implementation strategies for Consistency Enforcers such as spreadsheets, attribute grammars, mediators and constraint solvers. In this chapter, we assume concurrent interactions are independent, i.e., their effects can be resolved by serialization. We make this assumption in order to explain the operation of a Consistency Enforcer alone. We address dependent concurrent interactions in Chapter 7.

6.1 Constructing an Attribute Dependency Graph

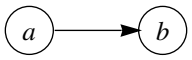


FIGURE 25: Simple ADG

An *Attribute Dependency Graph* captures dependencies among attributes in concurrent representations. When multiple models execute jointly, a change to an attribute may cause other dependent attributes to change. A dependency graph is a natural technique to capture such cause-effect dependencies among attributes. In an ADG, nodes correspond to attributes in a multi-model and arcs correspond to dependencies among the attributes. In the simple ADG shown in Figure 25, the left node corresponds to an attribute

a, and the right node corresponds to an attribute b. The arc connecting the two nodes shows that b depends on a, or a affects b. If the value of a changes, the value of b may change. If the value of b changes, there is no requirement for the value of a to change. For the relationship in the figure, a is the independent attribute and b is the dependent attribute. The ADG in Figure 25 does not show *how* b must change when a changes. A mapping function must encode how b changes when a changes. Dependency graphs such as ADGs capture cause-effect relationships in a number of contexts, for example, task execution sequences in Petri nets [PETER77], data dependencies in dataflow models [DENNIS80], method invocation in object-oriented design [RUM91] [SHLAER92], and causal relationships in logical time systems [LAM78].

Let $Model^A$ be a low-resolution model and $Model^B$ be a high-resolution model. Recalling our definitions from Chapter 3, in UNIFY, a multi-model $Model^M$ is:

$$\begin{aligned} Model^M &= \langle Rep^M, Rel^M, Int^M \rangle \\ Rep^M &= Rep^A \cup Rep^B \\ Rel^M &= Rel^A \cup Rel^B \cup Rel^{cross-model} \end{aligned}$$

$Model^M$ is consistent if Rel^M, and in turn, $Rel^{cross\text{-}model}$ hold $\forall t \in T^M$. Previous MRM approaches do not capture complex cross-model relationships that may hold at different times. In selective viewing, $\forall t \in T^M$, $Model^M(t) = Model^B(t)$ and $Rel^{cross\text{-}model} = \varnothing$. In aggregation-disaggregation, at time $t_i \in T^M$, $Model^M(t_i) = Model^A(t_i)$, and at time $t_j \in T^M$, $t_i \neq t_j$, $Model^M(t_j) = Model^B(t_j)$. $Rel^{cross\text{-}model} \neq \varnothing$ only when a representation level is transitioned, i.e., $t_i, t_{i+1} \in T^M$, $Model^M(t_i) = Model^A(t_i) \wedge Model^M(t_{i+1}) = Model^B(t_{i+1}) \vee Model^M(t_i) = Model^B(t_i) \wedge Model^M(t_{i+1}) = Model^A(t_{i+1})$.

In UNIFY, an ADG has a node for each attribute $a \in Rep^M$, and an arc for each relationship $r \in Rel^M$. Recall that $Rel^{cross\text{-}model}$ is defined as a set of relationships such

that $\forall r$: $P \rightarrow Q$, $P \subseteq Rep^A \wedge Q \subseteq Rep^B \vee P \subseteq Rep^B \wedge Q \subseteq Rep^A$. An ADG has an arc for every $r \in Rel^{cross\text{-}model}$ because $Rel^{cross\text{-}model} \subseteq Rel^M$. An ADG is a technique for describing attributes in concurrent representations, relationships among those attributes and the semantics of the relationships.

In the following sub-sections, we show how to construct an ADG for an example MRE from jointly-executing battlefield models. Our example is derived from specifications of actual battlefield models [JPSD97] [JTFP97] [RPR97]. The choice of models reflects our familiarity with the domain, not a restriction on the kind of multiple models for which ADGs are relevant. Let $Model^A$ be a platoon model, $Model^B$ be a tank model, and $Model^M$ be a multi-model incorporating these two models. A platoon in $Model^A$ has attributes for position (Pos), velocity (Vel), firepower (Fire), strength (Str), appearance (App) and formation (Form). A tank in $Model^B$ has attributes for position (Pos), velocity (Vel), hits (Hits), ammunition (Ammo), damage status (Dam) and fuel level (Fuel). Our MRE, the Platoon-Tanks MRE in Figure 26, is a platoon represented at two levels: the platoon level

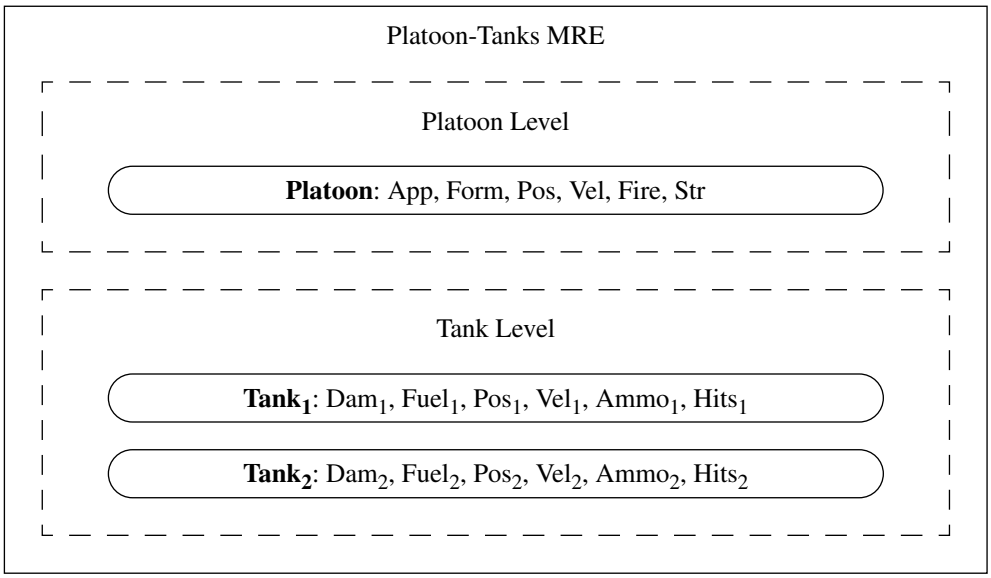


FIGURE 26: Platoon-Tanks MRE

and the tank level. Therefore, this MRE has attributes for a platoon and its constituent tanks. For ease of exposition, we assume that our platoon can be represented at the tank level by just two tanks. Attributes App, Form, Pos, Vel, Fire, Str $\in Rel^A$, and attributes Dam_1, Fuel_1, Pos_1, Vel_1, Ammo_1, Hits_1, Dam_2, Fuel_2, Pos_2, Vel_2, Ammo_2, $\mathrm{Hits}_2 \in Rel^B$. We will demonstrate the construction of an ADG for this MRE.

6.1.1 Assigning Nodes to Attributes

The first step in constructing an ADG is assigning nodes to attributes. In principle, a designer may assign a node to any set of attributes P such that $P \subseteq Rep^A \vee P \subseteq Rep^B$. A node can be assigned to any set of attributes that enables a designer to make straightforward decisions about applying the effects of interactions. For example, the designer may assign a node to the set of attributes of a tank. However, such an assignment does not aid the designer substantially in applying the effects of interactions. In practice, since interactions affect *attributes*, we expect the designer to assign nodes to attributes such as position and appearance. In a multi-model involving atoms and molecules, nodes could be assigned to atom-level attributes such as orientation and charge, and molecule-level attributes, such as orientation and valence. In a hierarchical autonomous agent model, nodes could be assigned to planner-level attributes such as absolute location and connectedness, and perception-action-level attributes such as relative location, colour and visibility. In our example, we assign a node in the ADG to every attribute in the concurrent representations of $Model^M$, i.e., every attribute $a \in Rep^M$. In Figure 27, we show all the attributes as nodes labelled with unsubscripted or appropriately-subscripted names.

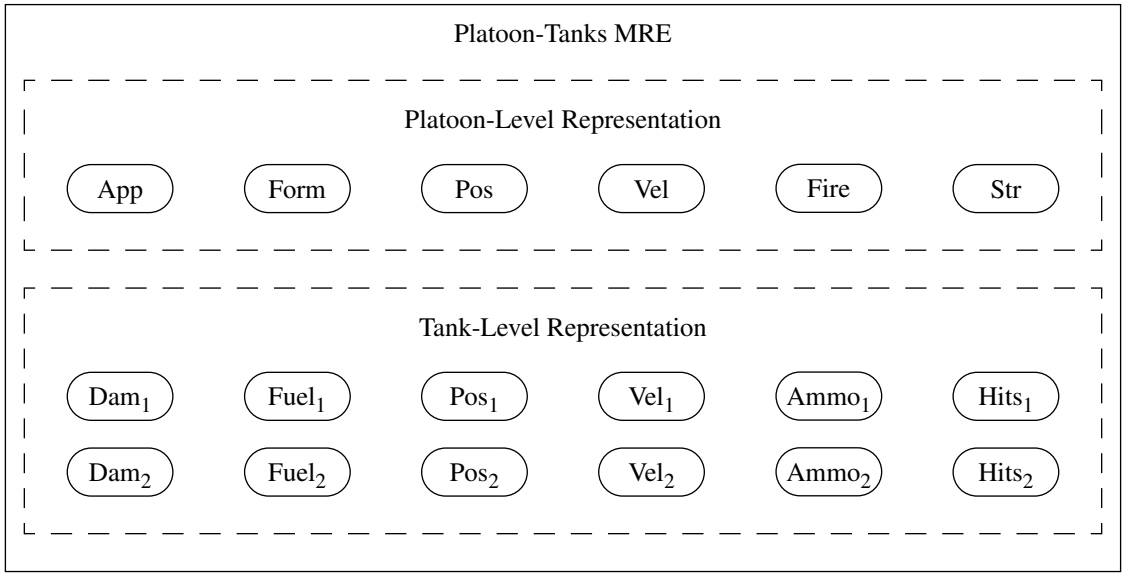


FIGURE 27: Nodes in the ADG for the Platoon-Tanks MRE

6.1.2 Assigning Arcs to Dependencies

The second step in constructing an ADG is assigning arcs to dependencies. An arc connecting two nodes represents a dependency between attributes corresponding to the nodes. Since a dependency between two attributes indicates that they are related, arcs in an ADG correspond to each relationship $r \in Rel^M$. In Figure 28, we show dependencies for our Platoon-Tanks example. The platoon position depends on each tank position and *vice versa*. The tank positions are unrelated because each tank may move independently. We identify similar dependencies among platoon and tank velocities. The firepower of the platoon depends on the ammunition levels of the tanks, and the strength of the platoon depends on the number of hits each tank has received. Likewise, the appearance of the platoon depends on the damage state of each tank and *vice versa*. The fuel level of the individual tanks is not represented at the platoon level. Unless the platoon model bases any decisions on the fuel level of the platoon, it is not necessary that the tank fuel levels be

represented at the platoon level. The platoon has a formation attribute that captures the relative positions of the tanks. The formation depends on tank positions and *vice versa*. Suppose moving out of formation may cause the platoon to appear weak. Therefore, the formation affects the appearance of the platoon. Lastly, the current positions of the platoon or tanks depend on the current values of the respective velocities.

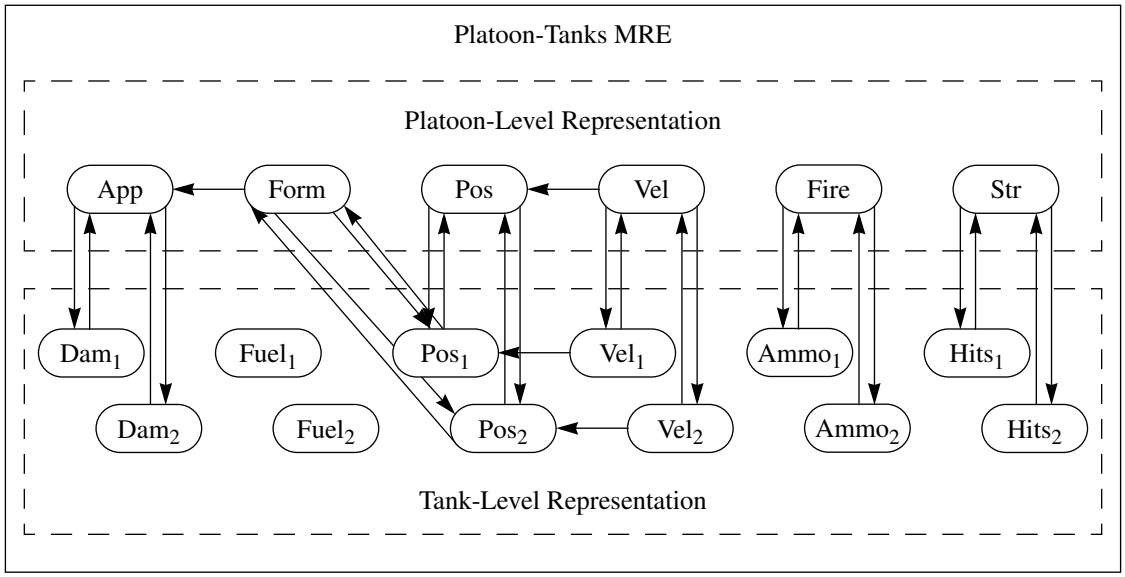


FIGURE 28: Dependencies in the ADG for the Platoon-Tanks MRE

6.1.3 Assigning Semantics to Dependencies

The third step in constructing an ADG is to assign semantics to dependencies. Assigning semantics to dependencies enables the construction of appropriate mapping functions for them. One way to assign semantics is to classify dependencies. Mapping functions associated with classes of dependencies have common requirements. Dependencies may be classified according to characteristics specific to an application. We classify dependencies in an application-independent manner into four categories: cumulative, distributive, interaction and modelling. Binary weights, fractional weights and

interaction classes are other techniques for capturing semantics of dependencies. Assigning weights to cumulative and distributive dependencies can capture how changes to an attribute contribute or distribute to other attributes. Modelling dependencies already capture semantic information, hence we do not associate any additional semantic information with them. The semantics we associate with interaction dependencies are the classes of interactions (discussed in Chapter 7). The semantics assigned to dependencies vary with applications.

6.1.3.1 Cumulative and Distributive Dependencies

Whole-to-parts and parts-to-whole relationships are common in models, for example, aggregation associations among objects in UML [ALHIR98] and OMT_R [RUM91], and relationships among objects such as part-whole, consists-of, composition, has-part and contains in other modelling methodologies [FOWLER97]. These associations and relationships usually are bidirectional, i.e., a relationship between P and Q implies another relationship between Q and P. These associations and relationships capture whole-to-parts and parts-to-whole relationships among objects; we capture similar relationships between attributes with cumulative and distributive dependencies.

Cumulative and distributive dependencies capture parts-to-whole and whole-to-parts relationships among attributes respectively. *Cumulative dependencies* are dependencies in which the value of a single attribute is influenced jointly by the value of many other attributes. For a relationship $r \in Rel^M$, $r: P \rightarrow Q$, $P, Q \subseteq Rep^M$, where $|Q| = 1$, $\forall a \in P$ and $b \in Q$, a cumulative dependency exists from a to b. *Distributive dependencies* are dependencies in which the value of a single attribute influences the value of many other attributes jointly. For a relationship $r \in Rel^M$, $r: P \rightarrow Q$, $P, Q \subseteq Rep^M$, where $|P| = 1$,

$a \in P$ and $\forall b \in Q$, a distributive dependency exists from a to b. In hierarchical models, cumulative dependencies capture relationships from disaggregate attributes to aggregate attributes, and distributive dependencies capture relationships from aggregate attributes to disaggregate attributes.

6.1.3.2 Interaction and Modelling Dependencies

Interactions cause changes to attributes. *Interaction dependencies* are dependencies between the sender of an interaction and the attributes changed directly by the interaction. An interaction $I \in Int^M$, may be viewed as a relationship r: $P \rightarrow Q$, where $Q \subseteq Rep^M$, but it is not necessary that $P \subseteq Rep^M$. An interaction dependency captures a cause-effect relationship from attributes of a sender to attributes of a receiver.

Other relationships may exist among attributes. These relationships may be inherent in the nature of the object or process being modelled, and may not be captured conveniently by cumulative, distributive or interaction dependencies. *Modelling dependencies* are dependencies that are not cumulative, distributive or interaction.

6.1.3.3 Selecting Dependencies

If a pair of attributes has a cumulative dependency between them, they may have a distributive dependency as well. A change to a part may affect the whole and *vice versa*. Let attributes a, a_1, a_2, …, a_n, b, b_1, b_2, …, $b_m \in Rep^M$. In Table 7, we list how dependency classes can be assigned to combinations of whole-to-parts and parts-to-whole relationships. The first column lists combinations of whole-to-parts and parts-to-whole relationships. The second and third columns list the attribute dependencies and their type. For the relationship $\{a\} \rightarrow \{b\}$, classifying the dependency as either cumulative or

dependency is valid since the relationship is one-to-one. One-to-one relationships are degenerate cases of both, whole-to-parts and parts-to-whole relationships.

TABLE 7: Assigning Cumulative and Distributive Dependencies

Relationship	Dependency	Class
$\{a_1, a_2, \ldots, a_n\} \rightarrow \{b\}$	$a_i \rightarrow b$	Cumulative
	$b \rightarrow a_i$	Distributive
$\{a\} \rightarrow \{b_1, b_2, \ldots, b_m\}$	$a \rightarrow b_j$	Distributive
	$b_j \rightarrow a$	Cumulative
$\{a\} \rightarrow \{b\}$	$a \rightarrow b$	Cumulative/Distributive
	$b \rightarrow a$	Distributive/Cumulative
$\{a_1, a_2, \ldots, a_n\} \rightarrow \{b_1, b_2, \ldots, b_m\}$	$a_i \rightarrow b_j$	Cumulative
	$b_j \rightarrow a_i$	Distributive

If an interaction can change an attribute, an interaction dependency exists to that attribute, i.e., $\forall I \in Int^M$, if $<a, \delta a> \in I.affects$, where δa is a change to attribute a caused by I, then an interaction dependency exists to a. Although many interaction types may change an attribute, we associate only one interaction dependency with the attribute because the identity of the independent attribute is irrelevant. Modelling dependencies have application-dependent semantics.

6.1.3.4 Properties of Dependency Classes

Our dependency classes are complete and extensible. Cumulative and distributive dependencies capture whole-to-parts and parts-to-whole dependencies, which are common in models. Interaction dependencies capture dependencies from entities outside to entities inside a model. By definition, modelling dependencies are all dependencies that are not cumulative, distributive or interaction. Although the dependency classes are complete, designers can extend them by identifying other classes of dependencies. Additional classes may refine cumulative, distributive or modelling dependencies, thus

enabling designers to specify requirements of mapping functions in greater detail. For example, the boards of a T-joint are connected rigidly, whereas the arms of a pair of pliers are connected non-rigidly. Therefore, the cumulative dependencies from the board positions to the T-joint position and from the arm positions to the pliers position can be refined into two classes: rigidly cumulative and non-rigidly cumulative. This refinement enables a designer to specify mapping functions that translate the positions in rigidly and non-rigidly cumulative dependencies differently.

6.1.3.5 Examples of Dependency Classes

By adding interaction dependencies to the ADG in Figure 28, we obtain the complete ADG shown in Figure 29. Cumulative dependencies capture the relationship from the tank positions to the platoon position. Distributive dependencies capture the converse relationship from the platoon position to the tank positions. Likewise, cumulative dependencies capture the relationship from the tank velocities to the platoon velocity, and distributive dependencies capture the converse relationship from the platoon's velocity to the tank velocities. In the same fashion, cumulative and distributive dependencies capture the relationships among other platoon attributes and tank attributes. Modelling dependencies capture the relationships from the velocities of the platoon and the tanks to the positions of the platoon and the tanks. Likewise, a modelling dependency captures the relationship from the platoon formation to the platoon appearance. An interaction dependency to each attribute captures the effects of interactions with other entities or simulation actions of the platoon and the tanks.

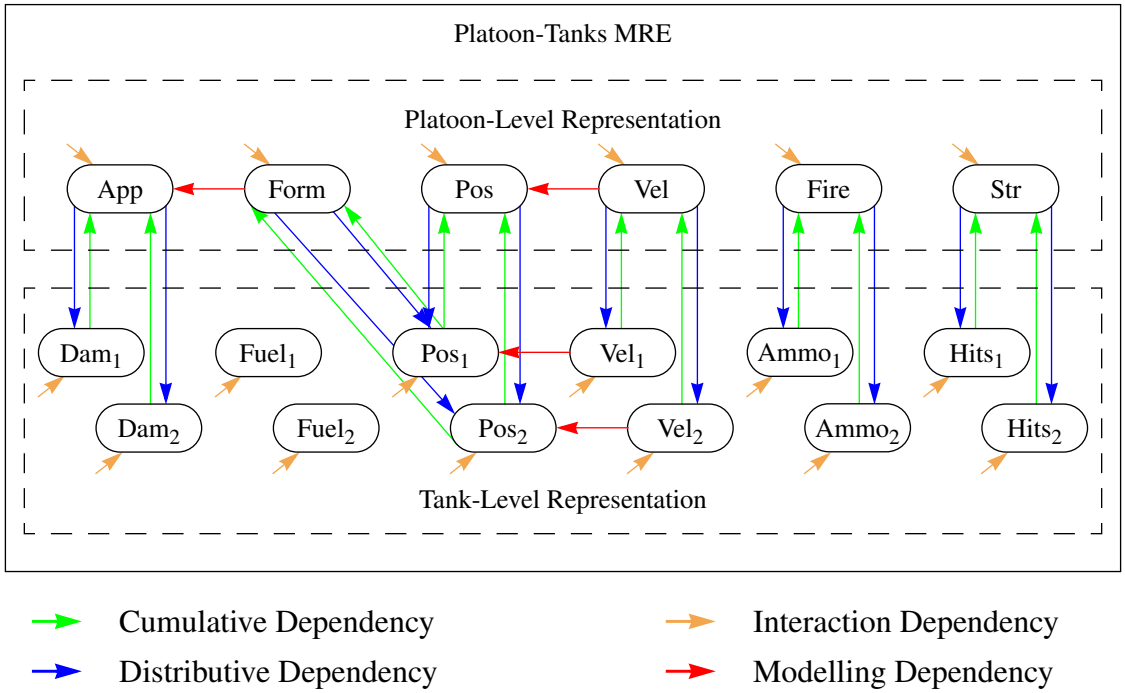


FIGURE 29: Dependency Classes in the ADG for the Platoon-Tanks MRE

6.1.3.6 Dependency Weights

Weighting dependencies with binary or fractional weights captures the semantics of *contribution*. The weight on a dependency indicates how much the independent attribute contributes to the dependent attribute. Although the assignment of weights can be construed as part of a mapping function, we view weights as an example of assigning semantics to dependencies prior to the construction of a mapping function.

Weights on Cumulative Dependencies: Weighting cumulative dependencies captures the manner in which many independent attributes affect one dependent attribute. A cumulative dependency can be weighted according to what fraction of the value of an independent attribute contributes to the dependent attribute. For example, the cumulative

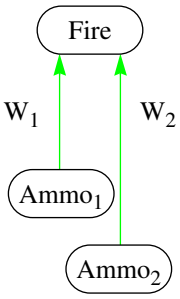


FIGURE 30: Cumulative Weights

dependencies from $Hits_1$ and $Hits_2$ to Str in Figure 29 could be weighted one, indicating that all tanks contribute their hits entirely to the platoon strength. This weighting satisfies the semantic requirement that the platoon strength is the sum of the hits of all tanks. In the case of firepower shown in Figure 30, the cumulative dependencies may be non-unity. If a tank, say $Tank_1$, fights a disaggregate-level battle, then $W_1 = 0$ indicates that $Tank_1$ expends all its ammunition in the disaggregate battle only. If $Tank_1$ could fire at both levels simultaneously (a physical impossibility, but assumed for exposition), and $Tank_1$ allocated 50% of its total ammunition for each engagement, then $W_1 = 0.5$.

Weights on Distributive Dependencies: Weighting distributive dependencies captures the manner in which one independent attribute affects many dependent attributes. A distributive dependency can be weighted according to what fraction of the change to an independent attribute propagates to the dependent attribute. For example, the distributive dependencies from the tank hits to the platoon strength in Figure 29 could be weighted as shown in Figure 31. If the platoon strength is reduced, fractions of that change propagate to the tank hits. In order to satisfy the semantic requirement that the platoon's strength is the sum of the tank hits, the sum of the propagated fractions must sum to the reduction in the platoon strength. An independent attribute may not affect all its dependent attributes uniformly. For example, either W_1 or W_2 (but not both) may be zero, meaning that a change to Str does not change the corresponding Hits. This weight could reflect a scenario in which the unaffected tank is shielded from the firepower of the enemy because of barriers, entrenchments or good defensive position.

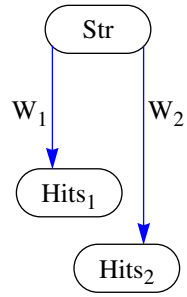


FIGURE 31: Distributive Weights

Assignment of Weights: The weights on cumulative and distributive dependencies may change during a simulation. For a battlefield simulation, weights may be assigned per engagement. Thus, strength reductions from different enemies may propagate with different sets of weights because of the nature of the enemies' firepower or their positions. In indiscriminate firing situations, weights may be assigned randomly to reflect the fog of war (see §3.4). Alternatively, weights may be assigned depending on the properties of the constituents. For example, boolean attributes signifying the visibilities of disaggregate entities are not fractions of a boolean attribute signifying the visibility of the corresponding aggregate entity. In such cases, boolean weights for distributive dependencies are more appropriate, and the product, rather than the sum, of the distributive weights must be one.

Interlinked Dependency Weights: The weights on distributive and cumulative dependencies are dependent on one another. The weights on these dependencies must be assigned with due consideration to the meaning of the combination of weights. For example, suppose a designer specifies that a tank, say Tank_1, does not fire in a platoon-level battle. Therefore, Ammo_1 does not contribute to Fire. Refining the specification further, we can say: If Ammo_1 does not contribute to Fire, then a change to Fire does not change Ammo_1. For the refined specification, a weight of zero on the cumulative dependency from Ammo_1 to Fire captures the *if*-part, and a weight of zero on the distributive dependency from Fire to Ammo_1 captures the *then*-part. Therefore, the specification above can be re-stated as: A zero-weight cumulative dependency from Ammo_1 to Fire $\Rightarrow$ a zero-weight distributive dependency from Fire to Ammo_1. If the distributive dependency is zero and the cumulative dependency is non-zero it just means

that $Tank_1$ contributed some of $Ammo_1$ to Platoon, but Platoon did not use $Ammo_1$ in this engagement. Other combinations of weights for the cumulative and distributive dependencies are possible for other attributes.

6.1.3.7 Interaction Semantics

Whether a change to an attribute occurs as a result of another entity's interaction or as a result of simulation activities performed by the MRE, the change originates from interaction dependencies. Since different interactions may change an attribute, a change to an attribute because of an interaction dependency can have different semantics. Although we associate only one interaction dependency per attribute, we say that the semantics of an interaction dependency change with the semantics of interactions. We discuss interaction semantics in Chapter 7.

6.1.4 Summary of Attribute Dependency Graphs

ADGs capture relationships among attributes in concurrent representations. Designers construct an ADG by assigning nodes and arcs to attributes and relationships in Rel^M. Next, they classify dependencies and assign semantics to them. In Figure 29, a cumulative or distributive dependency exists $\forall r \in Rel^{cross\text{-}model}$ and a modelling dependency exists $\forall r \in Rel^A$ and $\forall r \in Rel^B$. Since $Rel^M = Rel^A \cup Rel^B \cup Rel^{cross\text{-}model}$, a dependency exists in the ADG $\forall r \in Rel^M$. For other MREs, Rel^A, Rel^B and $Rel^{cross\text{-}model}$ may contain other combinations of cumulative, distributive and modelling dependencies.

In addition to the relationships in Rel^M, the ADG captures interaction dependencies. Interaction dependencies are a starting point for applying the effects of interactions. We discuss applying the effects of interactions in §6.3. After the ADG is constructed, the

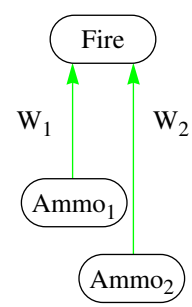
designer must choose appropriate mapping functions to perform the actual translations among attributes for each dependency. Next, we show how to select these functions.

6.2 Selecting Mapping Functions

Mapping functions translate value spaces or changes to values of attributes. An ADG indicates *which* attributes must change when an interaction occurs. Mapping functions along with an ADG indicate *how* the attributes must change. Mapping functions determine whether a relationship holds, i.e., whether the dependent attributes are consistent with the independent attributes. Determining whether attributes are consistent entails comparing them. The results of the comparison may be exact or within tolerable error.

Mapping functions translate value spaces or changes in the values of attributes. When an attribute changes as a result of an interaction, invoking appropriate mapping functions is necessary to ensure that dependent attributes change as well. Therefore, either the new value of an independent attribute or the change to its previous value must be translated to new values or changes to previous values of dependent attributes.

A mapping function may translate *value spaces* among attributes, i.e., the function has the form $\forall t_i, t_{i+1}, Q(t_{i+1}) = f(Q(t_i), P(t_{i+1}))$, where $P(t_{i+1})$ is determined by applying the changes $\Delta P(t_i)$ to $P(t_i)$. For example, a mapping function f translates tank ammunitions to platoon firepower. Here, $Q = \{\text{Fire}\}$ and $P = \{\text{Ammo}_1, \text{Ammo}_2\}$. An implementation of f is shown in Figure 32. Since cumulative



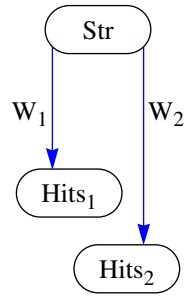
$\text{Fire} = f(\text{Ammo}_1, \text{Ammo}_2)$

$\text{Fire} = \text{W}_1 * \text{Ammo}_1 + \text{W}_2 * \text{Ammo}_2$

FIGURE 32: Mapping Value Spaces

dependencies connect the ammunitions to the firepower, a mapping function must include the contributions of each tank ammunition to compute the platoon firepower. Accordingly, the mapping function must utilise the weights on the cumulative dependencies.

A mapping function may translate *changes to values* among attributes, i.e., the function has the form $\forall t_i, \Delta Q(t_i) = f(Q(t_i), \Delta P(t_i))$, where $Q(t_{i+1})$ is determined by applying the changes $\Delta Q(t_i)$ to $Q(t_i)$. For example, a mapping function g translates a change in platoon strength to changes in tank hits. Here, $Q = \{\text{Hits}_1, \text{Hits}_2\}$ and $P = \{\text{Str}\}$. An implementation of g is shown in Figure 33. Since distributive dependencies connect the strength to the hits, a mapping function must distribute the change in the platoon strength to changes in each tank hits. Accordingly, the mapping function must utilise the weights on the distributive dependencies.



$$(\Delta\text{Hits}_1, \Delta\text{Hits}_2) = g(\Delta\text{Str})$$
$$\Delta\text{Hits}_1 = \Delta\text{Str} * \text{W}_1 \div (\text{W}_1 + \text{W}_2)$$
$$\Delta\text{Hits}_2 = \Delta\text{Str} * \text{W}_2 \div (\text{W}_1 + \text{W}_2)$$

FIGURE 33: Mapping Changes in Values

6.3 Traversing an ADG

After an ADG has been constructed and mapping functions selected, a CE can maintain consistency within an MRE by traversing the ADG and invoking the appropriate mapping functions. An interaction I may change the values of any attributes. These changes must propagate to dependent attributes. By traversing an ADG, a CE propagates *I.affects via* interaction dependencies, and *I.affects*$^+$ *via* cumulative, distributive and modelling dependencies. For each arc traversed, a mapping function computes the change to a dependent attribute as a result of a change to an independent attribute.

6.3.1 Algorithm for Traversing an ADG

Ensuring internal consistency within an MRE involves traversing an ADG when a change to any attribute occurs. The effects of an interaction can be applied by traversing an ADG and invoking appropriate mapping functions. In OMT_R, a similar concept is called propagation [RUM91]. Initially, an MRE is internally consistent; all relationships in Rel^M hold. When an interaction *I* occurs, a CE traverses an ADG starting from the nodes corresponding to the attributes in *I.affects*. *I.affects* is computed from semantic knowledge about the interaction. After the changes in *I.affects* are applied, the MRE is temporarily inconsistent. In order to regain the consistency of the MRE, its ADG must be traversed beginning from the nodes corresponding to the attributes in *I.affects*. For each arc traversed, a mapping function must be invoked to change dependent attributes.

In Figure 34, we present an algorithm for ADG traversal. The outer loop in the algorithm implicitly assumes that interactions are serialized. The first step in the loop initialises a set, `S`, which will contain the effects of an interaction *I*. The first inner loop includes *I.affects* in `S`. These effects can be represented by tuples, each consisting of an attribute and a change to it. The change to an attribute depends on the semantics of the interaction. Finally, in the second inner loop, for each unvisited element in `S`, the change to an attribute is applied, and the change to dependent attributes is computed and included in `S`. Marking an attribute as visited ensures that the effects of an interaction are not re-applied to the attribute. The change to an attribute, `a`, as a result of the interaction depends on the semantics of the attribute. Attributes dependent on `a` can be determined from the ADG. For each dependent attribute, a mapping function translates the change to the value

of the attribute. If a dependent attribute changes, a tuple consisting of the attribute and its change is included in S to account for *I.affects*$^{+}$.

```
For each interaction I
   Set S ← ∅
   For each attribute a in I.affects
      S ← S + <a, δa> // interaction effect
   For each unvisited element <a, δa> in S
      mark <a, δa> visited
      a ← function of a, δa // attribute semantics
      For each attribute d dependent on a in ADG
         δd ← function of a, δa, d // mapping function
      If δd ≠ 0 // or non-negligible
         S ← S + <d, δd>
```

FIGURE 34: Algorithm for ADG Traversal

We step through the algorithm in Figure 34 with an example. Let a tank in our example MRE receive a *move* interaction. This interaction changes the position of the tank, say Pos_2. Therefore, a tuple consisting of Pos_2 and a change to Pos_2 is included in S. When the change to Pos_2 is applied, the MRE is temporarily inconsistent. In order to regain the consistency of the MRE, a CE must traverse the ADG beginning from the node corresponding to Pos_2. The attributes that depend on Pos_2 are: Pos, Pos_1, Pos_2, Form, App, Dam_1 and Dam_2. Figure 35 shows a sub-graph of the ADG with only the nodes corresponding to attributes connected transitively to Pos_2. A CE must invoke mapping functions to translate changes to each of these attributes. For example, the change to Pos may be computed as the centroid of Pos_1 and Pos_2. If the change to Pos is non-zero, then a tuple consisting of Pos and its change is included in S. In like fashion, the CE propagates the effects of the interaction to each dependent attribute. Figure 36 shows a partial tree corresponding to the propagation of the change to Pos_2 to dependent attributes.

The algorithm in Figure 34 includes but intentionally does not make apparent intricate issues in ADG traversal, for example, cyclic dependencies and traversal path. We address these issues next.

6.3.2 Cyclic Dependencies

Cyclic dependencies among attributes may cause traversal of an ADG to never terminate. For example, when an interaction changes Pos_2, a CE changes Pos because of the cumulative dependency from Pos_2 to Pos. The CE propagates the change to Pos to Pos_1 and Pos_2 because of distributive dependencies. Since Pos_2 has already changed because of this interaction, the CE must stop the propagation of effects from Pos. If the CE does not stop the propagation of effects, the MRE may never reach a state at which consistency can be evaluated. Although an ADG may have cycles, the propagation of effects must be non-cyclic. It is reasonable for attributes to be mutually dependent. However, cycles should not prevent the graph traversal from terminating.

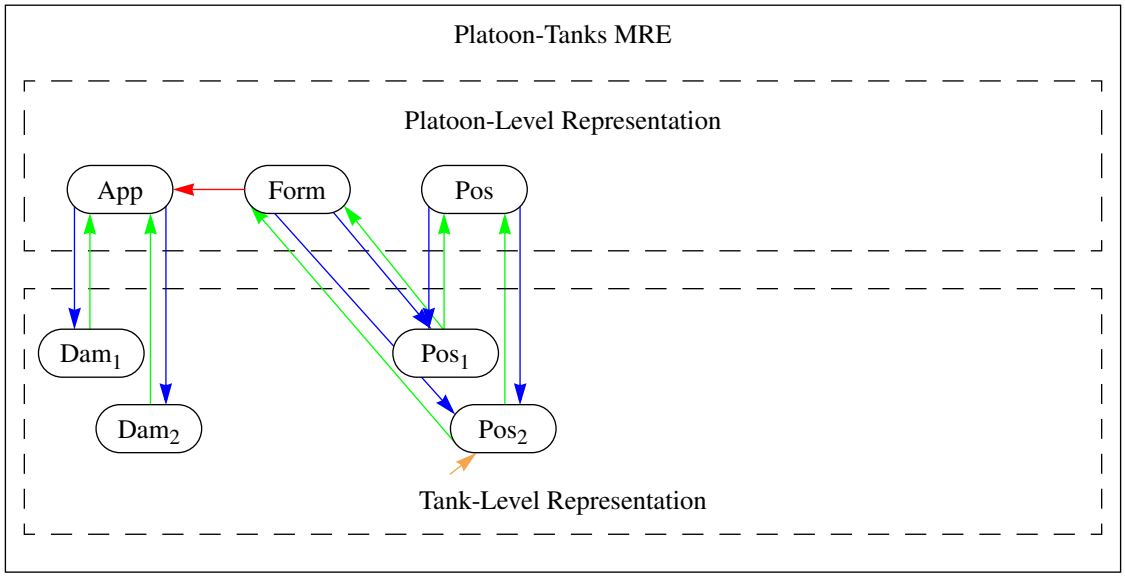


FIGURE 35: Applying the Effects of an Interaction

Reversible mapping functions break cycles in ADG traversal. If reversible mapping functions translate attributes, then the change to some attributes may be null, which breaks the cyclic traversal between them. In case of attributes for which reversible mapping functions cannot ensure zero change, the final condition in Figure 34 should be modified such that a tuple is included in `S` only when a non-negligible change occurs. Reversible mapping functions ensure that a change to Pos due to an initial change to Pos_2 does not affect Pos_2 again. Let mapping functions f and g translate Pos_2 to Pos and Pos to Pos_2 respectively. If $g(f(\text{Pos}_2)) = \text{Pos}_2$, then f and g are reversible. When an interaction changes Pos_2, f changes Pos, and g ensures that a subsequent change to Pos_2 will be zero. In Figure 34, effects are included in `S` only if necessary. Therefore, the zero change to Pos_2 is not included in `S`, breaking a cyclic traversal. When all cyclic traversals are broken, the traversal of an ADG can terminate. If ADG traversal terminates, an MRE can be consistent before the next observation point.

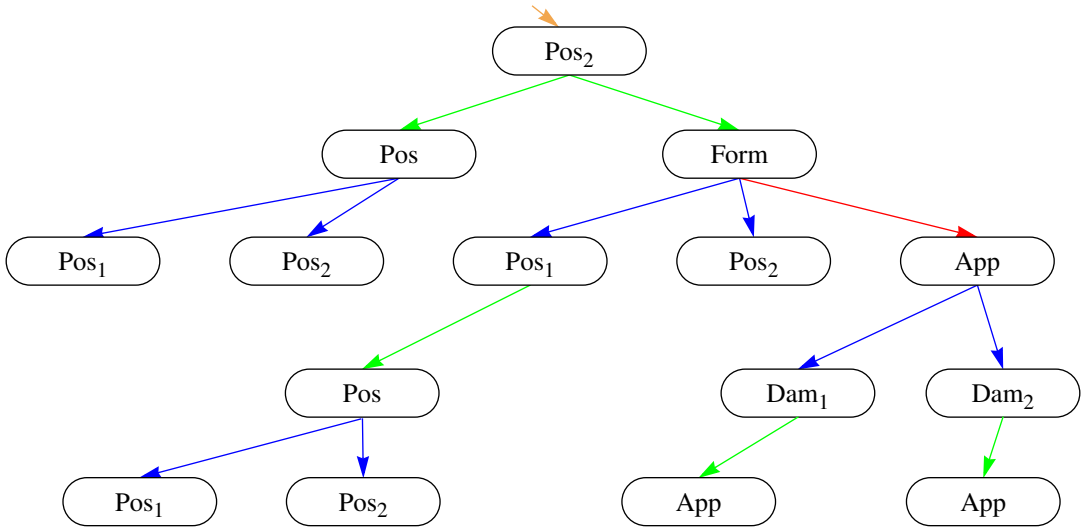


FIGURE 36: Propagation of Interaction Effects

6.3.3 Unplanned Dependencies

ADGs enable designers to identify and capture combined semantics of multiple models. Unplanned dependencies are an example of the combined semantics of jointly-executing models. An ADG captures attribute dependencies that may not have been planned by designers of the individual models. For example, the designer of the tank model may not have expected Pos_2 and Dam_1 to be dependent. However, because of transitive dependencies, these attributes are related, as seen from Figure 36.

6.3.4 Traversal Path

An issue with ADG traversal is the order in which a CE propagates the effects of interactions. When an interaction changes an attribute, a CE may change other attributes subsequently. For example, in Figure 36, if an interaction changes Pos_2, a CE must change Pos and Form. Suppose the CE changes Pos first. Next, it must change Form (because of the original change to Pos_2) and Pos_1 (because of the change to Pos). Changing Form first implies a breadth-first traversal of the ADG, whereas changing Pos_1 first implies a depth-first traversal. Other traversal orders are possible as well. Ideally, all traversal orders finally must propagate the effects in the same manner. Practically, because of errors accumulated during attribute translation, or because attribute translations are not commutative, different traversal orders may produce different results.

A breadth-first traversal is well-suited for propagating the effects of interactions. For distributive dependencies, the nature of the dependencies requires that effects propagate breadth-first. Moreover, when the comparison for consistency among attributes is inexact, i.e., they are consistent within tolerance, longer paths may accumulate errors that cause

reversibility to fail. With breadth-first traversal, a CE chooses the shortest paths between the initial attribute and dependent attributes [CORMEN89]. Intuitively, when an interaction changes an attribute, dependent attributes that are "closer" to the attribute in the ADG, i.e., reachable by fewer arcs, are affected more immediately by the interaction. Therefore, a CE should change those attributes earlier.

The algorithm in Figure 34 can be refined to mandate breadth-first traversal. `S` should be changed to a queue so that tuples are pushed the end of a queue. When selecting attributes dependent on the current attribute `a`, only attributes connected directly to `a` must be included. Including only directly-connected attributes and making `S` a queue ensure that the effects of an interaction are applied breadth-first.

For our example MRE, we show how a CE propagates the effects of a *move* interaction to attributes. We indicate the cause of each attribute change as well. Table 8 shows the effects of this interaction. The first column lists the attributes changed by the interaction. The second, third and fourth columns list the change to an attribute, the dependency that caused the change and the interaction or independent attribute for that change. The order in which we list changes to attributes corresponds to a breadth-first traversal of an ADG for our MRE, i.e., a breadth-first traversal of the graph in Figure 36.

TABLE 8: Effects of an Interaction

Attribute	Change	Dependency	From	Comment
Pos_2	$\delta P_2{}^1$	Interaction	*move*	Direct effect of interaction
Pos	δP^1	Cumulative	Pos_2	
Form	δF^1	Cumulative	Pos_2	
Pos_1	$\delta P_1{}^1$	Distributive	Pos	$\delta P_1{}^1 = 0$ — Pos_2 is unrelated to Pos_1
Pos_2	$\delta P_2{}^2$	Distributive	Pos	$\delta P_2{}^2 = 0$ — reversible mapping functions
Pos_1	$\delta P_1{}^2$	Distributive	Form	

TABLE 8: Effects of an Interaction

Attribute	Change	Dependency	From	Comment
Pos_2	$\delta P_2{}^3$	Distributive	Form	$\delta P_2{}^3 = 0$ — reversible mapping functions
App	δA^1	Modelling	Form	
Pos	δP^1	Cumulative	Pos_1	
Dam_1	$\delta D_1{}^1$	Distributive	App	
Dam_2	$\delta D_2{}^1$	Distributive	App	
Pos_1	$\delta P_1{}^2$	Distributive	Pos	$\delta P_1{}^2 = 0$ — reversible mapping functions
Pos_2	$\delta P_2{}^4$	Distributive	Pos	$\delta P_2{}^4 = 0$ — Pos_1 is unrelated to Pos_2
App	δA^2	Cumulative	Dam_1	$\delta A^2 = 0$ — reversible mapping functions
App	δA^3	Cumulative	Dam_2	$\delta A^2 = 0$ — reversible mapping functions

6.4 Possible Implementations of a Consistency Enforcer

Constructing a CE for an MRE is a reasonably straightforward task. A module for a CE may be implemented in a number of ways, as we show in the following sub-sections. We discuss broad implementation details in order to show that the CE is not a "black box" that magically solves consistency maintenance — one of the hardest problems in MRM.

6.4.1 As-Is

The most straightforward implementation of a CE is "as-is"; the ADG is instantiated as a graph data structure and mapping functions are function calls associated with each arc in the data structure. This implementation is effort-intensive for the designer but not as naïve as it first seems since it gives the designer the freedom to hand-craft relationships, mapping functions and traversal strategies that are best-suited for an application.

6.4.2 Spreadsheets

In a spreadsheet, data are organised as tables. Each spreadsheet element is addressed uniquely by row and column number. Each element may consist of a data value or a function. In the latter case, the value of the element is computed by invoking the function on data values or elements specified along with the function.

A CE can be implemented as a spreadsheet that has an element for each attribute in the ADG. The strict organisation of a spreadsheet as rows and columns is inconvenient but not restrictive. Mapping functions are specified by making some elements of the spreadsheet functions of other elements. However, typical spreadsheet functions are awkward for mapping functions. In typical spreadsheets, the function used to compute an element is indistinguishable from the value of the element, i.e., the function and the value for element change jointly. Therefore, if we change the value of an element, we automatically change the function that computes the element as well. Changing a function changes the relationship among elements in the spreadsheet, thus changing the relationship among attributes in the MRE. Changing the relationship may not have been part of the semantics of the interaction. A work-around for this problem involves using multiple elements for an attribute: one for the value and one for each relationship in which this attribute depends on other attributes. Not only is this work-around inelegant, but it also leads to circular references, i.e., elements that refer to one another. Spreadsheets such as *Excel*®* permit circular functions. Typically, such functions are invoked iteratively. In the first iteration, the values of elements are computed left-to-right top-to-bottom with initial values for the

* *Excel* is a registered trademark of Microsoft.

elements. In the next iteration, the values of the elements are re-computed left-to-right with values from the previous iteration. This process is continued until the number of specified iterations is exhausted. At the end of any iteration, including the final one, the values of some elements may not satisfy all relationships. Therefore, some related attributes may be inconsistent. Cyclic dependencies in an ADG increase the number of circular references in a spreadsheet. Finally, the traversal strategy in a spreadsheet is left-to-right and top-to-bottom, not the desired breadth-first strategy.

A spreadsheet implementation for a CE is suited only for very simple ADGs wherein cyclic dependencies are limited and left-to-right top-to-bottom traversal is sufficient to approximate breadth-first traversal.

6.4.3 Attribute Grammars

An attribute grammar enables specifying meaning to a string derived from a context-free grammar [KNUTH68] [KNUTH71]. Properties[†] associated with non-terminals, and functions associated with productions define the semantic meaning of strings. *Synthesised* properties are defined solely in terms of the descendents of the corresponding non-terminal symbol, i.e., in terms of the properties of the symbols on the right-hand side of a production. *Inherited* properties are defined solely in terms of the ancestors of the corresponding non-terminal symbol, i.e., in terms of the properties of the symbols on the left-hand side of a production. Synthesised and inherited properties are duals; synthesised

[†] Meaning is assigned to a non-terminal in an attribute grammar by associating an *attribute* with it. To avoid any confusion with our definition of an attribute — a part of a representation — we use the term *property* to mean an attribute of a non-terminal in attribute grammars.

properties alone are sufficient for attribute grammars. Attribute grammars have been used to design language-specific editing environments [HOR86]. Attribute grammars have been extended to include context-sensitive languages [REPS84].

A CE can be implemented as an attribute grammar that has a non-terminal for each attribute in the ADG. The property associated with each attribute is its value. Relationships among attributes are specified as productions in the grammar. Functions associated with each production compute the inherited properties of the non-terminals on the right-hand side of the production. A string derived according to this grammar corresponds to the effects of an interaction.

A number of factors make attribute grammars somewhat awkward for the design of a CE. First, a grammar in which all attributes are non-terminals will never terminate because there are no terminals. Second, attribute grammars disallow cyclic dependencies since such dependencies lead to infinite invocations of productions in a grammar. Third, the traversal strategy in attribute grammars is depth-first. All of these factors can be resolved by having separate grammars for each attribute. In other words, a separate grammar for each attribute in the ADG must specify how a change to the attribute affects dependent attributes. The non-terminal for each attribute is the start symbol for its own grammar. The terminals in each grammar serve merely to break dependency cycles among attributes. Thus attribute grammars can accommodate cyclic dependencies (by having separate grammars for each attribute) yet propagate effects of an interaction non-cyclically (since each grammar has no cycles).

Although the specification of mapping functions and a traversal strategy is non-intuitive, attribute grammars can be used to implement CEs.

6.4.4 Mediators

A mediator captures behavioral relationships in complex systems [SULL94]. A mediator is a first-class implementation object that realises behaviours external to an Abstract Behavioral Type (ABT). An ABT characterizes a class of objects in terms of its data, operations on data and events that trigger other behaviours.

A CE can be implemented as a number of ABTs whose relationships are realised by mediators. Each attribute in the ADG is an ABT. The value of the attribute is the datum of the ABT. Reading and writing the datum are operations on the ABT. The only event generated by the ABT is when the value of the datum changes. Mediators capture the behavioral relationships among ABTs, i.e., mediators encode the mapping functions among attributes. When an attribute ABT announces an event signifying that its datum has changed, mediators invoke the appropriate operations to ensure that relationships among all attribute ABTs hold.

The benefit of using mediators to design CEs is that consistency maintenance is decoupled from the design of the representation. Mediators, which are instantiated solely for ensuring that relationships among attributes hold at all times, free designers of representations from the concerns of consistency maintenance. Mediators must be designed carefully to ensure that the desired graph traversal strategy is realised by the appropriate attribute ABT events.

6.4.5 Constraint Solvers

Dependencies among attributes may be viewed as constraints [ALLEN92] [HILL92A] [HORR92]. A constraint restricts the range of a dependent attribute. In the absence of any

constraint, the range of a dependent attribute encompasses all values permitted by the type of the attribute. In the presence of a constraint, the range of a dependent attribute is limited by the relationship between the dependent and independent attributes.

A CE can be implemented as a constraint solver. The attributes in an ADG can be the symbols in a constraint-solving system. Mapping functions can be implemented using unification. Constraints define relationships among attributes as well as legal ranges for values of attributes. Many constraint-solving systems solve constraints among boolean or even numerical variables. Constraint solving in the Herbrand universe, which is the union of all symbols in a system, can be complex [FRÜH92A] [FRÜH92B] [JAFFAR94] [VAN96]. However, constraint systems can be simplified in many ways, such as incorporating optimizations [MARR93], exploiting constraint independence [GARCÍA93], using incremental constraints [FREE90], and building linear systems of equations that can be solved in polynomial time for numerical variables [JAFFAR92] [CORMEN89].

A general constraint solver may be too powerful for the relationships among attributes. We expect the relationships among attributes in a multi-representation model to be simple relationships. Since the multiple models represent the same object or process, typically, the relationships are those of equality (within tolerable error), whole-to-parts or parts-to-whole. Accordingly, the constraints within an MRE may be solved relatively simply. Therefore, a constraint solver specific to the domain of the attributes of the multiple representations would be suited for the design of a CE.

6.5 Chapter Summary

A Consistency Enforcer (CE) maintains internal consistency within an MRE. A CE consists of an Attribute Dependency Graph (ADG) and mapping functions. A CE may be implemented in a number of ways, such as spreadsheets, mediators and constraint solvers.

An ADG captures dependencies among attributes in concurrent representations. Individual attributes and the dependencies among them are the nodes and arcs in an ADG. We classify dependencies into four categories: cumulative, distributive, interaction and modelling. Semantics associated with dependencies capture semantics of relationships among attributes. Classifying dependencies and assigning semantics to them aids the construction of appropriate mapping functions that translate attributes. Traversing the ADG propagates the effects of an interaction to all dependent attributes. When an interaction changes the value of any attribute, traversing the graph and invoking the mapping functions associated with each arc can make the MRE consistent again.

Mapping functions encode application-specific translations of values and changes to values among attributes. Mapping functions must translate attributes and changes to attributes. Also, mapping functions must be composable and reversible and must complete their translations before the next observation point.

As long as single interactions occur or concurrent interactions are always serialized, ADGs and mapping functions maintain consistency in an MRE. In the next chapter, we show the design of an Interaction Resolver to resolve the effects of concurrent interactions. When concurrent interactions occur, we utilise semantic information about the interactions in order to resolve any dependencies among them. The CE applies the effects of the resolved concurrent interactions.

Let no act be done at haphazard, nor otherwise
than according to the finished rules that govern its kind.
— Marcus Aurelius Antonius

Chapter 7

Interaction Resolvers

For effective MRM, the effects of dependent concurrent interactions must be resolved in accordance with model requirements. Often, concurrent interactions may have dependent effects, for example, precluding or enhancing the effects of one another. Traditionally, the effects of concurrent interactions have been resolved by serialization, in which the interactions are ordered arbitrarily. However, serialization is often inappropriate because it isolates even those interactions whose effects must be applied concurrently. Other policies, such as combining or ignoring some or all interactions, do not isolate the interactions and may be more suitable for resolving dependent effects.

In Chapter 5, we presented Multiple Representation Entities (MREs). An *Interaction Resolver* (IR) is a component of an MRE that encodes policies for resolving the effects of concurrent interactions. Since specifying policies for all possible concurrent interactions can be complex, we present a taxonomy consisting of classes of interactions. We assume that designers of multi-models understand the semantics of interactions in their application well enough to classify interactions and formulate policies for resolving

concurrent instances of classes of interactions. We present example policies for resolving classes of concurrent interactions. Our taxonomy enables a designer to choose appropriate policies for resolving concurrent interactions.

We describe interactions in §7.1. In §7.2, we discuss serialization and its alternatives. In §7.3, we motivate the need for policies other than serialization. In §7.4, we explore the problem of dependent concurrent interactions by means of an abstract application. We start with a simple system, add one dependency among its components, and study the effect of single, and subsequently, multiple interactions. We show how resolving the effects of concurrent interactions can be a complex design issue. In §7.5, we present a taxonomy to classify interactions based on intrinsic characteristics of interactions we encountered often in models. These characteristics lead naturally to policies for classes of interactions. We present example policies in §7.5. We describe the operation of an Interaction Resolver for an example MRE in §7.6.

7.1 Interactions

Entities communicate with one another or influence one another by means of interactions. As described in §3.2, an entity changes its own or another entity's behaviour by means of an interaction. Interactions are a fundamental part of any useful model because they connect the model to its environment. We regard a communication between any two entities as an interaction.

Interactions are ubiquitous — they may be physical occurrences such as movement, a temperature increase or an explosion, or some sort of communication, such as a television broadcast, a dissertation submission or an order received from a superior. Examples of

interactions are database transactions and operations [ESWA76]; processor interrupts; cache operations [HENN96]; reads and writes to shared memory in parallel processing systems; operations, events and actions in object-oriented and process modelling [RUM91] [SHLAER92] [ALHIR98]; method invocations and function calls in object-oriented systems; messages in distributed processing systems and logical time systems [LAM78]; accesses to a blackboard [ERMAN80]; and exceptions in programming languages [GOOD75]. We include all of these interactions as well as changes an entity makes to its own state in our definition of interactions. Since we are concerned only with the effects of interactions, we consider specific techniques for implementing interactions to be irrelevant to our work.

A model that permits concurrent interactions requires a policy to resolve any dependencies among interactions and a mechanism to implement the policy. The traditional policy for resolving the effects of concurrent interactions is serialization.

7.2 Serialization

Serialization, the traditional policy for resolving the effects of concurrent interactions, involves applying those effects in sequential order, i.e., one after another. Serialization is a valid policy for resolving the effects of concurrent interactions in many domains, for example, databases. Consider the clients and server in the system in Figure 37.

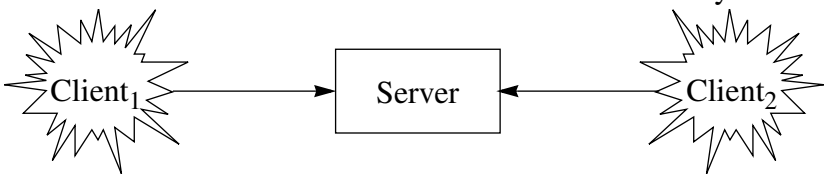


FIGURE 37: Clients and Server

Transactions from a client to the server are interactions, indicated by arrows. If only one client interacts with the server at any given time, the server returns to a valid state trivially

at the end of each interaction. If multiple clients interact with the server concurrently, ensuring that the server returns to a valid state is non-trivial.

Consider two interactions, *I* and *J*, independently issued to Server by Client_1 and Client_2 respectively. *I* and *J* each consists of operations, i.e., reads and writes, to variables *a* and *b*, denoted by $\mathtt{R}(a)$, $\mathtt{R}(b)$, $\mathtt{W}(a)$ and $\mathtt{W}(b)$.

I: $\mathtt{R}(a)\,\mathtt{W}(a)\,\mathtt{R}(b)$

J: $\mathtt{R}(b)\,\mathtt{W}(b)\,\mathtt{W}(a)$

Server's state will be as if Client_1 issued *I* to Server and when *I* completed, Client_2 issued *J* to Server, or the other way around. Each client may not be aware of the other's presence since the system guarantees that its behaviour will be as if each client is the only client interacting with the server. This property of the system's behaviour, called *isolation*, is one of the ACID properties for database transactions [HAER83].

The actual order in which operations occur on Server is called a *schedule*. In a *serial schedule*, interactions are ordered one after another [PAPA86]. A serial schedule ensures that clients interact with the server in isolation. Z_1 and Z_2 below are serial schedules for *I* and *J*. For clarity, we underline *I*'s operations in every schedule.

Z_1: $\underline{\mathtt{R}(a)\,\mathtt{W}(a)\,\mathtt{R}(b)}\,\mathtt{R}(b)\,\mathtt{W}(b)\,\mathtt{W}(a)$

Z_2: $\mathtt{R}(b)\,\mathtt{W}(b)\,\mathtt{W}(a)\,\underline{\mathtt{R}(a)\,\mathtt{W}(a)\,\mathtt{R}(b)}$

When *I* and *J* occur concurrently, the system must control how these interactions change Server. Since *I* and *J* are concurrent, their operations may interleave. A possible interleaved schedule for *I* and *J* is Z_3 below*. Z_3 is not a serial schedule because *I* and *J* are not ordered one after another.

---

* We assume that the individual operations, i.e., reads and writes, are indivisible and atomic.

Z_3: $\underline{\mathtt{R}(a)}\,\mathtt{R}(b)\,\underline{\mathtt{W}(a)}\,\mathtt{W}(b)\,\mathtt{W}(a)\,\underline{\mathtt{R}(b)}$

A schedule is *serializable* if it is equivalent to a serial schedule for some definition of equivalence [ESWA76]. If Z_3 is equivalent to Z_1 or Z_2, Z_3 is a serializable schedule. *Serialization* is a policy that resolves concurrence by permitting only serializable schedules, i.e., by ordering or interleaving concurrent interactions appropriately. Concurrency control mechanisms, such as locking and time-stamp ordering are used to implement serialization.

Serialization has been chosen as a policy for resolving interactions in database systems because it satisfies clients' expectations of isolation yet permits concurrence [PAPA86] [BERN87]. Isolation assumes that client interactions are not predicated on one another, i.e., they are independent of one another. Serialization isolates client interactions.

Some researchers have recognised that serialization can be too strict for many concurrent interactions. In advanced databases, serialization can reduce concurrence significantly. Accordingly, researchers have proposed alternative policies that relax or extend serialization yet maintain isolation. These policies utilize varying levels of semantic information about transactions in order to increase concurrence yet maintain database consistency. Semantic information has been utilized for scheduling long and short transactions [BRAHMA90]; extending and relaxing serialization [BARG91]; applying counter-transactions [GARCIA83]; commuting interpreted operations on abstract data types [WEIHL88]; aborting conflicting transactions [BARG91]; and recovering database states [BADRI92][†]. In general, serialization is considered correct but too strict, and alternative criteria relax or extend serialization in order to permit increased concurrence [BERN81] [LYNCH83] [MUNSON96] [KORTH88] [THOM98]. Moreover, isolation of transactions is

considered a desirable property of database systems. Next, we discuss situations where isolation may be undesirable.

7.3 Abandoning Isolation

For some applications, the system must not isolate concurrent interactions since they may be dependent on one another. Serialization and alternative policies that relax or extend serialization isolate interactions. Therefore, they cannot be correct policies for resolving the effects of dependent concurrent interactions. Correct policies for these interactions must provide alternatives for isolating the interactions.

In the following examples, *not* isolating concurrent interactions, i.e., abandoning isolation, enables resolving their dependent effects correctly. Consider entities E_1 and E_2 that concurrently write to an attribute v with the interactions E_1.`W`(v, …) and E_2.`W`(v, …). The ellipses denote other interaction parameters. A sequential order for these interactions could be E_1.`W`(v, …) followed by E_2.`W`(v, …) or E_2.`W`(v, …) followed by E_1.`W`(v, …).

In a model of a billiards table, E_1 and E_2 could be ball entities and v could be the velocity of a ball. The two interactions could be E_1.`W`(v, δv_1) and E_2.`W`(v, δv_2), where δv_1 is a change in v caused by E_1 and δv_2 is a change in v caused by E_2. The correct policy to resolve these two interactions is to change v by the vector addition of δv_1 and δv_2. Serializing these interactions may be incorrect for a number of reasons as discussed below.

---

† A detailed analysis of each correctness criterion and policy presented for databases would take up too much time and space. Over 100,000 pages of new material are published every year in databases alone [DATE95].

Let $\oplus$ denote vector addition. v_1, v_2 and v_3 are three possible outcomes of adding δv_1 and δv_2 to the original value v_0 of the velocity v.

$$v_1 = (v_0 \oplus \delta v_1) \oplus \delta v_2$$

$$v_2 = (v_0 \oplus \delta v_2) \oplus \delta v_1$$

$$v_3 = v_0 \oplus (\delta v_1 \oplus \delta v_2)$$

The parentheses show the order in which the interactions take effect. v_1 and v_2 are computed by serializing the two interactions. In contrast, v_3 is computed by combining the two interactions before applying them to v. Mathematically, $v_1 = v_2 = v_3$. However, when executing a model, the results of these orderings can differ. For example, δv_1 and δv_2 may be so small that adding them to v_0 individually does not change the velocity v. However, δv_1 and δv_2 combined may be sufficient to change v. In such a case, $v_1 = v_2 \neq v_3$. This thresholding anomaly may occur because of low precision in the representation of v. Another instance of thresholding could be that δv_1 and δv_2 can overcome the inertia of the entity with velocity v when combined, but not individually. As another example, suppose an entity E_3 continuously plots the trajectory of the ball with velocity v. If v changes to v_1 or v_2, E_3 will plot two changes, whereas if v changes to v_3, E_3 will plot only one change. This example is an instance of temporal inconsistency. v_1 and v_2 are computed by serialization, whereas v_3 is computed by combination. For this model, combination is a more meaningful policy than serialization.

In a model of an autonomous agent, E_1 could be a planner that pre-determines the steps to fulfill the agent's goal, E_2 could be a perception-action (PA) system that observes and acts on the agent's environment, and v could be the visibility of an obstacle. The two interactions could be E_1.`W`(v, `yes`) and E_2.`W`(v, `no`), implying that the planner reports that

the obstacle can be seen, whereas the PA system reports that the obstacle is hidden. Serializing these interactions causes the final value of v to be either `yes` or `no` arbitrarily. However, applying E_2's interaction and ignoring E_1's interaction may be a more reasonable, if pessimistic, policy to resolve these interactions. Alternatively, applying E_1's interaction and ignoring E_2's interaction may also be a reasonable, if optimistic, policy. Another reasonable policy may be to construct a belief system that assigns weights to the two interactions for a final value of v that is not bi-modal. Ignoring one or the other or weighting both interactions are policies that ensure meaningful behaviour when these interactions occur concurrently.

In a model of a chemical reaction, E_1 could be an acid entity, E_2 could be a catalyst entity, and v could be the volume of a by-product retrieved at the end of the reaction. The two interactions could be $E_1.\mathtt{W}(v, \delta v_1)$ and $E_2.\mathtt{W}(v, \delta v_2)$, where δv_1 and δv_2 are increases in the value of v when E_1 and E_2 are added. In chemical reactions, it is well-known that adding a catalyst can increase the rate of a reaction tremendously. As a result, the final change in v may be more than $\delta v_1 + \delta v_2$. Serializing the interactions does not capture the cooperative nature of these interactions. If the interactions are serialized, then either the model's representation must be augmented with an attribute that keeps track of whether the acid or catalyst has been added previously, or the model must capture the effects of adding a catalyst — an increase in the surface area of the reaction — at a finer level of detail. Alternatively, a special policy can be formulated to increase v appropriately if these concurrent interactions occur.

In the above examples, serializing concurrent interactions produces unintended effects. Isolating them from one another produces effects that are semantically incorrect.

Since serialization and alternative policies that relax or extend serialization isolate interactions, none of them is a correct policy for resolving them. These interactions are dependent particularly because they are concurrent. Therefore, they require correctness criteria that abandon isolation. The correctness criteria for dependent concurrent interactions are application-specific. Next, with the help of an abstract application, we show how resolving the effects of dependent concurrent interactions by abandoning isolation makes the design of a system complex.

7.4 Switches — A Simple System

We use a simple system of switches as an abstraction for models with concurrent interactions. We add constraints to the initial model, explaining the effort required to design the corresponding system. Next, we introduce dependent concurrent interactions and show how designing such a simple system becomes complex. We argue that the effects of dependent concurrent interactions must be resolved in an organised manner.

7.4.1 Unconstrained System

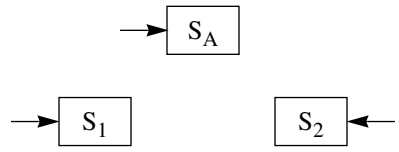


FIGURE 38: Switches

We begin with an unconstrained system. Consider the switches S_A, S_1 and S_2 in Figure 38, each with two states: on (or 1) and off (or 0). A client may turn a switch on or off by an interaction (shown by an arrow). The state of the system is an ordered triplet, individual triplet elements being the states of S_A, S_1 and S_2 respectively. In the state transition diagram in Figure 39, an oval is a possible state of the system, a solid arrow is a state transition caused by turning one switch on, and a dashed arrow is a state transition caused by turning

one switch off. Transitions that cause the system to begin and end in the same state, for example, turning S_1 off in the state [0 0 0], are not shown in Figure 39 to reduce clutter. Since the switches are independent, all possible states are present in the state diagram.

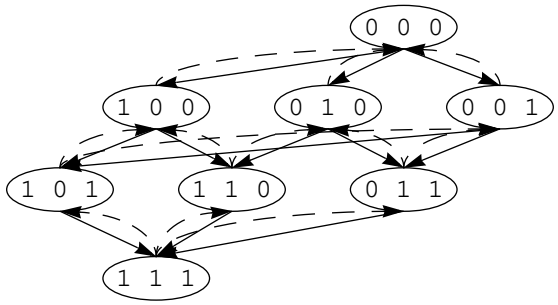


FIGURE 39: State Transition Diagram

7.4.2 Constrained System

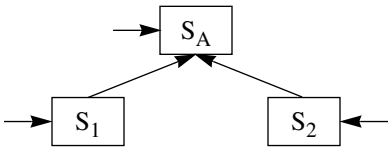


FIGURE 40: Constrained

Most practical systems are constrained, i.e., there exist relationships among components of the system. Accordingly, we add a constraint to our switches:

If S_1 and S_2 are both on, then S_A must be on.

This constraint can be re-written as $(S_1 = 1) \wedge (S_2 = 1) \Rightarrow (S_A = 1)$. As a result of this constraint, the switches are no longer independent. Figure 40 shows the new version of the switch system with the constraint depicted by arrows between the switches. The arrows merely depict a dependency between switches without outlining the nature of the dependency. The new set of valid states for the system is a subset of the old set of valid states. Figure 41 shows the new set of valid states. The crossed-out state does not exist in the new system.

S_A	S_1	S_2
0	0	0
0	0	1
0	1	0
~~0~~	~~1~~	~~1~~
1	0	0
1	0	1
1	1	0
1	1	1

FIGURE 41: New

Usually, constraints reduce the possible states of a system, i.e., some states in the unconstrained state transition diagram become unreachable. All transitions going into those states must be redirected elsewhere. The implications of the reduction in the set of valid states on the state transition diagram are shown in Figure 42. The oval corresponding to the state [0 1 1] has been removed since that state can never be reached. The outward arrows from that state have been removed since transitions from an unreachable state are meaningless (unless error recovery is desired). The arrows from the states [0 1 0] and [0 0 1] to [0 1 1] have been redirected to [1 1 1] in accordance with the constraint. However, the constraint does not indicate which state to transition from [1 1 1] if only S_A is turned off. In theory, it is possible to transition to any of the seven states (or even a hitherto absent state) in such a situation. However, let us abide by the constraint as far as possible. The following are re-statements of the constraint.

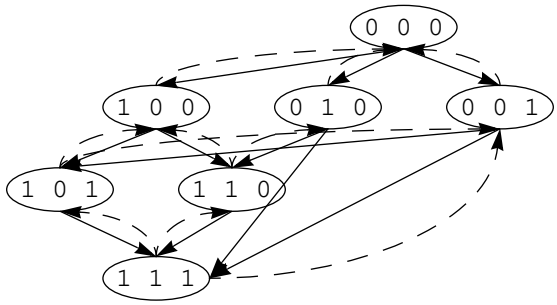


FIGURE 42: Constrained State Transition Diagram

$(S_1 = 1) \wedge (S_2 = 1) \Rightarrow (S_A = 1)$

$\neg((S_1 = 1) \wedge (S_2 = 1)) \vee (S_A = 1)$ [Implication rule]

$\neg(S_1 = 1) \vee \neg(S_2 = 1) \vee (S_A = 1)$ [DeMorgan's laws]

$(S_1 = 0) \vee (S_2 = 0) \vee (S_A = 1)$ [Switch states]

$(S_A = 1) \vee (S_1 = 0) \vee (S_2 = 0)$ [Re-arrangement]

$\neg(S_A = 1) \Rightarrow (S_1 = 0) \vee (S_2 = 0)$ [Implication rule]

$(S_A = 0) \Rightarrow (S_1 = 0) \vee (S_2 = 0)$ [Switch states]

The last statement suggests what to do when S_A is turned off while S_1 and S_2 are on. In order to keep transitions deterministic, we choose [0 0 1] arbitrarily as the state to transition from [1 1 1] in case S_A is turned off, i.e., we turn S_1 off.

State transition diagrams describe a model effectively when sequences of interactions occur. The effects of each interaction are captured by appropriate transitions. Since a state transition diagram can never put the system in an inconsistent state, every interaction can take effect without violating any constraint. Concurrent interactions, whether dependent or not, introduce problems with state transition diagrams, as we show next.

7.4.3 Dependent Concurrent Interactions

In order to demonstrate the effects of dependent concurrent interactions that cannot be serialized, we add new transitions. Consider the switch system from §7.4.2, with two concurrent interactions. Let the system be in the state [0 0 1], and let the two interactions be turning S_A off and turning S_1 on. If we serialize the interactions, turning S_A off before turning S_1 on results in the transitions [0 0 1] → [0 0 1] → [1 1 1], while turning S_1 on before turning S_A off results in the transitions [0 0 1] → [1 1 1] → [0 0 1]. The order in which the concurrent interactions are serialized determines the final state of the system. If the final state is immaterial as long as the system stays in a valid state, i.e., a state present in the state transition diagram, then serialization is correct but non-deterministic.

For deterministic behaviour, we add other state transitions that capture the effects of concurrent interactions. In Figure 43, we add a transition between [0 0 1] and [0 1 0]. The semantics of this transition could be, for example, that if S_A is turned off and S_1 is turned

on *concurrently* in the state [0 0 1], then transition directly to state [0 1 0]. The fact that the interactions were concurrent caused this transition, and the final state of the transition is different from that if the two interactions were serialized.

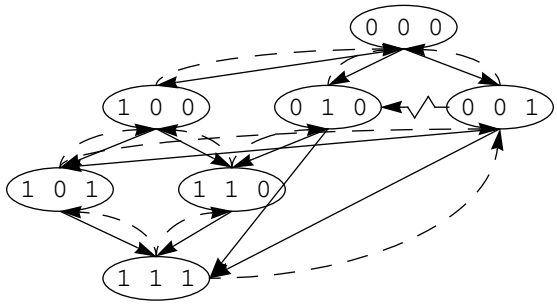


FIGURE 43: Transitions on Concurrent Interactions

7.4.4 Complexity

We desire systems to behave predictably no matter what interactions occur and how they occur. Accordingly, sequential interactions as well as concurrent interactions must have predictable results. A brute-force approach to resolving the effects of all possible concurrent interactions is exponential in complexity. Therefore, a means of encoding the dependencies among interactions is necessary.

For the switches system in §7.4.2, given the different kinds of interactions (six kinds, turning one of the switches on or off) and the number of different states (seven states), an exponential number of transitions are possible on concurrent interactions. In the worst case, the total number of transitions for the switch system is: $(2^{\text{number of interaction types}} - 1) \times \text{number of states} = (2^6 - 1) \times 7 = 441$. This calculation assumes that concurrent interactions of the same kind can be serialized without changing their effect. In other words, concurrent multiple occurrences of the interaction to turn S_1 off, for example, can be serialized. Nevertheless, even in our simple system, the number of transitions is large.

Applications with more attributes, some non-Boolean, are likely to have many more states than our simple system. Consequently, the number of transitions can grow further. However, a number of mitigating factors can reduce the number of state transitions for a system. In the switch system, in order to reduce the number of possible transitions, we stipulated that multiple occurrences of the same interaction can be serialized. Another reasonable assumption is that a switch client will not send concurrent on and off interactions to its switches. This assumption reduces the number of transitions to the product of the number of states and the number of all possible concurrent interactions. The latter number is the sum of concurrent interactions occurring in all combinations of threes, twos and ones. Therefore, the total number of transitions is: $\sum_{i=1}^{3} \binom{3}{i} \times 2^i \times 7 = 182$. This number of transitions is an upper bound, because we assume that no set of concurrent interactions is serializable.

Applications must exhibit predictable behaviour when concurrent interactions occur. Serialization is an example of predictability. However, as we have seen in §7.3, serialization fails to resolve dependent concurrent interactions correctly, because it assumes that the interactions can be isolated. Another example of predictability is commutation, wherein the effects of commutable interactions are the same regardless of the order in which they are applied [ROSSER82]. Since commutation also assumes that interactions can be isolated, it cannot resolve the effects of dependent concurrent interactions correctly. When dependent concurrent interactions occur, predictability can be achieved by encoding transitions in rigorous formulæ. In such an approach, the behaviour of the system when any set of concurrent interactions occur must be encoded *a priori*. Such an encoding is similar to specifying transitions in a state diagram for every possible

set of concurrent interactions. As we have shown with our simple switches system, specifying all possible transitions can become a complex task.

We encode semantic information in interactions in our technique for predictable behaviour when dependent concurrent interactions occur. Our technique does not isolate interactions, and does not incur the complexity cost of specifying all transitions *a priori*.

7.5 A Taxonomy of Interactions

The effects of dependent concurrent interactions are application-specific. Specifying policies for resolving the effects of every set of interactions that may occur concurrently is a complex design task. However, specifying policies for resolving the effects of *classes* of interactions can be less complex. We discuss the properties of a good taxonomy of interactions. MRM designers may classify their interactions into any taxonomy that exhibits these properties. We present and justify one such taxonomy consisting of four classes of interactions. Our taxonomy is based on semantic characteristics of interactions we encountered often in models. Also, we present policies for resolving the effects of classes of concurrent interactions.

7.5.1 Properties of a Taxonomy of Interactions

A good taxonomy exhibits the following properties [AMO94] [HOW97]:

- *mutually exclusive*: classes do not overlap
- *exhaustive*: classes jointly cover all possible members
- *unambiguous*: classification is independent of the classifier
- *repeatable*: subsequent trials lead to same classification

- *accepted*: logical and intuitive classes
- *useful*: must lead to insights in particular field

MRM designers may choose any taxonomy of interactions as long as it exhibits the above properties. Traditional taxonomies of interactions, for example, reads *versus* writes or serializable *versus* non-serializable, may not exhibit these properties.

A straightforward classification of interactions is as reads or writes. This classification does not exhibit the property of usefulness because there is inadequate semantic information associated with the classes to resolve the effects of concurrent interactions. When writes occur concurrently, we cannot determine whether the co-occurrence was a happenstance of model execution or whether the writes are simultaneous events. In the former case, the writes are independent and indistinguishable from their sequential occurrence, while in the latter case, they may be dependent concurrent interactions and must be resolved accordingly.

We rejected classifying interactions as serializable *versus* non-serializable. Such a classification does not aid us in resolving non-serializable interactions. Moreover, serializable and non-serializable are relative classes. An interaction may be serializable with respect to another interaction, but non-serializable with respect to yet another. Therefore, the same interaction falls into both classes, implying that the chosen characteristic does not partition interactions into exclusive classes. In other words, this taxonomy does not exhibit the property of mutually exclusive classes.

In §7.5.2, we present a taxonomy of interactions. We identify four characteristics of interactions: *request*, *response*, *certain* and *uncertain*. By combining these characteristics, we identify four classes of interactions: Types 0, 1, 2 and 3. We were able to categorise

interactions encountered in a number of models into these classes. Other characteristics of interactions may exist, and if identified, may introduce new classes of interactions, which may lead to new policies or refinements of our policies for resolving the effects of dependent concurrent interactions. We evaluate our taxonomy in §7.5.3.

7.5.2 Interaction Characteristics and Classes

We present four interaction characteristics and four classes of interactions that we have defined. We show how to classify interactions based on semantic characteristics. We identify four high-level semantic characteristics of interactions. These characteristics are application-independent, i.e., they are not specific to any application. The characteristics themselves are well-known; however, using them to classify interactions is novel. We identify four interaction classes from these characteristics of interactions.

7.5.2.1 Request and Response

Interactions may be requests or responses. Request interactions are concerned with an entity soliciting some behaviour from another entity. For example, when an entity queries the status of another entity, the former sends the latter a request interaction. Likewise, if an officer entity orders a soldier entity to fire, the former sends the latter a request interaction. Response interactions are concerned with an entity responding to an action generated as part of a model's behaviour, for example, a request. Responses may not be solicited explicitly, i.e., a response may not have a request associated with it. For example, a status update is a response interaction. Likewise, billiard ball entities may send one another response interactions generated because of a collision.

The distinction between request and response interactions is temporal. A request interaction is made regarding a future action. A response interaction is made regarding an action in the past. An interaction may be a request or a response, but not both[‡].

- *Request*: An interaction concerned with eliciting future behaviour from an entity.
- *Response*: An interaction concerned with the effects of an action in the past.

7.5.2.2 Certain and Uncertain

Interactions may or may not have the desired outcomes. Certain interactions have predictable outcomes. For example, when billiard ball entities collide, the outcome of their interaction is predictable because of physical laws. Likewise, when an acid entity is added to an alkali entity, the outcome of their interaction is predictable because of chemical laws. Uncertain interactions are those whose outcomes are not predictable. For example, a request for information may not always be satisfied, or satisfied truthfully. Likewise, a request to perform an action is not guaranteed to be satisfied.

Uncertainty in interactions may be defined along a continuum. For example, interactions may be distinguished on a scale with completely certain interactions at one

---

[‡] Interactions cannot refer to actions in the present. One explanation is that the sender may not know when an interaction may be received. Therefore, the sender cannot base the effects of an interaction on actions that will happen precisely during the time-step that a receiver receives the interaction. Another explanation is that we can think of a time-step as having two phases: a *send-receive* phase during which interactions are sent and received and a *perform* phase during which the effects of interactions are applied. If the perform phase occurs first, effects in that phase are in the past of the send-receive phase, whereas if the perform phase occurs second, effects in that phase are in the future of the send-receive phase.

end and increasingly uncertain interactions further away from that end. In such a case, the uncertainty of an interaction is a measure of its distance from the completely-certain end of the scale. Priorities may be viewed as an example of such a continuum. High-priority interactions always take effect preferentially over lower-priority interactions.

- *Certain*: An interaction whose outcome is predictable.
- *Uncertain*: An interaction whose outcome is unpredictable.

7.5.2.3 Combining Characteristics

Combining these characteristics yields four classes of interactions, which we name Types 0, 1, 2 and 3. We list the four classes below along with the conjunction of characteristics that defines each class. Also, we present an example interaction for each class. We depict the four classes in Figure 44.

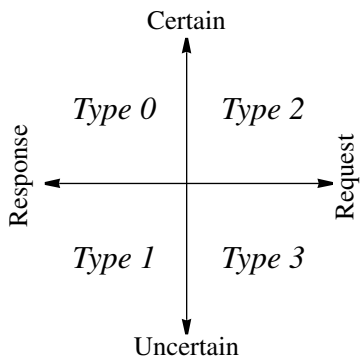


FIGURE 44: Classes of Interactions

Type 0:	Response	$\wedge$	Certain	e.g., physical events
Type 1:	Response	$\wedge$	Uncertain	e.g., updates
Type 2:	Request	$\wedge$	Certain	e.g., reads
Type 3:	Request	$\wedge$	Uncertain	e.g., orders

7.5.3 Evaluating the Taxonomy

Our taxonomy of interactions exhibits the properties of a good taxonomy discussed in §7.5.1. Our four interaction classes are mutually exclusive since no two of them possess the same conjunction of characteristics. Our taxonomy is exhaustive because the four interaction classes cover all possible combinations of the four interaction characteristics. We believe our taxonomy is unambiguous, repeatable, intuitive and useful. Our

characteristics capture semantic information about interactions. An interaction can be classified into our four classes according to *semantic* information, (i.e., its expected effect on its sender and receiver), rather than non-semantic information (e.g., its syntax, the variables it reads or writes, its size, the time taken to transmit it). We assume model designers can identify the semantics of an interaction and determine its characteristics subsequently. Determining the class of an interaction from its characteristics is unambiguous and repeatable. Our classes are logical combinations of orthogonal interaction characteristics. The classes are intuitive because they are derived from well-known characteristics of interactions. All of the interactions we have encountered exhibit combinations of these characteristics. Next, we will demonstrate the usefulness of our taxonomy by showing how to resolve the effects of concurrent interactions.

7.5.4 Resolving Effects of Concurrent Interactions

We show how to resolve the effects of concurrent interactions based on the two sets of characteristics of interactions defined above: response *versus* request and certain *versus* uncertain. Independent interactions are those whose concurrent occurrence is indistinguishable from their sequential occurrence. If we can determine that concurrent interactions are independent, then they may be resolved by serialization. The following properties enable designers to determine whether concurrent interactions are independent.

Property 1: *If the concurrent occurrence of interactions is indistinguishable from a sequential occurrence, the interactions are independent.*

Argument: Assume the interactions are dependent. Therefore, they are related by either cause-effect or concurrence. If they are related by cause-effect, they cannot occur concurrently, since cause precedes effect.

If they are related by concurrence, no sequential occurrence of the interactions can have the same effect as the concurrent occurrence. Since the interactions do not depend on one another by either cause-effect or concurrence, the initial assumption is false. □

Property 2: *If concurrent interactions affect disjoint sets of attributes, they are independent.*

Argument: If concurrent interactions affect disjoint sets of attributes, their effects can be applied sequentially. Therefore, the concurrent occurrence of these interactions is indistinguishable from their sequential occurrence. By Property 1, they are independent. □

If concurrent interactions affect disjoint sets of attributes, they are independent. If they do not, they *interfere*, but cannot be determined to be dependent yet. For interactions I_1 and I_2, if in terms of attributes, $I_1.affects^* \cap I_2.affects^* = \emptyset$ then I_1 and I_2 are independent, else they interfere. Figure 45 shows a number of interactions that occur during a time-step. Each interaction is shown as a labeled node in a graph. An arc between two nodes indicates that the corresponding interactions affect non-disjoint sets of attributes. For example, the arc between nodes labeled I_2 and I_3 indicates that in terms of attributes, $I_2.affects^* \cap I_3.affects^* \neq \emptyset$. The nodes that transitively affect non-disjoint sets of attributes form isolated sub-graphs. The interactions corresponding to nodes in a sub-graph are independent of the interactions corresponding to nodes in another sub-graph. For example, each of I_2, I_3 and I_4 is independent of each of I_1, I_5, I_6, I_7 and I_8. The set of interactions corresponding to nodes in a sub-graph may be serialized with respect to the set of interactions corresponding to nodes in another sub-graph. Therefore, the sets of interactions $\{I_2, I_3, I_4\}$, $\{I_1\}$ and $\{I_5, I_6, I_7, I_8\}$ can be serialized with one another.

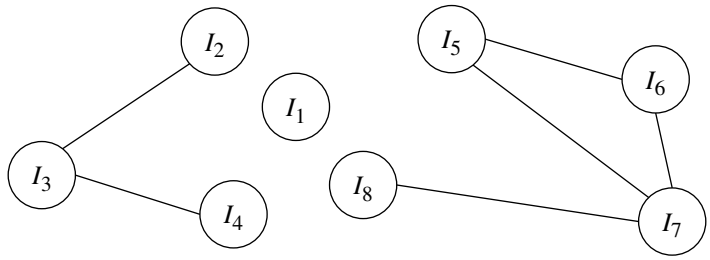


FIGURE 45: Concurrent Interactions Affecting Sets of

Property 3: *Concurrent response and request interactions are independent.*

Argument: Consider the interactions occurring during a time-step $[t_i, t_{i+1}]$ (see Figure 46). Response interactions received during this time-step refer to behaviour prior to time t_i. Request interactions received during this time-step refer to behaviour after time t_{i+1}. Let there be a time t' such that $t_i < t' < t_{i+1}$. Re-arrange the interactions such that all response interactions occur during the time-step $[t_i, t']$, and all request interactions occur during the time-step $[t', t_{i+1}]$. This re-arrangement does not alter the semantics of any interaction because all of the response interactions continue to refer to behaviour prior to time t_i and all of the request interactions continue to refer to behaviour after time t_{i+1}. All of the response interactions can occur before all of the request interactions. Therefore, the concurrent occurrence of response and request interactions is indistinguishable from a sequential occurrence, namely, responses before requests. By Property 1, responses and requests are independent. □

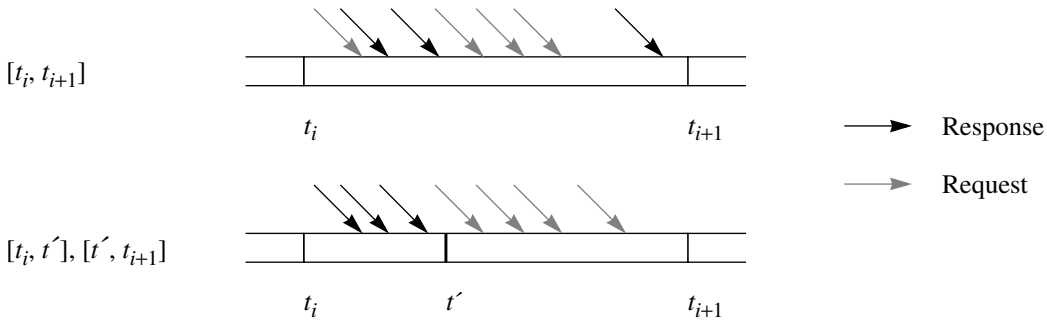


FIGURE 46: Independent Concurrent Response and Request Interactions

When two interactions interfere, but one of them has a certain outcome and the other has an uncertain outcome, then the former takes effect preferentially over the latter. Interactions with certain outcome *must* take effect, whereas interactions with uncertain outcome may be ignored, delayed or permitted to take partial effect. A partial effect for an interaction is its effect on some attributes but not others, or its fractional effect as opposed to its complete effect. If certainty or uncertainty of interaction outcomes is multi-modal (e.g., as in priorities), then interactions with higher degrees of certainty take effect preferentially over those with lower degrees of certainty.

When two interactions are resolved, either one of them takes effect preferentially over another, or they are combined. In the former case, the preferred interaction retains its type. In the latter case, the resultant interaction has the same type as the original interactions. If interactions of the same type interfere, they can be resolved by application-specific policies. For example, if two Type 0 interactions interfere, then they can be combined by a policy that reflects domain-specific laws. If they cannot be combined, then the model must be re-designed to avoid such paradoxical interactions. When concurrent interactions are combined, they may have cooperative or competitive effects. When the effect of combined interactions is "greater" than the combined effects of the individual interactions, the interactions are *cooperative*. When the effect of combined interactions is "less" than the combined effects of the individual interactions, the interactions are *competitive*. Determining "greater" and "less" is application-specific. If cooperative or competitive effects exist and the original interactions are serialized, new interactions can be added to account for these effects.

7.5.5 Policies for Resolving Effects of Interactions

In order to resolve the effects of dependent concurrent interactions, we present policies based on the characteristics of interactions, and our definitions of a model and interactions from §3.2. Designers of multi-models may choose from these policies to resolve the effects of dependent concurrent interactions. Recall that the effect of an interaction $Int(t_i)_k$ on a state of the model is the change $E(Int(t_i)_k)$, and applying the effect of that interaction on the representation $Rep(t_i)$ is equivalent to computing a function $F(Rep(t_i), E(Int(t_i)_k))$[**]. Applying the resolved effects of all of the interactions in one time-step results in the state of the model at the next time-step.

$$\begin{aligned} Int &= (Int(t_0), Int(t_1), Int(t_2), \ldots) \\ Int(t_i) &= (Int(t_i)_0 \bullet Int(t_i)_1 \bullet \ldots \bullet Int(t_i)_{n_i}) \\ Rep(t_{i+1}) &= F(Rep(t_i), E(Int(t_i))) \end{aligned}$$

Serializing: If interactions are independent, their concurrent effects are indistinguishable from their sequential effects (Property 1). Independent interactions may be serialized in an arbitrary order, and permitted to take effect one after another. The combined effect of concurrent interactions I and J is $E(I \bullet J)$. If I and J are independent, their serialized effects are $E(I) \lozenge E(J)$. If the effects of I are applied before the effects of J, we denote the combined effects as $E(I), E(J)$. The effect of I and J on the representation $Rep(t_i)$ can be applied by computing $F(Rep(t_i), (E(I), E(J)))$. Applying their serialized effects is equivalent to computing the function F recursively: $F(F(Rep(t_i), E(I)), E(J))$.

$$Rep(t_{i+1}) = F(F(F(F(Rep(t_i), E(Int(t_i)_0)), E(Int(t_i)_1)), \ldots), E(Int(t_i)_{n_i}))$$

[**] We will use sets and individual elements of a set of interactions interchangeably as parameters for F and E in order to avoid digressing into more formalisms. Distinguishing the "overloaded" uses of F and E will be clear from context.

$$E(Int(t_i)_0 \bullet Int(t_i)_1 \bullet \ldots \bullet Int(t_i)_{n_i}) = E(Int(t_i)_0) \lozenge E(Int(t_i)_1) \lozenge \ldots \lozenge E(Int(t_i)_{n_i})$$

$$\therefore E(Int(t_i)_0 \bullet Int(t_i)_1 \bullet \ldots \bullet Int(t_i)_{n_i}) = E(Int(t_i)_0), E(Int(t_i)_1), \ldots, E(Int(t_i)_{n_i})$$

Since no ordering is implied for the interactions within a time-step, the interactions may be ordered arbitrarily. If the representation at time t_{i+1}, $Rep(t_{i+1})$, is the same no matter how the interactions are ordered, then the interactions are commutative, i.e., the order in which their effects are applied does not change their combined effects.

$$E(Int(t_i)_0 \bullet Int(t_i)_1 \bullet \ldots \bullet Int(t_i)_{n_i}) = E(Int(t_i)_{n_i}), E(Int(t_i)_1), E(Int(t_i)_0), \ldots$$

$$Rep(t_{i+1}) = F(F(F(F(Rep(t_i), E(Int(t_i)_{n_i})), E(Int(t_i)_1)), E(Int(t_i)_0)), \ldots)$$

Response interactions are independent of request interactions because they are temporally disjoint. Accordingly, if the first k interactions in the set $Int(t_i)$ are responses and all of the remaining interactions are requests, then they can be resolved as below:

$$E(Int(t_i)) = (E(Int(t_i)_0) \lozenge \ldots \lozenge E(Int(t_i)_{k-1})), (E(Int(t_i)_k) \lozenge \ldots \lozenge E(Int(t_i)_{n_i}))$$

$$Rep(t_{i+1}) = F(F(Rep(t_i), E(Int(t_i)_0) \lozenge \ldots \lozenge E(Int(t_i)_{k-1})), E(Int(t_i)_k) \lozenge \ldots \lozenge E(Int(t_i)_{n_i}))$$

Ignoring: The effects of some sets of dependent concurrent interactions can be resolved meaningfully by ignoring some of them. For example, if uncertain interactions interfere with certain interactions, the former may be ignored. If the interactions in $Int(t_i)$ are sorted such that the first k interactions take effect while the rest are ignored, then:

$$E(Int(t_i)_0 \bullet Int(t_i)_1 \bullet \ldots \bullet Int(t_i)_{n_i}) = E(Int(t_i)_0 \bullet Int(t_i)_1 \bullet \ldots \bullet Int(t_i)_{k-1})$$

$$Rep(t_{i+1}) = F(Rep(t_i), E(Int(t_i)_0 \bullet Int(t_i)_1 \bullet \ldots \bullet Int(t_i)_{k-1}))$$

Delaying: The effects of some sets of dependent concurrent interactions can be resolved meaningfully by delaying some of them. For example, uncertain request interactions may be delayed if the receiver cannot resolve their effects within the current time-step. If the interactions in $Int(t_i)$ are sorted such that the first k interactions take effect during the time-step $[t_i, t_{i+1}]$, while the rest are delayed to a future time-step $[t_j, t_{j+1}]$, then:

$$Rep(t_{i+1}) = F(Rep(t_i), E(Int(t_i)_0 \bullet Int(t_i)_1 \bullet \ldots \bullet Int(t_i)_{k-1}))$$

$$Rep(t_{j+1}) = F(Rep(t_j), E(Int(t_j) \bullet Int(t_i)_k \bullet Int(t_i)_{k+1} \bullet \ldots \bullet Int(t_i)_{n_i}))$$

Combining Cooperatively or Competitively: Resolving the effects of some dependent concurrent interactions may result in enhancing or diminishing the effects of the individual interactions. If the effects are enhanced, the interactions have cooperative effects, whereas if the effects are diminished, the interactions have competitive effects. The effects of such interactions may be resolved by applying the effects of the individual interactions as well as compensatory interactions that account for the cooperative or competitive effects. Let two interactions in $Int(t_i)$ have cooperative or competitive effects. Let the compensatory interaction be denoted by $Int(t_i)_{0,\,1}$. The effect of $Int(t_i)$ is:

$$E(Int(t_i)_0 \bullet Int(t_i)_1) = E(Int(t_i)_0) \lozenge E(Int(t_i)_1) \lozenge E(Int(t_i)_{0,\,1})$$

7.6 Constructing an Interaction Resolver

An Interaction Resolver (IR) resolves the effects of concurrent interactions received by an MRE. This process involves determining the class of each interaction, determining if interactions of the same type interfere, propagating the effects of interactions and resolving the effects on each attribute using application-specific policies. The IR may be a single component or a number of components distributed over the attributes in an ADG for the MRE. Conceptually, the distinction is unimportant; during implementation, the distributed view may be more efficient.

7.6.1 Operation of an IR

The operation of an IR involves implementing policies for resolving the effects of classes or types of dependent concurrent interactions.

At design time, a designer encodes the type or characteristics of each interaction. Encoding the type or characteristics enables an IR to classify interactions. Also, at design time, the designer encodes policies in the IR for resolving types of concurrent interactions. For example, if Type 1 and Type 0 interactions interfere, the former can be discarded. The designer must specify a policy for discarding the Type 1 interactions. Examples of such a policy are ignoring or delaying the interactions (see §7.5.5). If choice of policies varies during run-time, the designer must specify conditions under which a policy is chosen.

At run-time, an MRE sends and receives concurrent interactions during a time-step. The IR groups the interactions according to their type. Initially, the IR determines the effect of each interaction on all attributes assuming that the interaction occurs in isolation. The semantics of an interaction I determine how $I.affects$ is constructed. The ADG and mapping functions determine how $I.affects^{+}$ is constructed. The effects are not applied immediately to the attributes since interfering effects have not been resolved yet. For each attribute, a list of potential changes caused by the concurrent interactions is constructed. Not all of these changes will be applied to the attribute. The IR resolves changes caused by interactions by considering the type of interactions and policies that eliminate conflicts among types of interactions. The IR considers the changes to each attribute in the order: Type 0, 1, 2 and 3 to preserve dependencies among the corresponding interactions.

The first group of interactions the IR considers is the Type 0 group. Type 0 interactions are communications about events that have already occurred. Their effects on the receiver are certain and can be computed in accordance with model requirements. If two or more Type 0 interactions interfere, then their effects can be combined. The IR permits each Type 0 interaction to take effect.

The next group of interactions the IR considers is the Type 1 group. Type 1 interactions are communications about events that may have occurred. Their effects on the receiver are uncertain. Type 1 interactions may interfere with one another as well as with Type 0 interactions. Let the tuple $<a, \delta a>$ denote an attribute and a change to it caused by a Type 1 interaction I. The IR determines whether δa conflicts semantically with any Type 0 change. If it does, the IR marks $<a, \delta a>$ as discarded. If I is discarded entirely, the IR marks all tuples in $I.affects^*$ as discarded. Thus, the interaction can take effect entirely or not at all. If I may have partial effects, then not all of the tuples in $affects^*$ need be discarded. When the only Type 1 changes remaining are the ones that do not conflict with the Type 0 changes, the Type 1 changes are checked for conflicts among themselves. If there are conflicts, the IR selects a set of non-conflicting interactions among them based on appropriate policies.

Next, the IR considers interactions in the Type 2 group. Type 2 interactions are communications about events that will occur. Type 2 interactions may be reads of attribute values, in which case, they do not interfere with any other interactions and can take effect immediately. Some Type 2 interactions may not be just reads. For example, a particular Type 2 interaction may read an attribute and have the side-effect of writing to another attribute, such as a counter. As another example, a Type 2 interaction may be a communication about an event that is certain to happen, such as a collision between two entities within the current time-step. Although Type 2 interactions occur during the same time-step as Type 0 or Type 1 interactions, Type 2 interactions are serialized with respect to Type 0 and Type 1 interactions. If Type 2 interactions interfere with one another, they can be combined in the same manner as Type 0 interactions.

Finally, the IR considers interactions in the Type 3 group. Type 3 interactions are communications about events that may occur. Type 3 interactions may be requests, orders or commands that may not be satisfied. Type 3 can be serialized with respect to Type 0 and Type 1 interactions. Type 3 interactions are resolved with respect to Type 2 interactions in the same way as Type 1 interactions are resolved with respect to Type 0 interactions. Although the actual policies may differ, Type 1 and Type 3 interactions may be discarded in favour of interactions in the other two classes.

In Figure 47, we present an algorithm for an IR. The IR determines the effects of all concurrent interactions by referring to policies encoded by the designer. The fourth step in the algorithm refers to an algorithm similar to the one we presented in Figure 34 in which we applied the effects of interactions as soon as they were determined. In Figure 47, we apply the effects of interactions after all dependent interactions have been resolved.

```
For each time-step
   List L = sort interactions by type
   For each interaction I in L
      Determine effects of I on each attribute in ADG
   For each attribute a in ADG
      If cooperative/competitive effects exist
         Insert compensatory effects in L
      If Type 0 and Type 1 interactions interfere
         Discard Type 1 changes
      If Type 2 and Type 3 interactions interfere
         Discard Type 3 changes
   For each attribute a in ADG
      Apply remaining changes
```

FIGURE 47: Algorithm for Resolving Interactions

7.6.2 An Example IR

We demonstrate the operation of an IR with the example MRE described in Chapter 6. Let the interactions in Table 9 be received concurrently by the MRE. The class of each interaction is listed in the column headed "Type". The sets *affects* and *affects*$^+$ have been shown in the last two columns. The semantics of the various interactions are as below:

- Move_Tank$_1$: Tank$_1$ moves in the current time-step.
- Move_Platoon: Platoon moves in the current time-step.
- Collide_Tank$_2$: Tank$_2$ suffers a collision in the current time-step.
- See_Tank$_1$: An entity requests the values of some attributes of Tank$_1$.
- Refill_Tank$_1$: Tank$_1$ is refuelled and repaired in the current time-step.
- Fire_Platoon: Platoon fires in the current time-step.
- Detonation: Platoon is in the path of a detonation in the current time-step.

TABLE 9: Example Concurrent Interactions

Interaction	Type	*affects*	*affects*$^+$
Move_Tank$_1$	3	Pos$_1$	Pos, Pos$_2$, Form, App, Dam$_1$, Dam$_2$
Move_Platoon	3	Pos	Pos$_1$, Pos$_2$, Form, App, Dam$_1$, Dam$_2$
Collide_Tank$_2$	0	Vel$_2$, Pos$_2$	Vel, Pos, Form, Vel$_1$, Pos$_1$, App, Dam$_1$, Dam$_2$
See_Tank$_1$	2	$\varnothing$	$\varnothing$
Refill_Tank$_1$	1	Ammo$_1$, Fuel$_1$	Fire, Ammo$_2$
Fire_Platoon	3	Fire	Ammo$_1$, Ammo$_2$
Detonation	0	App	Dam$_1$, Dam$_2$

At design-time, a designer encodes the type of each interaction and policies for resolving types of concurrent interactions. The encoded types of the interactions appear in Table 9. Suppose the encoded policies are:

L1: If Move_Platoon occurs concurrently with $Move_Tank_1$ or $Move_Tank_2$, then Move_Platoon takes effect preferentially.

L2: If Detonation occurs concurrently with $Collide_Tank_1$ or $Collide_Tank_2$, the interactions have competitive effects.

L3: If a change caused by an interaction is discarded, the interaction is discarded entirely, i.e., no partial effects of interactions are permitted.

At run-time, the IR resolves the effects of concurrent instances of the interactions in Table 9. Accordingly, the IR constructs a table similar to Table 10 for these interactions. The first column lists the name of the attribute. The second column lists the interactions affecting that attribute. The rows for which the second column reads "competitive" refer to a compensatory interaction added by the IR to enforce L2. The third column lists the type of each interaction. The fourth column lists changes to that attribute caused by a corresponding interaction. These changes are computed by permitting each interaction to take effect in isolation initially, determining the changes to attributes caused directly by the interaction, traversing the ADG and invoking the appropriate mapping functions to determine the changes to attributes caused indirectly by the interaction. In Chapter 6 we explained a similar procedure in detail for singly-occurring interactions.

TABLE 10: Effects of Concurrent Interactions

Attribute	Interaction	Type	Change
Pos	$Collide_Tank_2$	0	δP^1
	Move_Platoon	3	δP^2
	$Move_Tank_1$	3	δP^3
Pos_1	$Collide_Tank_2$	0	$\delta P_1{}^1$
	$Move_Tank_1$	3	$\delta P_1{}^2$
	Move_Platoon	3	$\delta P_1{}^3$

TABLE 10: Effects of Concurrent Interactions

Attribute	Interaction	Type	Change
Pos_2	$Collide_Tank_2$	0	$\delta P_2{}^1$
	$Move_Tank_1$	3	$\delta P_2{}^2$
	Move_Platoon	3	$\delta P_2{}^3$
Vel	$Collide_Tank_2$	0	δV^1
Vel_1	$Collide_Tank_2$	0	$\delta V_1{}^1$
Vel_2	$Collide_Tank_2$	0	$\delta V_2{}^1$
Form	$Collide_Tank_2$	0	δF^1
	Move_Platoon	3	δF^2
	$Move_Tank_1$	3	δF^3
App	Detonation	0	δA^1
	$Collide_Tank_2$	0	δA^2
	competitive	0	δA^3
	Move_Platoon	3	δA^4
	$Move_Tank_1$	3	δA^5
Dam_1	Detonation	0	$\delta D_1{}^1$
	$Collide_Tank_2$	0	$\delta D_1{}^2$
	competitive	0	$\delta D_1{}^3$
	Move_Platoon	3	$\delta D_1{}^4$
	$Move_Tank_1$	3	$\delta D_1{}^5$
Dam_2	Detonation	0	$\delta D_2{}^1$
	$Collide_Tank_2$	0	$\delta D_2{}^2$
	competitive	0	$\delta D_2{}^3$
	Move_Platoon	3	$\delta D_2{}^4$
	$Move_Tank_1$	3	$\delta D_2{}^5$
Fire	$Refill_Tank_1$	1	δR^1
	Fire_Platoon	3	δR^2
$Ammo_1$	$Refill_Tank_1$	1	$\delta A_1{}^1$
	Fire_Platoon	3	$\delta A_1{}^2$

TABLE 10: Effects of Concurrent Interactions

Attribute	Interaction	Type	Change
$Ammo_2$	$Refill_Tank_1$	1	$\delta A_2{}^1$
	Fire_Platoon	3	$\delta A_2{}^2$
$Fuel_1$	$Refill_Tank_1$	1	δU^1

For each attribute, the IR resolves the changes caused by different interactions. The order in which attributes are chosen is unimportant. Interactions that do not change any attributes, i.e., whose $affects^* = \emptyset$, cannot cause any inconsistencies among the multiple representations. If such interactions are reads, the values returned may be the values of attributes before any changes are applied or after all changes have been applied. We show how to resolve concurrent changes for all of the attributes in our example.

- Pos: The concurrent changes are δP^1, δP^2 and δP^3. δP^1 is a Type 0 change and can be applied. δP^2 and δP^3 are Type 3 changes that conflict with each other, but are independent of δP^1 which is a Type 0 change. By L1, δP^2 is applied and δP^3 is discarded. By L3, $Move_Tank_1$ is discarded entirely, and the IR discards $\delta P_1{}^2$, $\delta P_2{}^2$, δF^3, δA^5, $\delta D_1{}^5$ and $\delta D_2{}^5$ — the changes caused by this interaction to each attribute in $Move_Tank_1.affects^*$.
- Pos_1: The concurrent changes remaining are $\delta P_1{}^1$ and $\delta P_1{}^3$. $\delta P_1{}^1$ can be applied since it is a Type 0 change. In practice, we expect $\delta P_1{}^1 = 0$ since a collision involving $Tank_2$ will not affect $Tank_1$. However, this is an artifact of the particular interactions we have chosen, hence it does not factor into the decision about which changes are applied. $\delta P_1{}^3$ does not conflict with $\delta P_1{}^1$ because of the types of these changes. The IR has discarded $\delta P_1{}^2$ already.
- Pos_2: The concurrent changes remaining are $\delta P_2{}^1$ and $\delta P_2{}^3$. $\delta P_2{}^1$ can be applied

since it is a Type 0 change. $\delta P_2{}^3$ can be applied since it does not conflict with $\delta P_2{}^1$ because of the types of these changes. $\delta P_2{}^2$ has been discarded already.

- Vel: δV^1 can be applied.
- Vel_1: $\delta V_1{}^1$ can be applied.
- Vel_2: $\delta V_2{}^1$ can be applied.
- Form: Both the remaining changes, δF^1 and δF^3, can be applied.
- App: δA^1, δA^2 and δA^3 can be applied because they are Type 0. δA^3 is a competitive change caused by the compensatory interaction added by the IR. Since δA^3 is a compensation for two Type 0 interactions, it is also Type 0. After the Type 0 interactions are applied, the Type 3 changes are applied. Since δA^5 has been discarded, only δA^4 can be applied.
- Dam_1: $\delta D_1{}^1$, $\delta D_1{}^2$ and $\delta D_1{}^3$ can be applied because they are Type 0. $\delta D_1{}^4$ is applied subsequently.
- Dam_2: $\delta D_2{}^1$, $\delta D_2{}^2$ and $\delta D_2{}^3$ can be applied because they are Type 0. $\delta D_2{}^4$ is applied subsequently.
- Fire: The potential changes are δR^1 and δR^2. δR^1 is a Type 1 change. Since there are no previously-applied changes, it can be applied. δR^2 can be applied as well since Type 3 interactions do not conflict with Type 1 interactions.
- $Ammo_1$: $\delta A_1{}^1$ and $\delta A_1{}^2$ can be applied.
- $Ammo_2$: $\delta A_2{}^1$ and $\delta A_2{}^2$ can be applied.
- Fuel: δU^1 can be applied.

When all of these changes have been applied, the MRE will be consistent. The IR enforces policies L1, L2 and L3 specified for this application. Since the specified policies

for dependent concurrent interactions do not isolate the interactions, the effects of these interactions can be resolved in a manner meaningful to the application. Consequently, the MRE interacts at multiple representation levels concurrently and consistently.

7.7 Chapter Summary

Concurrent interactions may have effects that are dependent on one another. Resolving the effects of such interactions by serializing them is incorrect since serialization isolates the interactions. Dependent concurrent interactions can be resolved efficiently by classifying them and formulating policies for resolving classes of interactions. We present four characteristics of interactions — request, response, certain and uncertain — and four classes of interactions based on combinations of these characteristics — Types 0, 1, 2 and 3. The classes distinguish semantic types of interactions encountered in models. Based on these classes of interactions, we presented policies for resolving the effects of their concurrent occurrence. We showed how to construct an Interaction Resolver (IR) for an MRE. An IR resolves the effects of types of interactions at run-time. By designing a Consistency Enforcer and an Interaction Resolver, a designer can ensure that an MRE interacts at multiple representation levels concurrently. Next, we present a process for applying our framework, `UNIFY`, to jointly-executing models.

Woe to the author who always wants to teach;
The secret of being a bore is to tell everything.
— *Voltaire,* De la Nature de l'Homme

Chapter 8

Applying UNIFY: A Process

In this chapter, we present guidelines and a process for applying our techniques to achieve effective MRM. Designers can apply UNIFY by reading this chapter first and referring to preceding chapters when necessary. We presented Multiple Representation Entities (MREs) as a means of maintaining concurrent representations (Chapter 5). A Consistency Enforcer (CE) maintains internal consistency within an MRE (Chapter 6). An Interaction Resolver (IR) resolves the effects of dependent concurrent interactions on an MRE (Chapter 7). Here, we present a process for applying the techniques in UNIFY. By following these steps, designers can achieve effective MRM in their applications:

1. Construct an MRE from the representations of jointly-executing models.
2. Capture dependencies among the attributes with an ADG.
3. Select mapping functions for each dependency.
4. Classify interactions according to a taxonomy.
5. Select policies for resolving the effects of concurrent interactions.
6. Construct a CE and an IR for the MRE.

We expect designers to construct solutions for their MRM applications based on general guidelines. In §8.1, we justify each guideline briefly. Since UNIFY is intended to aid designers of multi-models, in §8.2 we show how UNIFY can be used in conjunction with a familiar modelling methodology. In §8.3, we explain how to apply the techniques in UNIFY with the example application employed in previous chapters.

8.1 Guidelines for MRM Designers

We present guidelines for achieving effective MRM using UNIFY. We justify each guideline briefly and refer to earlier sections in this dissertation for detailed explanations. We assume that a designer desires to construct a multi-model from models that meet their users' requirements. For each model, the designer must identify the representation of entities, relationships among attributes and interactions that change the state of entities. We assume the designer can identify the cross-model relationships in the multi-model, can understand the intertwined semantics of interactions and can make time-steps in the multi-model compatible.

G1: Represent entities at levels at which they can interact.
This guideline arises from FO-1 in §4.2. For effective MRM, entities should interact at a representation level at which their semantics are compatible (see Figure 9 in Chapter 4).

G2: Maintain concurrent representations for jointly-executing models.
Maintaining concurrent representations means preserving them at all times and permitting interactions to change them. MREs maintain concurrent representations (see Chapter 5). Designing MREs can ensure that entities interact at levels at which their semantics are compatible.

G3: Make the time-steps of the multiple models compatible.

If jointly-executing models have compatible time-steps, neither violates any assumptions made by another during any time-step. Achieving compatible time-steps may involve executing some models at finer or coarser time-steps (see §3.3.3). Accordingly, the attributes in the models may be augmented with tolerance values, which determine acceptable variances in the values of the attributes at overlapping simulation times (see §4.2.4).

G4: Capture cross-model relationships.

Capturing relationships among representations involves determining the semantics of attributes that are part of the representations. Attributes with overlapping semantics are likely to be related to one another. Relationships among models can be captured by Attribute Dependency Graphs and mapping functions (see Chapter 6).

G5: Propagate the effects of an interaction to all representation levels.

An interaction affects the attributes at its own representation level as well as related attributes at other representation levels (see FO-2 in §4.2.2). Propagating the effects of interactions to all relevant attributes ensures that multiple representations are consistent.

G6: Select mapping functions for each relationship between representations.

These functions translate value spaces or changes in values among related attributes. Mapping functions must satisfy the properties time-bounded completion, composability and reversibility (see §6.2).

G7: Identify semantics characteristics of interactions.

In §7.5, we presented a taxonomy of interactions, consisting of four classes, in order to reduce the complexity of resolving concurrent interactions. Alternative taxonomies are possible. Classifying an interaction involves understanding its semantics, i.e., its effects on its sender and receiver.

G8: Select policies for resolving the effects of dependent concurrent interactions.

The effects of concurrent interactions may depend on one another (see FO-3, §4.2.3, §7.3). In §7.5.4, we presented example policies for resolving the effects

of dependent concurrent interactions. Specifying policies for resolving interactions involves capturing the semantics of their concurrent occurrence.

By following these guidelines, designers can incorporate effective MRM into their applications. A multi-model can satisfy its users' requirements if MRM is effective.

8.2 Using UNIFY with a Specification Methodology

We expect designers of multi-models to achieve effective MRM by employing UNIFY in conjunction with a specification methodology. We augment an existing specification methodology so that designers can build on familiar modelling techniques when they apply UNIFY. Specification methodologies such as OMT_R [RUM91], OOA [SHLAER92] and UML [ALHIR98] support specifying model representations and relationships, but not the effects of interactions. In contrast, OMT [OMT98] supports specifying the effects of an interaction in terms of its parameters, its sender, its receiver and the attributes it affects. Since resolving the effects of concurrent interactions is one of the hardest problems in MRM, we regard the support for interactions in OMT suitable for MRM. We augment OMT by permitting designers to specify attribute relationships, interaction types and policies for resolving concurrent interactions.

In the Department of Defense's High Level Architecture (HLA) initiative, multiple models may execute together in a "plug-and-play" fashion. Individual models and multi-models are specified using a methodology called the Object Model Template (OMT). In OMT, individual models, or federates, are specified by tables describing their interface. These tables together are called the Federate Object Model (FOM) for that federate. The FOM for a particular model has the following tables:

- Object Class Structure Table (OCST) shows the class hierarchy along with

information for whether each class is publishable (shareable with other models), subscribable (interesting to the current model) or both.

- Attribute/Parameter Table (APT) lists object attributes and interaction parameters along with their data type, cardinality, units, resolution, accuracy, accuracy condition, update type and update condition.
- Object Interaction Table (OIT) lists each interaction, its sender, its receiver, the attributes it affects and whether a model initiates, senses or reacts to it.
- Enumerated Data Table (EDT) lists the values of all enumerations.
- Complex Data Table (CDT) lists the definitions of all structured data types.
- Object Class Definitions (OCD) describes the role of each entity.
- Object Interaction Definitions (OID) describes each interaction.
- Attribute/Parameter Definitions (APD) describes each object attribute and interaction parameter.

The OCST and the APT enable a designer to construct the representations for a multi-model. The APT and the OIT enable the designer to describe the interactions for the multi-model. In OMT, the only relationships that can be determined are those of base class and derived class [STROU91]. These relationships capture neither attribute relationships nor complex entity relationships. Furthermore, although in OMT a designer can specify the effects of an interaction, the designer cannot specify effects of *concurrent* interactions.

We augment OMT with tables that permit a designer to capture relationships among attributes and specify policies for resolving the effects of concurrent interactions. The inability to express entity relationships is a serious shortcoming in OMT. The class hierarchy captured by the OCST captures static entity relationships such as inheritance,

but not the dynamic relationships among entities, for example, relationships of configuration. Therefore, we augment OMT with an Attribute Relationship Table (ART). This table lists each attribute dependency, its class and specifications for its associated mapping function. In OMT, the OIT and APT permit interactions to be specified in detail. We augment the OIT in OMT with a column for specifying the class for each interaction. Once the class for an interaction has been specified, policies for resolving the effects of concurrent interactions can be formulated. We augment OMT with a table, the Concurrent Interactions Table (CIT), which permits a designer to specify such policies. The CIT permits a designer to specify policies in terms of combinations of classes of interactions or individual interactions. Table 11 is an ART and Table 12 is a CIT for the example application developed in Chapters 6 and 7. With these additions, designers can employ OMT and `UNIFY` to incorporate MRM into their applications.

TABLE 11: Example Attribute Relationship Table

Dependency	Type	Specification
$\text{Hits}_1 \rightarrow \text{Str}$	Cumulative	Str is the weighted sum of Hits_1 and Hits_2. Changes to Str are distributed to Hits_1 and Hits_2 according to weights on the dependencies.
$\text{Hits}_2 \rightarrow \text{Str}$	Cumulative	
$\text{Str} \rightarrow \text{Hits}_1$	Distributive	
$\text{Str} \rightarrow \text{Hits}_2$	Distributive	
$\text{Ammo}_1 \rightarrow \text{Fire}$	Cumulative	Fire is the weighted sum of Ammo_1 and Ammo_2. Changes to Fire are distributed to Ammo_1 and Ammo_2 according to weights on the dependencies.
$\text{Ammo}_2 \rightarrow \text{Fire}$	Cumulative	
$\text{Fire} \rightarrow \text{Ammo}_1$	Distributive	
$\text{Fire} \rightarrow \text{Ammo}_2$	Distributive	
$\text{Pos}_1 \rightarrow \text{Pos}$	Cumulative	Pos is the centroid of Pos_1 and Pos_2.
$\text{Pos}_2 \rightarrow \text{Pos}$	Cumulative	
$\text{Pos} \rightarrow \text{Pos}_1$	Distributive	
$\text{Pos} \rightarrow \text{Pos}_2$	Distributive	
$\text{Vel} \rightarrow \text{Pos}$	Modelling	Position Pos changes with Velocity Vel according to physical laws.
$\text{Vel}_1 \rightarrow \text{Pos}_1$	Modelling	
$\text{Vel}_2 \rightarrow \text{Pos}_2$	Modelling	

TABLE 12: Example Concurrent Interactions Table

Concurrent Interactions	Condition	Policy
Move_Platoon, any combination of (Move_Tank$_1$, Move_Tank$_2$)	All received	Ignore all except Move_Platoon
Detonation, any combination of (Collide_Tank$_1$, Collide_Tank$_2$)	All received	Add compensatory interaction for competitive effects to Dam$_1$ or Dam$_2$; actual damage less than sum of damages
Type 0, Type 1	All received	Ignore Type 1
Type 2, Type 3	All received	Ignore Type 3
Any Interaction	Ignored or Delayed	Ignored or Delayed entirely, i.e., no partial effects permitted

8.3 Process for Effective MRM

UNIFY can be summarised by the process diagram in Figure 48. The unshaded boxes represent steps in the process of applying UNIFY, whereas the shaded boxes represent steps in the design of models or a multi-model. The dashed arrows represent feedback in the process. We view designing models, constructing a multi-model and achieving MRM as iterative processes. We employed a running example of a Platoon-Tanks MRE in Chapters 6 and 7 in order to explain our techniques for effective MRM. Here, we present the process of applying those techniques.

UNIFY does not address the design of individual models. However, the steps in UNIFY depend on the successful completion of steps in the design of individual models. For example, constructing an MRE requires that the designer identify the representations of jointly-executing models, Model$_A$ and Model$_B$. Conversely, constructing an MRE may provide insights into identifying representations of the models. Likewise, constructing an ADG and selecting mapping functions for an MRE requires that the designer identify the

relationships within and among jointly-executing models. In the design of a model, identifying relationships can be carried out in parallel with identifying interactions. In like fashion, in UNIFY, constructing an ADG and selecting mapping functions can be carried out in parallel with classifying interactions and selecting policies for resolving concurrent interactions. In practice, these steps may be carried out sequentially; however, their relative order is unimportant.

Verification and validation (V&V) is an important step in the design of models. V&V is undertaken to ensure that a model is effective, i.e., meets its users' requirements. V&V for a multi-model depends on V&V for constituent models as well as V&V for MRM. V&V for constituent models is outside the scope of our work. V&V for MRM involves ensuring that jointly-executing models satisfy MRM requirements: multi-representation interaction (R1), multi-representation consistency (R2) and cost-effectiveness (R3). The MRE approach satisfies R3. If R1 and R2 are not satisfied, a designer must iterate through the process of achieving MRM. In turn, the designer may have to re-examine the construction of jointly-executing models.

We list the steps in UNIFY for the Platoon-Tanks MRE from Chapters 6 and 7. The Platoon-Tanks multi-model captured the combined semantics of a Platoon model and a Tank model. We employed UNIFY in order to achieve effective joint execution of the Platoon and Tank models. The steps we undertook in the process of employing techniques in UNIFY are listed below along with the sections in which we performed each step.

1. Construct an MRE for the jointly-executing models: §6.1. The Platoon-Tanks MRE captured the concurrent representations of a Platoon and two Tanks. The MRE could interact at either or both representation levels at any time.

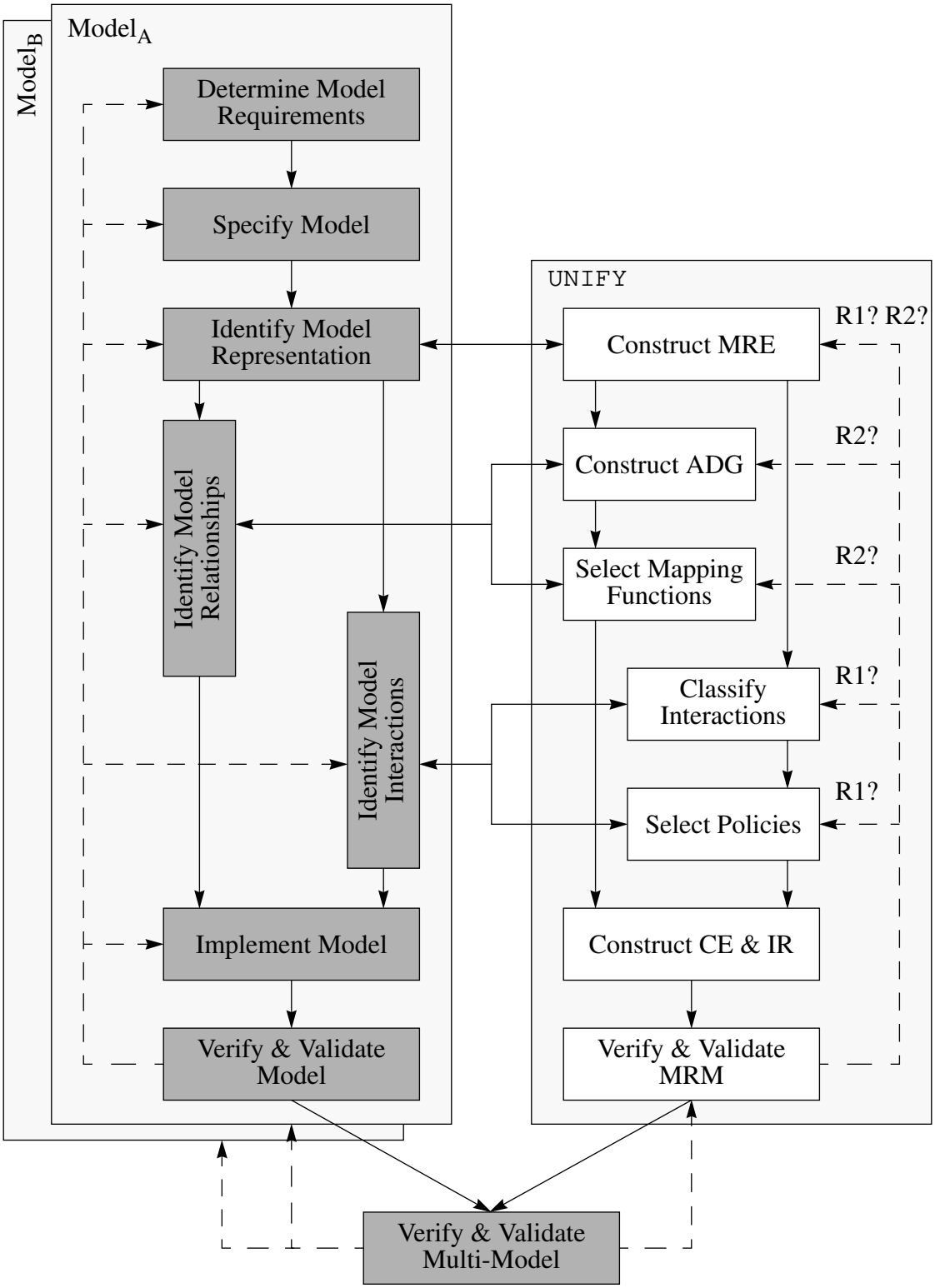


FIGURE 48: Process for Effective MRM

2. Capture dependencies among the attributes in the MRE: §6.1. An ADG captured the dependencies among Platoon and Tank representations. By classifying and weighting dependencies, we captured their static and dynamic semantics.
3. Select mapping functions for each dependency: §6.2. We selected mapping functions to translate values or changes in values among Platoon and Tank attributes. These mapping functions ensured that the Platoon-Tanks MRE was internally consistent at all observation times.
4. Classify interactions: §7.6.2. We classified the interactions in the Platoon and Tank models according to our taxonomy. This classification enabled us to capture the semantics of interactions.
5. Select policies for resolving concurrent interactions: §7.6.2. We selected policies for capturing the dependencies among concurrent interactions. These policies resolved the effects of dependent concurrent interactions.
6. Construct a CE and an IR for the MRE: §6.3 and §7.6.2. A CE consists of an ADG and application-specific mapping functions, whereas an IR consists of policies for resolving the effects of interactions. We presented processes for the operation of a CE and IR for the Platoon-Tanks MRE. A CE and IR maintain internal consistency within an MRE when concurrent interactions occur.

The above steps constitute a process for achieving effective MRM for an application. The process and the techniques employed in each step are part of `UNIFY`, our approach for effective MRM. We demonstrated how to apply `UNIFY` to a multi-model application. We present our experience in applying the above process to several models in the appendices. Next, we evaluate `UNIFY`.

The purpose of computing is insight, not numbers.
— Richard Hamming

Chapter 9

Evaluation

Our framework, UNIFY, is a sufficient and practical approach for effective MRM. UNIFY is sufficient because it satisfies three requirements for MRM: multi-representation interaction (R1), multi-representation consistency (R2) and cost-effectiveness (R3). We described these requirements in §1.3 and §3.4. UNIFY is practical because it offers techniques and processes for designing a multi-model. Designers can apply UNIFY in conjunction with a model specification methodology such as OMT to construct effective multi-models. In §9.1, we evaluate UNIFY in terms of the MRM requirements. In §9.2, we discuss briefly how UNIFY can be applied to existing applications to achieve effective MRM. In §9.3, we present limitations of our work.

9.1 Evaluating UNIFY in terms of MRM Requirements

We evaluate UNIFY with regard to our three sufficiency requirements R1, R2 and R3: multi-representation interaction, multi-representation consistency and cost-effectiveness. Since the joint execution of multiple models is intended to capture their combined

semantics, an MRM approach must permit the execution of the individual models. Therefore, the MRM approach must permit entities at all representation levels to interact. An MRM approach must maintain consistency among the representations of jointly-executing models. If the representations of jointly-executing models are consistent, the behaviours of the models can be consistent, thus leading to effectiveness of the multi-model. Consistency can be maintained among multiple representations by propagating changes from one representation to another. Lastly, an MRM approach must keep simulation and consistency costs low. We reiterate the definitions of R1, R2 and R3 here.

- **Multi-representation Interaction (R1)**: Entities in each jointly-executing model may initiate and receive interactions concurrently.
- **Multi-representation Consistency (R2)**: The multiple models must be consistent with one another, i.e., cross-model relationships must hold.
- **Cost-Effectiveness (R3)**: The total cost of simulating multiple models and maintaining consistency among them should be low.

`UNIFY` satisfies these requirements. In the following sub-sections we evaluate `UNIFY` and alternative MRM approaches such as selective viewing and aggregation-disaggregation in terms of these requirements.

9.1.1 Multi-Representation Interaction

`UNIFY` satisfies R1 by permitting interactions to occur at all representation levels. Let $Model^M$ be a multi-model constructed from low-resolution model, $Model^A$, and a high-resolution model, $Model^B$. Recall from Chapter 3 that $Model^M = \langle Rep^M, Rel^M, Int^M \rangle$.

Alternative approaches, such as selective viewing and aggregation-disaggregation, do not satisfy R1. In selective viewing, interactions at only the most detailed representation level are permitted. In other words, in selective viewing, $Int^M = Int^B$ at all times. Therefore, selective viewing does not satisfy R1. In aggregation-disaggregation, interactions at different representation levels are permitted, but at only one level at a given time. In other words, at time $t_i \in T^M$, $Int^M(t_i) = Int^A(t_i)$ but at some time $t_j \in T^M$, $t_j \neq t_i$, $Int^M(t_j) = Int^B(t_j)$. Since at any given time, interactions at only one level are permitted, aggregation-disaggregation does not satisfy R1.

In contrast with selective viewing and aggregation-disaggregation, `UNIFY` permits concurrent interactions at multiple representation levels. In `UNIFY`, $Int^M = Int^A \cup Int^B$. Since interactions at all representation levels can occur at all times, `UNIFY` satisfies R1.

9.1.2 Multi-Representation Consistency

`UNIFY` satisfies R2 by maintaining consistency among jointly-executing models. A Consistency Enforcer (CE) maintains consistency among the concurrent representations within an MRE. A CE propagates a change caused by an interaction to all dependent attributes. A CE consists of an Attribute Dependency Graph (ADG) and application-specific mapping functions. An ADG captures dependencies among attributes at different representation levels. Mapping functions translate changes to attributes before the next observation time occurs. Consequently, an MRE is always internally consistent.

In alternative MRM approaches, such as aggregation-disaggregation and selective viewing, multi-representation consistency is not satisfied because cross-model relationships do not hold at all times. For a valid model, $Model = \langle Rep, Rel, Int \rangle$, Rel

must hold at all observed times. For a multi-model, $Model^M = Model^A \cup Model^B$, $Rel^M = Rel^A \cup Rel^B \cup Rel^{cross\text{-}model}$. If the models, $Model^A$ and $Model^B$, are not related to one another, $Rel^{cross\text{-}model} = \emptyset$, i.e., cross-model relationships are null. In such a case, cross-model relationships hold at all observation times for any approach, including UNIFY. However, for typical jointly-executing models, $Rel^{cross\text{-}model} \neq \emptyset$. Selective viewing forces $Rel^{cross\text{-}model}$ to be null, since only one representation level exists. Likewise, aggregation-disaggregation forces $Rel^{cross\text{-}model}$ to be null except during transitions from one representation level to another. Forcing cross-model relationships to be null ensures that they hold trivially, but does not capture relationships among jointly-executing models at all observed times. Therefore, selective viewing and aggregation-disaggregation satisfy R2 partially only.

In UNIFY, $Rel^{cross\text{-}model}$ holds at all observation times. ADGs and mapping functions capture $Rel^{cross\text{-}model}$ completely. A CE, which consists of an ADG and mapping functions, ensures that changes to attributes of an MRE propagate to all dependent attributes before the next observation time. Consequently, no two entities can receive inconsistent views of an MRE at overlapping simulation times. Therefore, an MRE exhibits temporal consistency. Mapping functions ensure that attributes in an MRE do not change in a manner inconsistent with model requirements. As a result, the MRE exhibits mapping consistency. Since an MRE is always internally consistent, UNIFY satisfies R2.

9.1.3 Cost-Effectiveness

UNIFY satisfies R3 by reducing the total cost of executing a model. A sufficient approach to MRM must achieve multi-representation interaction and multi-representation

cost-effectively. *Simulation cost* is the cost of executing multiple models. *Consistency cost* is the cost of maintaining consistency among concurrent representations. Together, simulation and consistency costs constitute the total cost of executing a model. Simulation and consistency costs can be translated to resource consumption costs. For example, simulation cost can be translated to the amount of processing required to apply the primary effects of interactions. In other words, when an interaction occurs, the processing required to change the values of attributes affected directly by the interaction is a simulation cost. Likewise, consistency cost can be translated to the processing cost incurred in order to keep entities consistent. In other words, when an interaction occurs, the processing required to apply the secondary effects of the interaction is a consistency cost. Simulation and consistency costs tend to be trade-offs, i.e., an approach with low simulation cost tends to have high consistency cost and *vice versa*. `UNIFY` enables reducing the two costs, i.e., their sum is lower when `UNIFY` rather than aggregation-disaggregation or selective viewing is the MRM approach. `UNIFY` satisfies R3 by reducing simulation and consistency costs.

We compare simulation and consistency costs for selective viewing, aggregation-disaggregation and `UNIFY`. It is hard, if not impossible, to change the MRM approach dynamically for an application in order to measure costs fairly. Hence, we construct a synthetic application for which we can change the MRM approach. We present the assumptions we make in our cost comparison.

9.1.3.1 Assumptions

The semantics of our synthetic application are unimportant; we merely count simulation and consistency actions undertaken by the application. Each action reflects a

processing or communication operation with an associated application-specific resource cost. For a fair comparison, each approach should permit interactions at all levels. However, aggregation-disaggregation and selective viewing do not permit interactions at non-simulated levels, whereas `UNIFY` permits interactions at all levels. Accordingly, we compare `UNIFY` with hypothetical variants of aggregation-disaggregation and selective viewing that permit interactions at non-simulated levels. Comparing `UNIFY` with these variants does not bias our cost analyses because the variants have the same remaining characteristics as their corresponding original approaches.

In our hypothetical aggregation-disaggregation approach (AD), an entity is simulated at its lowest resolution or most aggregate level. As long as interactions occur at this level, the entity is represented at this level alone. However, when an interaction at a higher resolution occurs, the entity is disaggregated into sub-entities at the level of the interaction. After the effects of the interaction have been applied to the appropriate sub-entity, all sub-entities are aggregated back to the lowest resolution. AD can be improved; partial disaggregation and pseudo-disaggregation are improvements over AD (see §2.2.2). However, as it stands, AD captures the essence of the aggregation-disaggregation approach. AD has low simulation cost since only a few entities are simulated.

In our hypothetical selective viewing approach (SV), an entity is simulated at the highest resolution level. The entity is disaggregated initially into its sub-entities at the highest resolution. Each sub-entity exists throughout the duration of the simulation. When lower-resolution interactions occur, they are translated into their highest-resolution equivalents. If a low-level interaction affects a single low-resolution entity, we translate

the interaction to many high-resolution interactions that affect a corresponding number of high-resolution entities. SV has low consistency cost since only one level is simulated.

In `UNIFY`, an MRE is constructed for an entity at multiple resolution levels. In our synthetic application, we maintain attributes at all resolution levels at all times. The effects of an interaction are applied at the appropriate resolution level and propagated to other resolution levels. Computing simulation costs for an MRE simulated at all resolution levels would bias our analysis against `UNIFY` unfairly. An MRE simulated at all levels permits *concurrent* interactions at different levels, which none of AD, SV, aggregation-disaggregation and selective viewing permit. Therefore, for our analyses, we simulate an MRE at any one of its levels at a given time. Simulating the lowest resolution level would incur low simulation cost. However, we choose the simulated level uniform-randomly to reflect the capability of an MRE to be simulated at any level.

The model for our synthetic application consists of one entity (shown in Figure 49) represented at multiple resolution levels. The entity may interact at any level. In order to satisfy R2, the representations of the entity at all resolution levels must be consistent with one another. We make some assumptions about our model for our analyses:

- There are L resolution levels, level 0 being the lowest (most aggregate) and level $L-1$ being the highest (most disaggregate).
- A sub-entity at a resolution level j consists of N identical sub-entities at level $j+1$ if $0 \leq j < L-1$, and zero sub-entities if $j = L-1$. We refer to N as the fan-out.
- All sub-entities at all levels have exactly a attributes. All of the attributes of a sub-entity at a particular level are modified by every interaction at that level.
- Interactions may occur at any resolution level.

- All interactions are independent of one another. Therefore, concurrent interactions are serialized.
- An entity executes *progress* interactions to advance in the simulation. These interactions do not change attributes, but involve processing on the part of the entity. An entity receives R interactions before receiving a progress interaction.

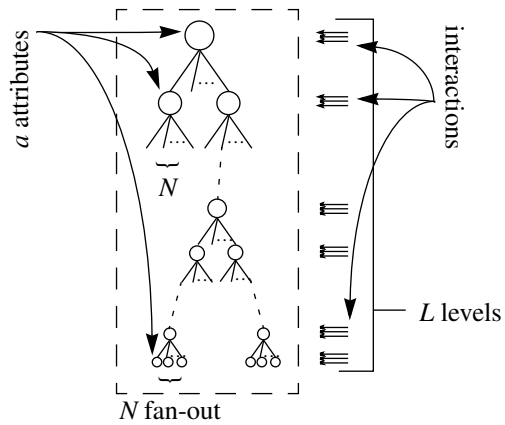


FIGURE 49: Entity in Synthetic Application

We define Ψ as a function on X and Y such that: $\Psi(X, Y) = \sum_{i=0}^{Y-1} X^i = \frac{X^Y - 1}{X - 1}$. If an entity is represented at L resolution levels with a fan-out of N, it has $\Psi(N, L)$ sub-entities. In AD, an entity may be disaggregated down to level L–1, thus requiring $O(\Psi(N, L))$ memory. In SV, only level L–1 sub-entities are simulated, thus requiring $O(N^{L-1})$. In `UNIFY`, all sub-entities at all levels are present, thus requiring $O(\Psi(N, L))$ memory. The memory consumption for all three approaches is of the order $O(N^{L-1})$.

9.1.3.2 Consistency Cost

Consistency Cost (CC) reflects the number of actions required to maintain consistency when interactions at different resolution levels occur.

Aggregation-disaggregation: In AD, an entity is always simulated at level 0. If an entity receives an interaction at level r ($0 < r < L$), the entity disaggregates to level r,

applies the effects of the interaction at level r and re-aggregates to level 0. Aggregation and disaggregation maintains consistency among the multiple representations because the effects of an interaction propagate to attributes at the simulated level. In order to disaggregate to level r from the current level 0, or aggregate from level 0 to level r, the costs incurred are $O(a \times \Psi(N, r))$. Thus, CC_{AD} (Figure 50) = $O(2a \times \Psi(N, r))$.

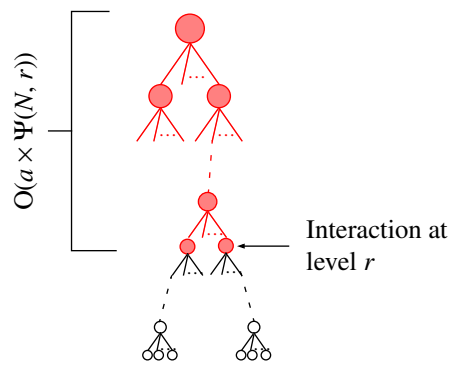


FIGURE 50: AD — Consistency Cost

Selective Viewing: In SV, an entity is always simulated at level L–1. There exists only one level of resolution, namely, the highest. Consistency is maintained only within one level. All interactions occur at level L–1, where $L = 1$. Therefore, CC_{SV} (Figure 51) = $O(a)$.

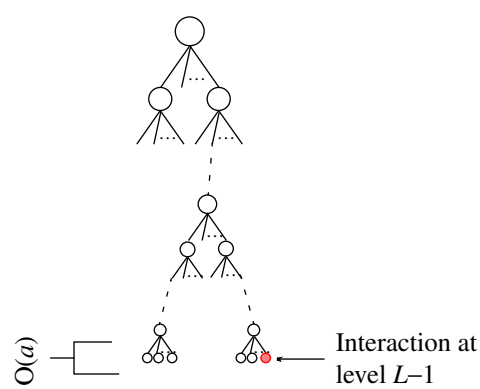


FIGURE 51: SV — Consistency Cost

UNIFY: In an MRE, an entity is represented consistently at all levels of resolution. If an interaction occurs at level r ($0 \leq r < L$), the entity applies the effects of the interaction at

level r and propagates the effects to all other levels. In order to propagate the effects to higher resolution levels, the cost incurred is $O(a \times \Psi(N, L-r))$. The cost incurred in propagating the effects to lower resolution levels is $O(ra)$. Thus, $CC_{\texttt{UNIFY}}$ (Figure 52) $= O(ra + a \times \Psi(N, L-r))$.

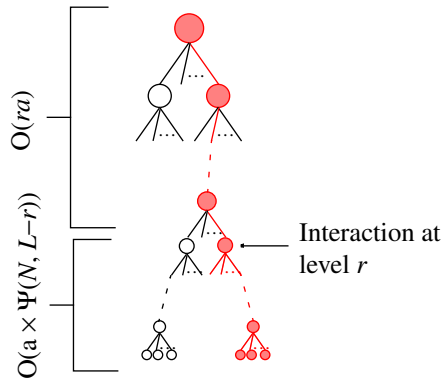


FIGURE 52: UNIFY — Consistency Cost

9.1.3.3 Simulation cost

Simulation Cost (SC) reflects the number of actions required to simulate an entity. In AD, an entity is simulated at level 0. Therefore, $SC_{AD} = O(a)$. In SV, an entity is simulated at level $L-1$. Therefore, $SC_{SV} = O(a \times N^{L-1})$. In UNIFY, at a given time, an entity is simulated at one of the multiple levels. If the entity is simulated at level r $(0 \leq r < L)$, $SC_{\texttt{UNIFY}} = O(a \times N^{r})$. Figure 53 shows SC for AD, SV and UNIFY.

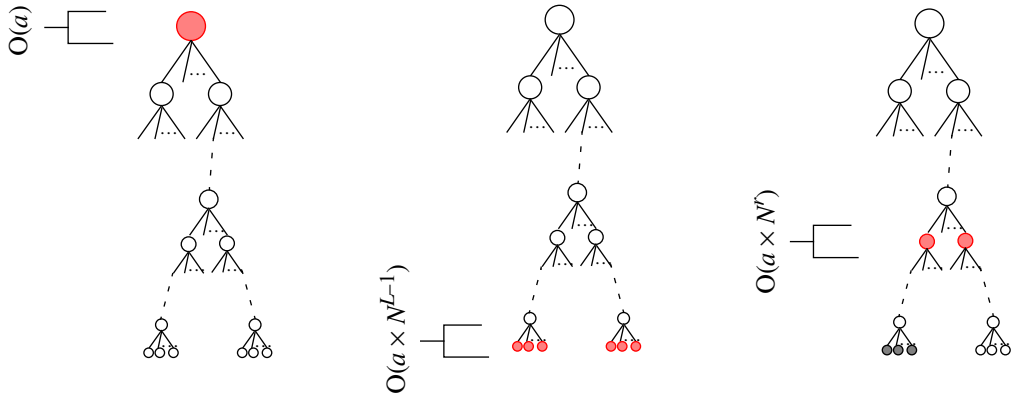


FIGURE 53: (Left to Right) AD, SV and UNIFY — Simulation Cost

9.1.3.4 Expected Costs

Table 13 compares the expected costs for the different approaches. Figure 54 shows a rough diagram of expected simulation and consistency costs for AD, SV and `UNIFY`.

TABLE 13: Cost Comparison among MRM approaches

	CC	SC
AD	$O(2a \times \Psi(N, r))$	$O(a)$
SV	$O(a)$	$O(aN^{L-1})$
`UNIFY`	$O(ra + a\Psi(N, L-r))$	$O(a \times N^r)$

As Figure 54 shows, simulation and consistency costs are trade-offs. Consistency costs decrease with approaches that execute more in the disaggregate. However, simulation costs increase. An approach executing mostly in the aggregate has low simulation costs, but high consistency costs.

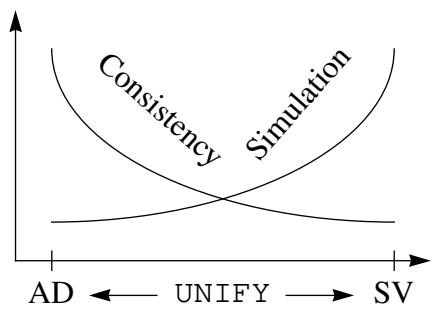


FIGURE 54: Expected Costs

`UNIFY` lies between extremes of multi-resolution approaches, i.e.,

$$SC_{AD} \leq SC_{\texttt{UNIFY}} \leq SC_{SV}$$

$$CC_{AD} \geq CC_{\texttt{UNIFY}} \geq CC_{SV}.$$

Therefore, `UNIFY` enables reducing the sum of simulation and consistency costs.

9.1.3.5 Experimental Costs

We constructed a simulation to measure and compare SC and CC for AD, SV and `UNIFY`. The simulation confirmed our predictions about how the costs grow as factors

such as number of levels and fan-out grow. Also, the simulation confirmed our expectation that the total of simulation and consistency costs can be reduced in `UNIFY`.

All costs were measured in terms of the number of actions. SC was the total number of actions to execute a progress interaction (SC^P) and apply the primary effects of an interaction (SC^I), i.e., $SC = SC^P + SC^I$. SC^P_{AD} and SC^I_{AD} were one per interaction. For each interaction, SC^P_{SV} was equal to the total number of entities at the highest resolution, and SC^I_{SV} was equal to the number of sub-entities affected by the interaction (after translating a low-resolution interaction into high-resolution interactions). For each interaction, $SC^P_{\texttt{UNIFY}}$ was equal to the number of entities at a level chosen uniform-randomly when a progress interaction occurred, and $SC^I_{\texttt{UNIFY}}$ was one. CC_{AD} was the number of times sub-entities were created and destroyed per interaction. CC_{SV} was the number of sub-entities created and destroyed initially. $CC_{\texttt{UNIFY}}$ was the number of actions required to propagate the effects of each interaction to all sub-entities and to each parent.

We measured costs by varying four independent parameters one at a time:

- T: total number of interactions during the simulation. T = 10, 100, …, 100000.
- R: number of interactions between progress interactions. R = 1, 2, 3, …, 10.
- F: fan-out, or the number of sub-entities per entity. F = 1, 2, 3, …, 10.
- L: number of levels. L = 1, 2, 3, …, 10.

The canonical case was L = 3, F = 2, T = 1000, R = 5. The graphs that follow should be interpreted for trends rather than actual numbers. The relationship between costs of simulation and consistency and the above parameters are as follows:

1. As the number of interactions increased, primary effects on sub-entities increased, and more progress interactions occurred (since the number of progress

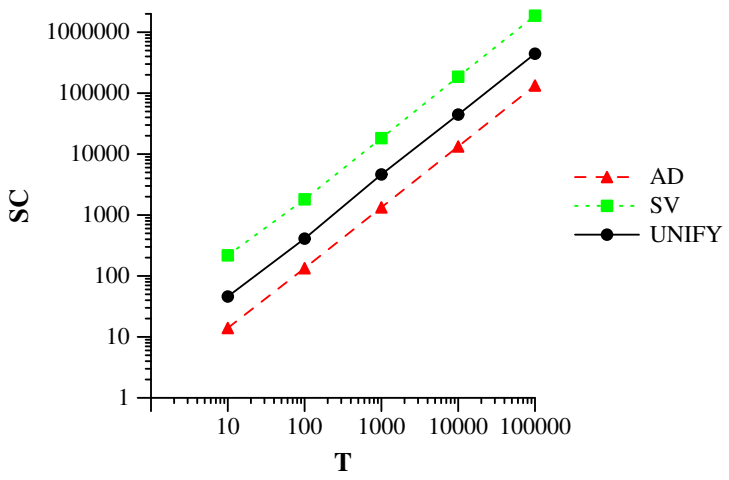
interactions was T÷R). Therefore, SC increased with T for all approaches (Figure 55). SC_{SV} increased the most since all interactions were translated into equivalent highest-resolution interactions. The translation usually resulted in more interactions being generated since a low-resolution interaction affects many sub-entities at higher resolution levels.

2. As the number of interactions increased, secondary effects on sub-entities increased. Therefore, CC_{AD} and $CC_{\texttt{UNIFY}}$ increased with T (Figure 56). No consistency maintenance is required for SV since only one level is present.
3. As R increased, progress interactions occurred less frequently, since the number of progress interactions was T÷R. Accordingly, SC decreased with an increase in R for all approaches (Figure 57).
4. The increase or decrease in R did not change CC since the progress interactions were purely simulation interactions. Accordingly, CC was unaffected by R for all approaches (Figure 58).
5. As the number of sub-entities for each level increased, SC_{SV} and $SC_{\texttt{UNIFY}}$ increased polynomially. SC_{SV} increased because an increase in the number of sub-entities increased the number of translated interactions. SC_{SV} and $SC_{\texttt{UNIFY}}$, increased because a greater number of sub-entities resulted in a greater number of actions when progress interactions occurred. SC_{AD} was independent of F because the effects of all interactions were applied at level 0 (Figure 59).
6. As the number of sub-entities for each level increased, CC increased polynomially for all approaches (Figure 60). An increase in F resulted in an increase in CC_{SV} because more sub-entities were created initially and destroyed

finally. CC_{AD} increased with F because more sub-entities were created and destroyed during aggregation and disaggregation. $CC_{\texttt{UNIFY}}$ increased with F because more effects were propagated to other resolution levels.

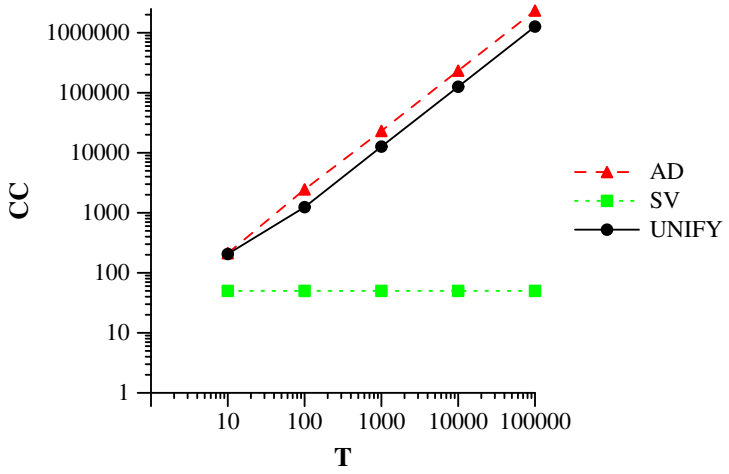
7. As the number of levels increased, SC_{SV} and $SC_{\texttt{UNIFY}}$ increased exponentially. SC_{SV} increased because the greater the number levels, the greater the number of translated interactions. For SC_{SV} and $SV_{\texttt{UNIFY}}$, a greater number of levels resulted in an greater number of actions for progress interactions. SC_{AD} was independent of L since the effect of all interactions, including progress interactions, were applied at level 0 (Figure 61).
8. As the number of levels increased, CC increased exponentially for all approaches (Figure 62). CC_{SV} increased with L because more sub-entities were created initially and destroyed finally at level $L-1$. CC_{AD} increased with L because more sub-entities were created and destroyed during aggregation and disaggregation. $CC_{\texttt{UNIFY}}$ increased with L because more effects were propagated to other resolution levels.

In Figure 63, we plot SC, CC and Total Cost using each approach for the canonical case (L = 3, F = 2, T = 1000, R = 5). Total Cost is a weighted sum of simulation and consistency costs. The weights for SC and CC are application-specific; in the graph in Figure 63 we assign equal weights to them, i.e. Total Cost = SC+CC. `UNIFY` incurs the least total cost in this case. Other cases in which the values of the above parameters were varied indicate similar trends.



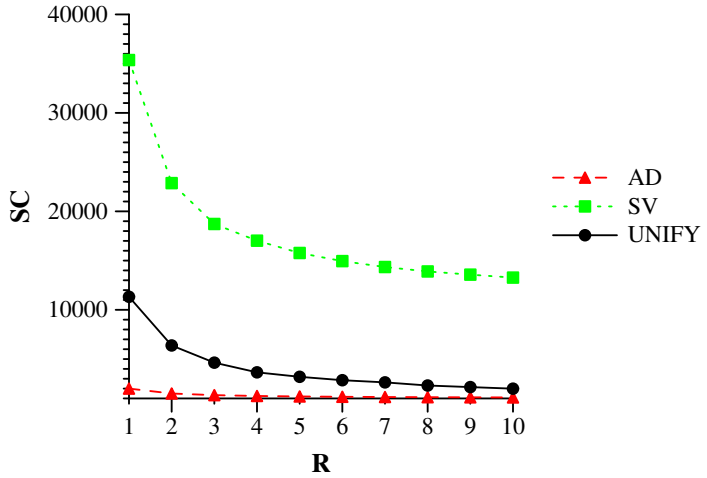
L = 3, N = 5, R = 3

FIGURE 55: Simulation Cost varying with Number of Interactions



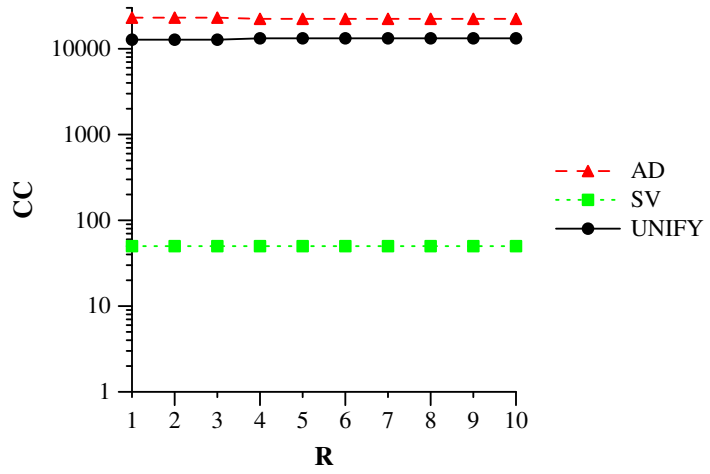
L = 3, N = 5, R = 3

FIGURE 56: Consistency Cost varying with Number of Interactions



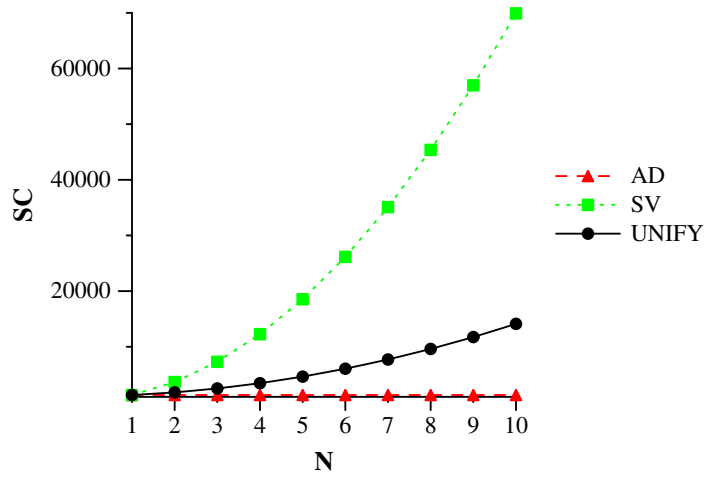
L = 3, N = 5, T = 1000

FIGURE 57: Simulation Cost varying with Rate of Simulation



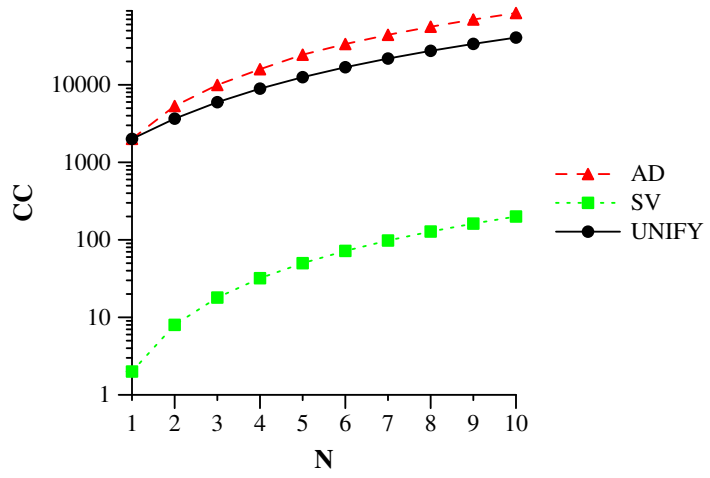
L = 3, N = 5, T = 1000

FIGURE 58: Consistency Cost varying with Rate of Simulation



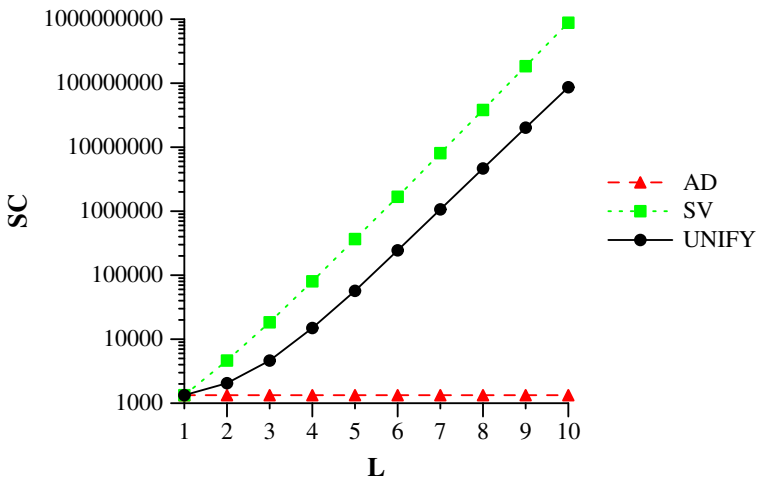
L = 3, T = 1000, R = 3

FIGURE 59: Simulation Cost varying with Number of Sub-entities



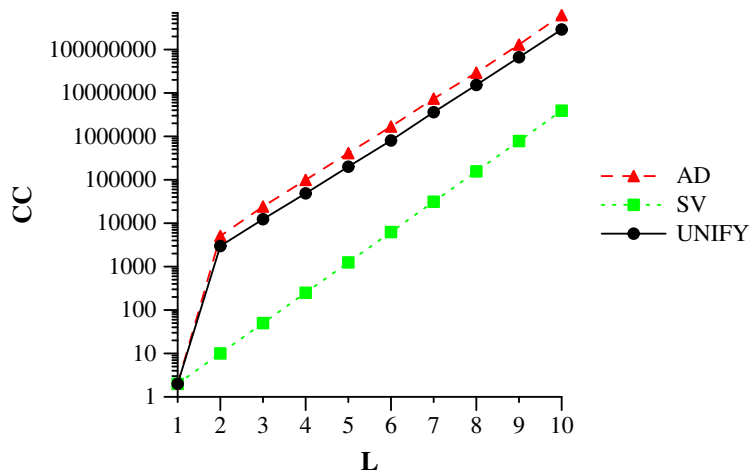
L = 3, T = 1000, R = 3

FIGURE 60: Consistency Cost varying with Number of Sub-entities



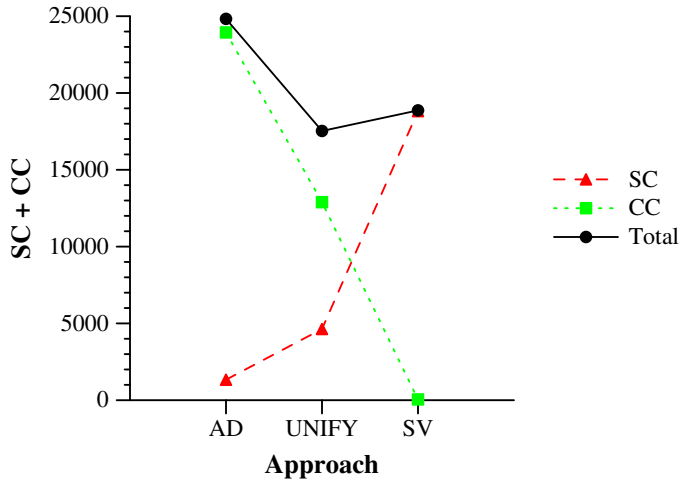
N = 5, T = 1000, R = 3

FIGURE 61: Simulation Cost varying with Number of Levels



N = 5, T = 1000, R = 3

FIGURE 62: Consistency Cost varying with Number of Levels



L = 3, N = 2, T = 1000, R = 3

FIGURE 63: AD, SV and `UNIFY` — Cost Comparison

9.1.3.6 Summary of Cost-Effectiveness

UNIFY satisfies R3 by enabling reductions in the costs of simulation and consistency maintenance. Although selective viewing incurs low consistency cost and aggregation-disaggregation incurs low simulation cost, both approaches fare poorly when both costs are considered. In contrast, UNIFY achieves lower total cost than either aggregation-disaggregation or selective viewing. An approach that achieves MRM at a high cost is ineffective because it does not satisfy R3. UNIFY enables the total of simulation and consistency costs to be reduced, thus satisfying R3.

9.1.4 Summary of Evaluation in Terms of MRM Requirements

UNIFY satisfies our three requirements for effective MRM: multi-representation interaction (R1), multi-representation consistency (R2) and cost-effectiveness (R3). These requirements must be satisfied by any approach in order to achieve effective joint execution of multiple models at reasonable cost. Alternative approaches such as aggregation-disaggregation and selective viewing do not satisfy all of R1, R2 and R3. Therefore, by these criteria, UNIFY is better than the popular MRM approaches.

9.2 Applying UNIFY to Existing Models

We have applied UNIFY to four models. Three of them are military models specified using OMT. The fourth is a hierarchical autonomous agent that is a research project at the University of Virginia. For all four models, we constructed an MRE from attributes at multiple representation levels. We constructed an ADG for each MRE. We classified the interactions in each model according to our taxonomy. For each model, we assumed

reasonable mapping functions and policies for resolving concurrent interactions. For each model, we worked only from specifications, since pursuing the project to implementation would have been an unreasonably large undertaking.

9.2.1 Military Models

The three military models we considered are part of the Department of Defense's High Level Architecture (HLA). They are: Joint Task Force prototype (JTFp) [JTFP97], Joint Precision Strike Demonstration (JPSD) [JPSD97] and Real-time Platform Reference (RPR) [RPR97]. These models have been the basis of many examples that we provided in this dissertation to explain techniques in UNIFY. The process for applying techniques in UNIFY to these models is shown in Chapter 8:

1. Construct a Multiple Representation Entity (MRE) from the OCST.
2. Capture relationships among the attributes with an Attribute Dependency Graph (ADG) constructed from the APT and the ART (see §8.2).
3. Select mapping functions for each dependency in the ART.
4. Determine the effects of interactions from the OIT, and classify interactions according to our taxonomy.
5. Resolve the effects of concurrent interactions from policies specified in the CIT (see §8.2).
6. Construct a Consistency Enforcer and an Interaction Resolver for the MRE.

The results of our experience with these models are a proof-of-concept for UNIFY. Designers of jointly-executing battlefield models can achieve effective MRM by applying UNIFY. For each of these models, we were able to apply UNIFY, thus avoiding pitfalls

encountered with alternative MRM approaches. Details of how we applied UNIFY to JTFp, JPSD and RPR appear in Appendices B, C and D respectively.

9.2.2 Autonomous Agent Model

We applied UNIFY to a hierarchical autonomous agent model [WAS98B]. The autonomous agent model we considered is part of a research project undertaken by the Vision Group at the University of Virginia. The autonomous agent, Marcus, has been programmed to construct complex arrangements from basic building blocks. Figure 64 shows Marcus with an example arrangement, an archway.

Marcus is a hierarchical autonomous agent that has two models, one corresponding to a planner and the other corresponding to a perception-action (PA) system. Typically, the planner maintains long-term or abstract representation, whereas the PA system maintains immediate and detailed representation. Each model may have its own representation of the world in which Marcus operates. Accordingly, each model may represent building blocks, partially-completed arrangements, obstacles, doors and pathways by a number of relevant attributes such as position, orientation and colour. Marcus considers relationships among blocks that are stacked or placed next to each other as an arrangement.

We constructed an MRE for Marcus's planner and PA system and captured dependencies among attributes with an ADG. In the current implementation of Marcus, interactions occur only at the PA level through sensors and effectors. Planner-level interactions originating from user directives are envisioned as future work. Therefore, we classified interactions at only the PA level. Figure 65 shows a partial ADG for an MRE constructed from the planner and PA representations for Marcus. The MRE contains all of

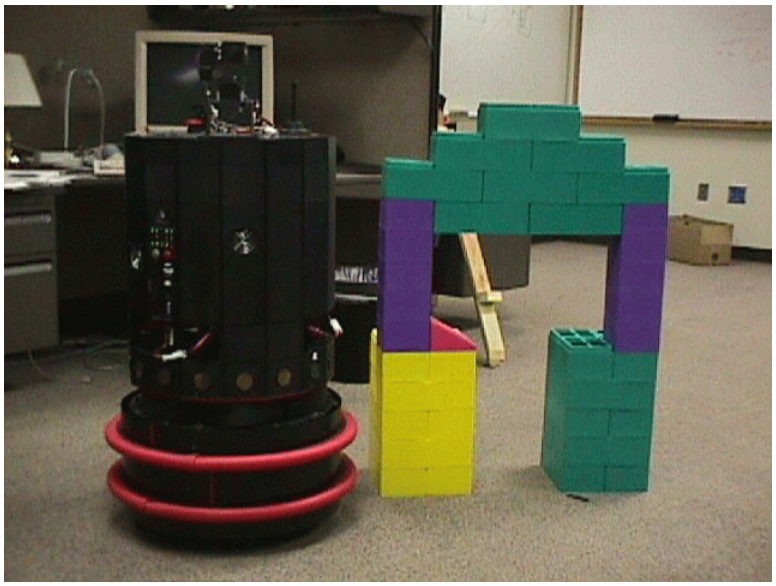


FIGURE 64: Marcus and Archway

the objects (and their attributes) that the planner considers important for the current task, and all of the objects (and their attributes) that the PA system senses and affects. For brevity, we show only objects represented by the planner and PA, but not their attributes. We show dependencies that exist among objects when Marcus constructs an arrangement. Wasson shows how representations can be constructed for the models in Marcus and how consistency can be maintained among the representations [WAS99].

Our experience with the hierarchical autonomous agent model indicates that the techniques in UNIFY can be applied to multi-models in different domains. A valid

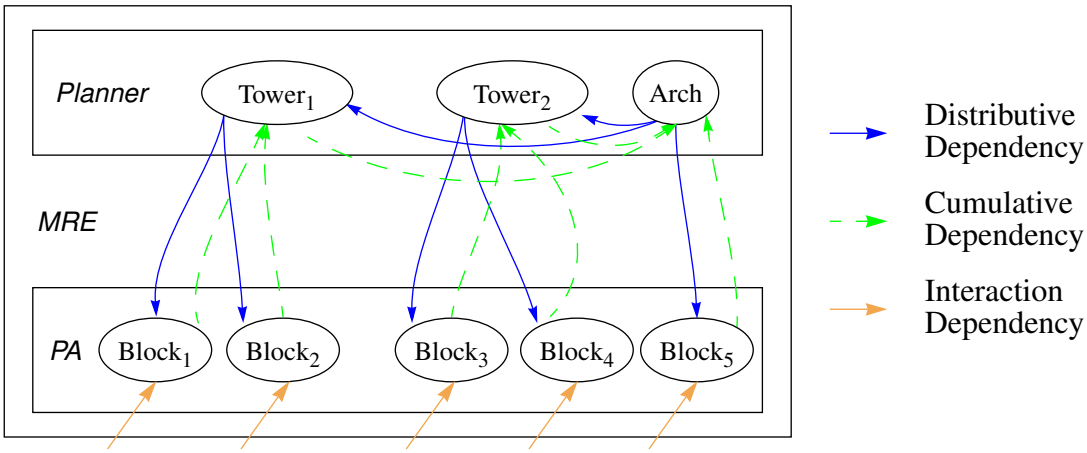


FIGURE 65: MRE for planner and PA system representations

concern with any framework-based approach is whether the framework is general enough to be useful to applications in many domains. Applying UNIFY to applications in many domains would be a convincing, but time-consuming, argument for the applicability of UNIFY. We chose one domain — that of hierarchical autonomous agents — to show that UNIFY can be applied to many domains. Details of how we applied UNIFY to a hierarchical autonomous agent appear in Appendix E.

9.3 Limitations

A fair evaluation of any research must include the known limitations of the work. The underlying feature of our work is a design decision to maintain concurrent representations of jointly-executing models to enable effective MRM. In order to support this decision, we constructed a framework, UNIFY, consisting of techniques and processes for achieving effective MRM. However, in order to make UNIFY a viable approach for MRM, we made some assumptions about jointly-executing models. These assumptions are the limitations of UNIFY. These limitations, individually and together, neither make UNIFY unusable nor outweigh its benefits.

UNIFY is limited to models in which representation exists for objects and processes that are part of a model. We assume that designers can describe properties of objects and processes in a model, i.e., they can represent a model. UNIFY is not applicable to models wherein representation does not exist. Our assumption about representation is reasonable because a large number of practical models represent objects and processes.

In UNIFY, we assume that individual models meet their users' requirements. UNIFY permits designers to capture the combined semantics of multiple models of the same

phenomenon. Whether the individual models meet their users' requirements or not is an important issue, but outside the scope of our work. Our work addresses the effectiveness of the joint execution of multiple models alone.

In UNIFY, we assume that multi-models progress in compatible time-steps. We discussed compatible time-steps in §5.2. We regard our assumption of compatible time-steps as the most critical assumption in UNIFY. General techniques for achieving compatible time-steps would be a desirable addition to UNIFY.

UNIFY requires appropriate mapping functions to translate attributes from one representation to another and appropriate policies for resolving the effects of dependent concurrent interactions. We do not regard our assumptions about the presence of mapping functions and interaction policies as critical assumptions because:

1. Mapping functions and interaction policies capture semantic information about an application. Semantic information is specific to an application and can be provided by a designer.
2. Alternative approaches to MRM also require similar mapping functions and interaction policies (see §5.3).
3. We guide designers in the selection of mapping functions and interaction policies (see §6.2 and §7.6).

Despite these limitations, UNIFY is a viable approach for MRM. Its benefits outweigh its limitations. It eliminates or reduces many problems with alternative MRM approaches (see §5.5). It provides designers with techniques for resolving concurrent interactions (see Chapter 7) and applying their effects consistently (see Chapter 6). It provides designers

with a process for achieving MRM (see Chapter 8) effectively and practically (see §9.1 and §8.2). Hence, UNIFY enables designers to achieve effective MRM.

9.4 Chapter Summary

UNIFY is a sufficient and practical approach for effective Multi-Representation Modelling (MRM). It is the first known approach to MRM that satisfies R1, R2 and R3. Its limitations are not serious. We have applied it to four practical applications and established that it supports MRM exactly as we have claimed it would. Next, discuss contributions of our work and present future directions for research.

Never rise to speak till you have something to say;
and when you have said it, cease. — John Witherspoon

When the effective leader is finished with his work,
the people say it happened naturally. — Lao-Tzu, Tao Te Ching

Chapter 10

Conclusions

We presented a sufficient and practical framework, UNIFY, for effective Multi-Representation Modelling (MRM). MRM, the joint execution of multiple models, is a significant challenge facing model designers. Previous approaches have been unsuccessful in helping model designers overcome this challenge; these approaches they do not satisfy all of our requirements for effective MRM. The techniques and processes that are part of UNIFY help designers to overcome the challenge of executing multiple models jointly by enabling consistency maintenance among the concurrent representations of the models. UNIFY is a sufficient approach for achieving effective MRM because it satisfies the requirements for effective MRM. UNIFY is practical because designers can apply it in conjunction with a familiar model specification methodology. UNIFY is a significant contribution to the practice of modelling and simulation.

Previous MRM approaches such as aggregation-disaggregation and selective viewing can fail to achieve effective MRM for many applications because they do not satisfy critical MRM requirements. These approaches encounter many problems such as temporal

inconsistency, chain disaggregation and thrashing, which render the approaches ineffective for many applications. Our fundamental observations about jointly-executing models address the causes of these problems. These observations indicate that maintaining consistency among the representations of jointly-executing models can eliminate or reduce the problems encountered in other approaches.

UNIFY, our approach for achieving effective MRM, involves maintaining consistency among concurrent representations. The techniques and processes in UNIFY address consistency maintenance in concurrent representations. The viability of UNIFY rests on the assumptions that designers can (i) select mapping functions to capture application-specific aspects of attribute relationships, (ii) select policies to resolve the effects of concurrent interactions by understanding their semantics, and (iii) make time-steps compatible. These assumptions are reasonable because without them, no approach can capture the application-specific semantics of jointly-executing models. Alternative approaches fail to achieve effective MRM despite making similar assumptions.

UNIFY aids designers in incorporating MRM effectively in their applications. Effective MRM leads to the design of multi-models that satisfy their users' requirements. We provided guidelines for designers so that they can apply our techniques and processes to achieve effective MRM within their applications.

10.1 Contributions

Our work benefits the practice of modelling and simulation. UNIFY is the first known framework for effective MRM. The focus of UNIFY is to execute multiple models jointly. UNIFY is intended for designers who desire to incorporate MRM into their applications.

These designers can construct MRM solutions for their applications by applying the techniques and processes within UNIFY.

The main contribution of our work is UNIFY — a framework for the joint execution of multiple models. We formulated three requirements for MRM: multi-representation interaction, multi-representation consistency and cost-effectiveness. We showed how alternative MRM approaches do not satisfy these requirements, while UNIFY does. The contributions of our work are the following:

1. Fundamental Observations about MRM
2. UNIFY
    a. Multiple Representation Entities (MREs)
    b. Attribute Dependency Graphs (ADGs)
    c. Properties and requirements of mapping functions
    d. Process for constructing Consistency Enforcers (CEs)
    e. A Taxonomy for Interactions
    f. Process for constructing Interaction Resolvers (IRs)
3. A Cost Study of various MRM approaches
4. Guidelines for MRM designers

We presented the fundamental observations to show how problems arise in the joint execution of multiple models [REYN97]. We made these observations after studying the joint execution of many models. The fundamental observations address the causes of ineffectiveness in jointly-executing models, such as inconsistency among their representations and dependent concurrent interactions. Addressing the fundamental observations forms the basis of any approach to effective MRM, such as UNIFY.

MREs are an approach for maintaining concurrent representations of jointly-executing models [NAT95]. An MRE permits interactions at all representation levels, yet is internally consistent. MREs eliminate or reduce many problems seen with alternative MRM approaches, such as aggregation-disaggregation and selective viewing. MREs eliminate chain disaggregation, temporal inconsistency, mapping inconsistency, transition latency and thrashing, and reduce network flooding. MREs require a means of capturing the relationships among multiple representations and policies to resolve the effects of concurrent interactions. Provided these requirements are satisfied, MREs reduce the MRM problem to the problem of maintaining consistency among concurrent representations when interactions at multiple representation levels occur.

ADGs and mapping functions capture relationships among concurrent representations. ADGs are a technique to capture dependencies among attributes in an MRE, whereas mapping functions capture application-specific information about the dependencies. ADGs permit designers to express how attributes in representations are dependent on one another, and how the execution of a multi-model affects the representations of each model. Mapping functions translate attributes from one representation level to another. ADGs and mapping functions can be used to construct a CE for an MRE. A CE is responsible for maintaining an MRE consistent at all observation times. When an interaction changes the value of an attribute, a CE traverses an ADG and invokes the appropriate mapping functions in order to maintain consistency in an MRE. We demonstrated the construction of a CE by showing how to construct an ADG and select mapping functions for an MRE. We showed how to assign static and dynamic semantics to dependencies captured by an ADG by classifying dependencies into four types and weighting them. We presented

requirements and properties of mapping functions. We discussed how an ADG can be traversed in order to propagate the effects of an interaction. Finally, we presented an algorithm for the operation of a CE.

We presented one taxonomy for classifying interactions semantically and resolving their dependent effects [NAT99]. We presented four characteristics of interactions and showed how to classify interactions into four classes based on these characteristics. We showed how serialization, the traditional approach for resolving the effects of concurrent interactions, can be inappropriate for dependent concurrent interactions. Based on our taxonomy, we presented policies for resolving the effects of classes of dependent concurrent interactions. Our taxonomy is applicable to interactions in a variety of modelling and simulation applications. We believe that in any application where concurrent interactions may be dependent on another, such a taxonomy is applicable and can be used to resolve the effects of concurrent interactions. We demonstrated the construction of an IR and presented an algorithm for its operation.

We presented the first cost study comparing various MRM approaches [NAT97]. The study compares simulation and consistency costs for `UNIFY` and alternative approaches. We showed how simulation and consistency costs vary for the different approaches. Lastly, we showed that `UNIFY` reduces the total of simulation and consistency costs.

The fundamental observations, MREs, ADGs and our taxonomy of interactions enable designers to incorporate effective MRM in their applications. Providing designers with techniques and guidelines to achieve effective joint execution of multiple models is our main contribution to modelling and simulation.

10.2 Future Work

In the future, we expect to eliminate a few of the assumptions we made in UNIFY and apply UNIFY to applications in a variety of domains. Eliminating some of the assumptions we made in our work would make UNIFY more beneficial to model designers. Applying UNIFY to more applications, would provide us with greater experience with regard to MRM.

A critical assumption we made was that designers can make the time-steps of jointly-executing models compatible. Jointly-executing models executing with compatible time-steps can be temporally consistent. Application-independent guidelines for making time-steps compatible would be a desirable addition to UNIFY. Alternatively, providing techniques for maintaining temporal consistency among jointly-executing models that execute with incompatible time-step would eliminate a critical assumption in UNIFY.

Another assumption was that designers can select mapping functions to translate attributes among representations. We specified requirements and properties of mapping functions as guidelines for selecting them. However, specifying requirements and properties in greater detail, perhaps for classes of applications, would enable designers to select mapping functions with greater ease.

Yet another assumption was that designers can select policies for resolving the effects of concurrent interactions after classifying the interactions. We showed how to classify interactions and select policies for resolving classes of interactions. Providing sub-classes of interactions would enable designers to refine the classification of the different kinds of interactions in various applications. Refined classification may lead to refined policies for resolving the effects of concurrent interactions.

An area of future work would be applying UNIFY to a larger variety of models. Applying UNIFY to a wide variety of models would increase our understanding of MRM. We would like to apply UNIFY to models in areas such as economics, weather prediction and graphics. Applying UNIFY to such models would enable us to specify detailed requirements and properties of mapping functions and to refine the classification of interactions. Also, we would like to study the implementation of applications that employ UNIFY to incorporate MRM. Such studies details may reveal connections between requirements and properties of mapping functions, policies for resolving concurrent interactions and the implementation of modules for enforcing consistency and resolving interactions. UNIFY can gain widespread acceptability if it is applied successfully to a large number of multi-model applications.

Honest disagreement is often a good sign of progress.
— Mahatma Gandhi

Appendix A

Examples of Multiple Representations

Multi-model applications in a number of domains maintain multiple representations or views with some degree of concurrence and consistency. In §2.1, we presented these applications briefly and evaluated whether they satisfy the MRM requirements R1, R2 and R3 (Table 1). Here, we evaluate these applications in detail. Briefly, we discuss how an approach based on MREs may benefit these applications.

A.1 Multi-Resolution Graphical Modelling

Multi-resolution graphical modelling involves maintaining multiple representations, or *levels of detail*, of the same object [CLARK76]. For example, a lamp may be rendered in full detail when a viewer is close to it, but as the viewer moves away, successively coarser levels of detail are rendered. As the viewing distance increases, the lamp occupies a smaller portion of the viewed screen space, and coarser levels of detail for the lamp are sufficient to cover this portion. The coarser the level of detail, the fewer the polygons required to render it. The system always maintains multiple levels of detail for all objects,

and selects the appropriate level of detail depending on an object's distance from the viewer. The challenges in graphical MRM are to generate the multiple representations such that each captures sufficient detail as to be visually appealing, and to transition among representations smoothly [GAR95] [HECK94] [HECK97] [LUEBKE97] [PUPPO97].

Typically, users cannot change multi-resolution graphical entities, although they may issue *view interactions*, which essentially read the values of attributes such as position and colour. Moreover, concurrent interactions to multiple levels of detail of the same object are not supported. Since interactions cannot change entities' representations and cannot be concurrent, R1 is violated. R2 is satisfied trivially after the levels of detail are created because the representations do not change. A few multi-resolution graphical models permit a single user to change entities dynamically, but do not support concurrent multi-representation interaction [BERM94] [LEE98] [ZORIN97]. Multi-resolution graphical models satisfy R3 because multiple levels of detail reduce simulation cost. As long as interactions cannot change the representations of objects, consistency cost is not an issue.

An MRE for a multi-resolution graphical model would incorporate all levels of detail. Designers can create the multiple levels of detail using refinement or simplification [HECK97] [LUEBKE97]. Refinement and simplification methods can be the mapping functions among the multiple levels of detail. When changes to any levels of detail occur, these mapping functions can ensure that the other levels of detail are changed so as to keep the MRE consistent. Consequently, an MRE for a graphical object may interact concurrently and consistently at multiple representation levels.

A.2 Hierarchical Autonomous Agents

An autonomous agent is an actual or simulated robot that attempts to fulfill a goal by performing actions from its basic skill set. Traditionally, there have been two approaches regarding the manner in which an agent fulfills its goal. In the *deliberative* approach, an agent constructs a plan to fulfill its goal by composing actions from its skill set before beginning any action [SACER74]. The agent may form optimal or provably correct plans; however, unexpected occurrences can sabotage any plan easily. In the *reactive* approach, an agent forms no plan at all, relying on reactions to external stimuli to fulfill its goal [AGRE87]. This approach leads to extremely robust behaviour in the presence of urgent or unexpected circumstances; however, the agent may become trapped in local minima.

Multi-layered architectures for autonomous agents incorporate a deliberative layer (a planner) and a reactive layer (a perception-action or PA layer) with some intermediate layers. Multi-layered architectures balance varying requirements and capabilities of different layers, e.g., level of abstraction, amount of inference, time-scale and bandwidth [ALBUS97] [BON97] [FIRBY87] [GAT92] [LAIRD91] [HANKS90] [SIM94] [WAS98A].

Multi-layered, or hierarchical, autonomous agents satisfy R1. Such agents execute deliberative and reactive models jointly in order to take advantage of both. The planner and PA layer representations* are linked epistemologically, i.e., subsets of representations encode knowledge that depends on or is derived from knowledge encoded in other subsets [BRILL96]. Hierarchical agents do not satisfy R2 because dependencies among planner

* Although Brooks argues against representation in an agent [BROOKS86], Brill has shown that agents with representation can be effective [BRILL98].

and PA layer representations can give rise to inconsistencies. For some desired agent behaviour, the paradigm of executing both models jointly is more cost-effective than executing only one model. Hence hierarchical agents satisfy R3.

Provided designers can agree on what must be represented at each layer of hierarchical autonomous agents, an MRE for such agents would incorporate the representation for each layer. Typically, we can capture dependencies between the representations by simple relationships, such as `has-part` and `is-a` [WAS98B]. By ensuring that the individual relationships hold, we can maintain consistency between the representations.

A.3 Blackboard Systems

`Hearsay-II` is a layered system for translating spoken sentences into the corresponding alphabetic representation. In `Hearsay-II`, many processes access a single data structure, called a *blackboard* [ERMAN80]. Processes are *data-driven*, i.e., a process activates itself whenever appropriate data appears on the blackboard. The lowermost layer of the system interprets parts of sound waves as silence or non-silence. The next layer interprets non-silence as phonemes and predicts the sound wave corresponding to the next likely phoneme. The next layer composes phonemes into syllables and predicts the next word. The hierarchy of layers continues with the topmost layer composing phrases into sentences and predicting the next phrase.

`Hearsay-II`'s blackboard is a multi-representation system; each layer is a different model of the entire spoken sentence. R1 is satisfied because for each sentence fragment, the interpretation of the current layer and the prediction of the layer above are multi-representation interactions. `Hearsay-II` resolves conflicting interactions — different

predictions or interpretations of the same sentence fragment — by retaining each as a version of the sentence. Each version is consistent — the wavelets are consistent with the phonemes, the phonemes with the syllables, and so on — thus satisfying R2. The system ranks all versions by a credibility metric; the version with the highest credibility is the best translation of the sentence. However, retaining all versions may be impractical in a general sense, since many multi-representation systems may not tolerate multiple outcomes.

Each version of a sentence in `Hearsay-II` is similar to an MRE. However, `Hearsay-II` violates R3 because it resolves conflicting interactions by creating new MREs that subsequently execute concurrently with existing MREs. In effect, each MRE executes in a "parallel universe" in which it is the most credible version. The greater the number of versions, the greater the number of MREs in execution, putting a strain on available resources. Our technique of resolving concurrent interactions within a single MRE may miss the best possible version of a sentence when local minima occur. However, when many objects or processes are present in a system, `Hearsay-II`'s technique of creating a new MRE for every possible outcome of conflicting concurrent interactions can cause a combinatorial increase in consumption of resources.

A.4 Cache Coherence

In a multi-processor configuration, individual processors may access a small amount of fast memory locally in order to reduce accesses to main memory, which tends to be slow. The fast memory, called a *cache*, may store copies of data items stored in main memory. Processors may read and modify data items in their caches. Ensuring that

processors read correct versions of the data in their caches is known as the *cache coherence* problem [HENN96].

Cache coherence is a form of the MRM problem. The main memory copy and each cache copy of a single data item are concurrent representations of a variable. Processes issue interactions in the form of read and write operations to the copies. Processors may interact with cache copies as well as the main memory copy, the latter when a processor's cache copy is absent or stale or, in the case of write-through policies, whenever the processor writes to the variable. Concurrent interactions at multiple representation levels are assumed to have independent effects. Since multi-representation interactions may occur, but dependent interactions are not supported, R1 is satisfied partially. Cache coherence involves combining detection mechanisms such as snoopy bus or directory-based protocols with write policies such as write-back and write-through to maintain consistency among cache and memory copies. Although cache coherence solutions maintain consistency, typically, the relationships among memory and cache copies are simple relationships of equality. Therefore, cache coherence satisfies R2 partially. Various cache coherence strategies have different costs associated with them [ARCH86]. However, accessing caches is more cost-effective than accessing memory. Hence caches satisfy R3.

We do not forward any new solutions for cache coherence. Rather, we use the cache coherence example to highlight the benefits of maintaining concurrent representations.

A.5 Abstract Data Types and Object Inheritance

Polymorphic languages may associate multiple types for a single data item. For any data item, the operations that are valid on it, the contexts in which it can be used legally

and the manner in which it is allocated memory are determined by its type. If a data item has multiple types, the operations valid on it and the contexts in which it can be used is the union of the operations and contexts respectively for the individual types. Typically, the memory allocated to the data item is such that the data item has a single representation.

Some abstract data types present multiple views of the same data item, thus exhibiting a form of MRM. A data item defined as a union in C [KERN88] and C++ [STROU91] or as perspectives [GOLD80] [STEFIK86] can have multiple types, thus displaying *ad hoc* polymorphism [CARD85]. Consider the definition of a union in C:

```
union {
   int a;
   char b;
} X;
```

The data item `X` has two types, `int` and `char`, corresponding to `X.a`. and `X.b` respectively. `X.a` and `X.b` are different views of `X`. They occupy overlapping bytes of memory, i.e., if an `int` is stored as two bytes on a particular system and a `char` is stored as one byte, then `X` is allocated two bytes of memory. One byte holds the value of `X.b` as well as part of the value of `X.a`, while the other byte holds the remaining part of the value of `X.a`. `X.a` and `X.b` are accessed jointly by any operation accessing one of them, i.e., if an operation changes the value of `X.b`, it changes the value of `X.a` as well and *vice versa*. A type is a representation level; therefore, operations of different types constitute multi-representation interactions. However, these interactions are assumed to be independent of one another. Therefore, R1 is satisfied partially. Changing the value of any type automatically changes the value of other types for a data item. However, unions cannot capture general relationships, such as those among attributes of an MRE. For example, a

union cannot present two views such that a value in one view is an accumulation of values in another view. Hence, *ad hoc* polymorphism satisfies R2 partially.

Inheritance in object-oriented languages is an example of MRM, since a data item that inherits from one or more types has multiple views. Object-oriented languages such as Smalltalk-80 [BORN82], Simula-67 [DAHL66] [BIRT73] and C++ [STROU91] support inclusion polymorphism [CARD85]. Consider the C++-like example below:

```
class Mammal { ... }
class Oviparous { ... }
class Platypus: Mammal, Oviparous { ... } Bill;
```

Here, `Mammal` and `Oviparous` are base classes for the derived class `Platypus`. `Bill` is an instance of the class `Platypus`, and by inheritance, also an instance of the classes `Mammal` and `Oviparous`. Inheritance results in `Bill` having multiple views: one, as an instance of a base class and two, as an instance of a derived class. One view is subsumed by another; the view as `Mammal` is a subset of the view as `Platypus`. Multiple inheritance results in `Bill` having even more views. However, the views do not subsume each other; the view of `Bill` as a `Mammal` has no relation to the view of `Bill` as an `Oviparous`. The representation for `Bill` is the union of the representations defined by each of the above classes, assuming name conflicts are resolved. Likewise, the set of methods applicable to `Bill` is the union of the set of methods defined by each class. A class is a representation level; therefore, methods of the multiple classes constitute multi-representation interactions. However, these interactions are assumed to be independent of one another. Therefore, R1 is satisfied partially. Any operation that is performed on an instance of a derived class is performed on an instance of the base class as well. Therefore, the instance of the derived class is always self-consistent. However,

inheritance is only one kind of relationship among attributes of an MRE; for example, inheritance does not capture the accumulation relationship mentioned earlier. Hence, inclusion polymorphism satisfies R2 partially.

A.6 Views in Databases and Integrated Environments

Views, as defined for databases and integrated environments, are a form of MRM. A view in a database is a subset of the information contained in the system.

Database views are derived from the complete database by specifying relations that restrict the items displayed. In relational database applications, data are abstracted into relations, which essentially are tables of tuples and their values [CODD70]. Relational databases have been used for many applications [ASTRA76] [STONE76] and programming environments [LINTON84]. In object-oriented databases, data are abstracted as behavioral entity relationships [CHEN76] [BALZER85]. Hybrid approaches that maintain relations as well as attribute relationships have been used for editing programs [HOR86]. A view is a set of relations derived from existing relationships (in an object-oriented database) or relations (in a relational database) [CHAM75]. Changes to a view must be translated to changes to the database in order to maintain consistency in the database [BAN81]. Since all views are derived from one database, this approach is a form of selective viewing, which violates R3. Each view is constructed after the entire database has been constructed. Users may update data in any view; however, all updates are assumed to be independent. Hence, views in databases satisfy R1 partially. When users update data in a view, the system updates the database automatically, thus maintaining consistency and satisfying R2. Views in databases require the database to be the repository of all possible views, thus

making them unsuitable for MREs, wherein multiple representations may have been designed independently. Moreover, relations are powerful but not intuitive for some kinds of relationships [HOR86].

In some integrated environments, the complete database is constructed by conjoining all the views [GAR87]. Individual tools may maintain views of their data. Users may update data in any view as well as in the database; however, all updates are assumed to be independent. Hence, views in databases satisfy R1 partially. Users' updates are interactions. The system updates the views and databases to remain consistent with one another, thus satisfying R2. In database views, if each view is an independently-designed model, then requiring a single database for all the models put together violates R3. In contrast, in integrated environments, if each view is an independently-designed model, then the complete database is just the conjoining of the models, which satisfies R3. The latter approach is related closely to MREs. Each view may be considered a concurrent representation and the conjoining of all the views is the MRE. Garlan's work envisioned the multiple views to be used by tools that change databases. In MREs, other entities, other models and the environment change a representation.

A.7 Nested Climate Modelling

An increasingly popular approach to climate modelling is to nest the execution of Limited Area Models (LAMs), which predict regional climate, within Global Circulation Models (GCMs), which predict wide-ranging climate changes [GIORGI90] [GIORGI91] [RISBEY96]. GCMs model synoptic or large-scale climate changes. The resolution of these models is usually in the hundreds of kilometres, which means that regional climatic

variations are modelled poorly. LAMs model mesoscale or medium-scale climate changes. The resolution of these models is in kilometres, hence they model local climate well but at a huge computational cost. Of late, small sub-areas of the larger area modelled by the GCM are taken over by LAMs which discard all the GCM modelling information except at the edges of the sub-area modelled by each LAM. Subsequently, each LAM runs its own computations to predict local climates. The GCM-LAM linkage produces more accurate predictions than either a GCM alone (since the LAM usually has more detailed topographical and orographical information) or just a LAM driven by empirical data (which assumes that future climate will be very similar to past climate).

Nested climate models satisfy R1 but not R2. The nested models interact at multiple representation levels since climatic data at either level is incorporated. However, while researchers have had success translating GCM data for LAM input, the reverse is an open problem. As a consequence, global factors such as temperature fronts, monsoons and large mountain ranges can influence local climate models, but it is extremely hard to make local factors such as fires, nuclear waste build-up, small mountain ranges and anthropogenic pollution influence global climate models. Nested climate models satisfy R3 because they are more cost-effective than executing any one model individually.

A climate model MRE would incorporate GCM and LAM representations for a particular area. As a result, the climate of the area would be influenced by global factors as well as local factors. Nesting the LAM within the GCM would be one way to reconcile concurrent climate changes. However, as discussed earlier, this tends to make the execution of the LAM dominate the execution of the GCM, particularly close to the center of the area modelled by the LAM. In terms of accuracy of predictions, the MRE approach

can do no worse than nesting; the potential to do better lies in the ability of the MRE approach to capture dependencies between the two representations that are ignored in nesting. However, the limiting problem in either approach is the lack of techniques to translate local factors into global factors.

A.8 Integrated Molecular Modelling

When theoretical studies on the potential energy surfaces for chemical reactions of large systems are carried out, low-detail low-computation models, such as molecular mechanics (MM) models, are used for most of the system and high-detail high-computation models, such as molecular orbital (MO) methods, are used for a small part of the system. An MM model for the entire system is usually fast but inaccurate since the level of detail does not capture all interactions among atoms. An MO model for the entire system is accurate but computationally expensive. Integrated models such as `IMOMM` [MATSU96] and `ONIOM` [SVEN96A] strike a balance between resource usage and accuracy. These approaches integrate MM models, such as `MM2`, `MM3`, `CHARMM`, `AMBER` and `UFF`, with MO models such as Møller-Plasset second-order perturbation (`MP2`) and Hartree-Fock (`HF`), in order to compute potential energy surfaces. Some approaches, for example `IMOMO`, integrate an high-detail MO model with a low-detail MO model [HUMBEL96] [SVEN96B]. In all the integrations, the reported accuracy is comparable to a full-scale high-detail model, while resource usage is markedly below such a model.

Integrated molecular models satisfy R2 and R3. The models incorporate interactions at multiple levels of detail and are remarkably consistent. Also, reported costs are lower than running a detailed model for the entire system. However, these models satisfy R1 partially

because the multiple models are executed one after another. Therefore, multi-representation interactions are assumed to be independent of one another.

The integrated models for the reactions under study are MREs. Although experts in molecular modelling strive for better correlation between the MM and MO models, the high level of consistency already achieved suggests that the integrated approach is very well-suited for applications involving models at different representation levels.

A.9 Multi-Level Computer Games

A number of commercial computer games present a player with multiple views of the world inhabited by the characters controlled by the player. In games[†] like *Civilization*®, *WarCraft*®, *SimCity*®, *Doom*®, *Heretic*®, *Hexen*®, *Quake*® and *Duke Nukem*®, a player may view the playing area at multiple levels of resolution. In some games, the player may control characters at any resolution, while in others, the player may control characters only at the highest resolution level, with the game pausing when the player switches to a lower resolution level.

Multi-level games satisfy R2 but not R1. Merely displaying information at multiple resolutions amounts to processing read interactions that do not change the representations. Even games that permit changes to be made at either representation rarely permit concurrent changes, or concurrent interactions, thereby completely avoiding the hardest

---

[†] *Civilization* is a registered trademark of Sid Meier games. *WarCraft* is a registered trademark of Blizzard. *SimCity* is a registered trademark of Maxis. *Doom*, *Heretic*, *Hexen* and *Quake* are registered trademarks of id Software. *Duke Nukem* is a registered trademark of 3D Realms Entertainment.

problem in MRM. Most games adopt the approach of selective viewing, wherein all processing takes place at the highest resolution level. The player may request high-resolution information or may ask for low-resolution information. In the latter case, high-resolution information is aggregated and presented as low-resolution information. Selective viewing violates R3.

MREs for entities within such games would incorporate the representation at each resolution level. Players could be permitted to interact at any resolution level, and in the case of multi-player games, at multiple resolutions concurrently. Mapping functions that translate changes to one resolution level to changes to other resolution levels will keep the multiple resolution levels consistent.

A.10 Battlefield Simulations

In the domain of battlefield simulations that are used for training as well as analysis, MRM relates to resolving conceptual and representational differences arising from multiple levels of resolution in simulations that are joined for a common objective, particularly where the simulations were designed and implemented independently. The crux of the problem can be appreciated by considering what is required to simulate accurately an object *and* its constituents concurrently. For example, the abstraction *convoy* may have attributes such as position, velocity, orientation and state of repair. At a more detailed level, the convoy may be viewed as trucks that have attributes such as position, velocity, orientation, state of repair, fuel level, gross weight, carrying capacity and number of occupants. If the convoy abstraction and its constituent trucks are modelled

concurrently, all interactions with the convoy abstraction and its constituents in overlapping periods of time must be reflected accurately at both levels.

Many battlefield simulations satisfy none of R1, R2 or R3 fully. Typically, battlefield simulations employ aggregation-disaggregation to force entities to interact at the same resolution. Aggregation-disaggregation can preclude concurrent multi-representation interaction, can give rise to inconsistencies, and incur high resource costs. MREs for battlefield simulations would incorporate multiple representations of the same object. Typically, the object would be a hierarchical unit such as a corps, division or platoon.

What your actual solution is is unimportant as long as it has Quality.
— *Robert Pirsig,* Zen and the Art of Motorcycle Maintenance

Appendix B

Joint Task Force Prototype

We demonstrate how designers can employ UNIFY and Object Model Template (OMT) to achieve effective Multi-Representation Modelling (MRM). We incorporate UNIFY in Joint Task Force prototype (JTFp) [JTFP97], a military model that is part of the Department of Defence's High Level Architecture (HLA). JTFp is specified using OMT [OMT98]. From the JTFp specifications, we construct an MRE and show how to maintain consistency within this MRE when concurrent interactions occur.

We construct a Platoon-Tanks Multiple Representation Entity (MRE) from the JTFp specifications. We assume that the jointly-executing models in JTFp are a Platoon model and a Tank model. For brevity, we assume that a Platoon consists of only two Tanks, as shown in Figure 66. From the OMT tables in the JTFp specification, we determine the attributes in the representations of the Platoon and Tank models. Next, we capture the relationships among attributes using an Attribute Dependency Graph (ADG) and select mapping functions to maintain consistency in a Platoon-Tanks MRE. Finally, we select policies for resolving the effects of concurrent interactions.

In §B.1, we present the tables in OMT. In §B.2, we list steps for incorporating `UNIFY` in JTFp. We demonstrate each step in subsequent sections. In §B.3, we construct an MRE. In §B.4 and §B.5, we construct an ADG and select mapping functions for attribute dependencies in the MRE. In §B.6 and §B.7, we determine and resolve the effects of concurrent interactions. In §B.8, we construct a CE and IR for the MRE.

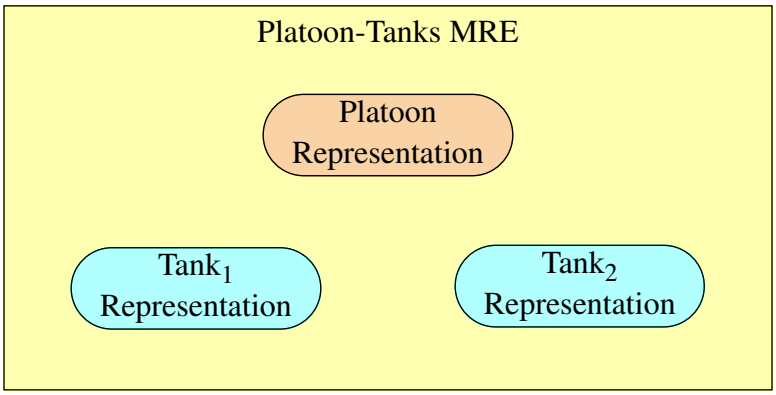


FIGURE 66: Platoon-Tanks MRE

B.1 OMT Tables

OMT consists of a number of tables for specifying parts of a model. They are:

1. Object Class Structure Table (OCST): Shows the class hierarchy along with publishable/subscribable information for each class.
2. Attribute/Parameter Table (APT): Lists object attributes and interaction parameters along their data type, cardinality, units, resolution, accuracy, accuracy condition, update type and update condition.
3. Object Interaction Table (OIT): Lists each possible interaction and associated information, such as its sender, its receiver and the attributes it affects.
4. Enumerated Data Table (EDT): Lists the values of all enumerations.
5. Complex Data Table (CDT): Lists the definitions of all structured data types.

6. Object Class Definitions (OCD): Describes the role of each entity.
7. Object Interaction Definitions (OID): Describes each interaction.
8. Attribute/Parameter Definitions (APD): Describes each object attribute and interaction parameter.

We augment the OIT with the class of each interaction. Also, we add two tables to OMT to capture attribute relationships and specify policies for concurrent interactions.

9. Attribute Relationships Table (ART): Lists each attribute dependency, its type, its mapping function and requirements and properties of the mapping function.
10. Concurrent Interactions Table (CIT): Lists policies for resolving classes and instances of concurrent interactions.

B.2 Steps

The steps for incorporating `UNIFY` in JTFp are:

1. Construct an MRE from the OCST and the APT
2. Construct an ADG from the APT and the ART
3. Select Mapping Functions for Dependencies in the ART
4. Determine the Effects of Interactions from the OIT
5. Resolve the Effects of Concurrent Interactions from the CIT
6. Construct a Consistency Enforcer and an Interaction Resolver

B.3 Construct an MRE from the OCST and the APT

We construct a Platoon-Tanks MRE to execute a Platoon model and a Tank model jointly. Using the OCST for JTFp (shown in Table 14), we derive a Platoon from

AggregateGroundPlayer, and a Tank from MobileGroundPlayer. Our Platoon-Tanks MRE consists of the representations of a Platoon and two Tanks, $Tank_1$ and $Tank_2$. The PS (publishable/subscribable) information associated with each class in Table 14 is used to manage data transfer within the HLA. `UNIFY` does not require this information.

TABLE 14: Object Class Structure Table for JTFp

Base Class	1st Subclass	2nd Subclass
Player (S)	AirPlayer (S)	BallisticMissile (PS)
		Aircraft (PS)
		Flight (PS)
	GroundPlayer (S)	FixedSite (PS)
		MobileGroundPlayer (PS)
		AggregateGroundPlayer (PS)
	AfloatPlayer (PS)	
Environment	Atmosphere (PS)	
	SurfaceCover (PS)	
	OpenWater (PS)	
FederateStatus (PS)		

From the APT, we determine the attributes that are part of the concurrent representations within our Platoon-Tanks MREs. For brevity, Table 15 shows only part of the APT for JTFp. The table lists attributes only for classes or base classes of Platoon and Tank. For each attribute, the designer may specify information such as its data type, units, resolution, accuracy, condition under which the specified accuracy is required and update type. The T/A and U/R information is not used in `UNIFY`.

From the OCST (Table 14) and APT (Table 15), we derive the attributes of a Tank and a Platoon. Table 16 lists the attributes of Platoon, $Tank_1$ and $Tank_2$. For brevity, we combine a number of attributes derived from the OCST and APT (second column) into

TABLE 15: Attribute/Parameter Table for JTFp

Object/ Interaction	Attribute/Parameter	Datatype	Cardinality	Units	Resolution	Accuracy	Accuracy Condition	Update Type	Update Condition	T/A	U/R
Player	entity_name	string	1			perfect	always	static		N	UR
	federate_id	enumeration	1			perfect	always	static		N	UR
	affiliation	enumeration	1			perfect	always	static		N	UR
	motion_type	enumeration	1			perfect	always	static		N	UR
	voice_nets	boolean	maximum		TRUE, FALSE	perfect	always	static		N	UR
	jtids_nets	boolean	maximum		TRUE, FALSE	perfect	always	static		N	UR
	trap_tre	boolean	1		TRUE, FALSE	perfect	always	static		N	UR
	commander_type	enumeration	1			perfect	always	static		N	UR
MobileGroundPlayer	radar_cross_section	float	1	meters	0.1 meters^2	0.1 meters^2	always	static		N	UR
	radar_detectable	boolean	1		TRUE, FALSE	perfect	always	conditional		N	UR
	elint_detectable	boolean	1		TRUE, FALSE	perfect	always	conditional		N	UR
	comint_detectable	boolean	1		TRUE, FALSE	perfect	always	conditional		N	UR
	ir_detectable	boolean	1		TRUE, FALSE	perfect	always	conditional		N	UR
	photint_detectable	boolean	1		TRUE, FALSE	perfect	always	conditional		N	UR
	air_to_air_engageable	boolean	1		TRUE, FALSE	perfect	always	static		N	UR
	air_to_surf_engageable	boolean	1		TRUE, FALSE	perfect	always	static		N	UR
	surf_to_air_engageable	boolean	1		TRUE, FALSE	perfect	always	static		N	UR
	surf_to_surf_engageable	boolean	1		TRUE, FALSE	perfect	always	static		N	UR
	damage_state	float	1	percent	0.01	0.01	always	conditional		N	UR
	entity_type	enumeration	1			perfect	always	static		N	UR
	time_at_last_cse_change	float	1	seconds	0.1 second	0.1 seconds	always	conditional		TA	UR
	lat_at_last_cse_change	float	1	degrees	1×10^{-5} degrees	1×10^{-5} degrees	always	conditional		TA	UR
	lng_at_last_cse_change	float	1	degrees	1×10^{-5} degrees	1×10^{-5} degrees	always	conditional		TA	UR
	alt_at_last_cse_change	float	1	meters	1 meter	1 meter	always	conditional		TA	UR
	cse_at_last_cse_change	float	1	degrees	1×10^{-5} degrees	1×10^{-5} degrees	always	conditional		TA	UR
	hspd_at_last_cse_change	float	1	meters/second	1 meter/second	1 meter/second	always	conditional		TA	UR
	vspd_at_last_cse_change	float	1	meters/second	1 meter/second	1 meter/second	always	conditional		TA	UR
	role	enumeration	1			perfect	always	static		N	UR

TABLE 15: Attribute/Parameter Table for JTFp

Object/ Interaction	Attribute/Parameter	Datatype	Cardinality	Units	Resolution	Accuracy	Accuracy Condition	Update Type	Update Condition	T/A	U/R
AggregateGroundPlayer	radar_cross_section	float	unbounded	meters	0.1 meters^2	0.1 meters^2	always	static		N	UR
	radar_detectable	boolean	unbounded		TRUE, FALSE	perfect	always	conditional		N	UR
	elint_detectable	boolean	unbounded		TRUE, FALSE	perfect	always	conditional		N	UR
	comint_detectable	boolean	unbounded		TRUE, FALSE	perfect	always	conditional		N	UR
	ir_detectable	boolean	unbounded		TRUE, FALSE	perfect	always	conditional		N	UR
	photint_detectable	boolean	unbounded		TRUE, FALSE	perfect	always	conditional		N	UR
	air_to_air_engageable	boolean	unbounded		TRUE, FALSE	perfect	always	static		N	UR
	air_to_surf_engageable	boolean	unbounded		TRUE, FALSE	perfect	always	static		N	UR
	surf_to_air_engageable	boolean	unbounded		TRUE, FALSE	perfect	always	static		N	UR
	surf_to_surf_engageable	boolean	unbounded		TRUE, FALSE	perfect	always	static		N	UR
	composition	enumeration	unbounded			perfect	always	conditional		N	UR
	time_at_last_cse_change	float	1	seconds	0.1 second	0.1 seconds	always	conditional		TA	UR
	lat_at_last_cse_change	float	1	degrees	1×10^{-5} degrees	1×10^{-5} degrees	always	conditional		TA	UR
	lng_at_last_cse_change	float	1	degrees	1×10^{-5} degrees	1×10^{-5} degrees	always	conditional		TA	UR
	alt_at_last_cse_change	float	1	meters	1 meter	1 meter	always	conditional		TA	UR
	cse_at_last_cse_change	float	1	degrees	1×10^{-5} degrees	1×10^{-5} degrees	always	conditional		TA	UR
	hspd_at_last_cse_change	float	1	meters/second	1 meter/second	1 meter/second	always	conditional		TA	UR
	vspd_at_last_cse_change	float	1	meters/second	1 meter/second	1 meter/second	always	conditional		TA	UR
	orientation	float	1	degrees	0.1 degree	perfect	always	conditional		N	UR
	depth	float	1	meters	1 meter	perfect	always	conditional		N	UR
	front	float	1	meters	1 meter	perfect	always	conditional		N	UR

one attribute (fourth column). We combine attributes that are logically similar and that have identical accuracy condition, update type and update condition. For example, we combine the attributes radar_detectable, elint_detectable, comint_detectable, ir_detectable and photint_detectable into an attribute called detectable. Likewise, we combine entity_name, federate_id, affiliation, motion_type, voice_nets, jtids_nets, trap_tre and commander_type into an attribute called initial_parameters. We combine such attributes so that we can present a simple MRE, for which an ADG will be presentable and specifying mapping functions will be manageable. Combining similar attributes is consistent with our discussion about assigning nodes of an ADG (§6.1.1). A node can be assigned to any subset of a representation for which a designer can specify how the effects of interactions must be applied. In practice, we expect designers to assign nodes to individual attributes rather than combined attributes.

TABLE 16: Attributes of Platoon, $Tank_1$ and $Tank_2$ (JTFp)

Entity	Original Attributes	Derived From	New Attributes
Platoon	entity_name	Player	initial_parameters
	federate_id		
	affiliation		
	motion_type		
	voice_nets		
	jtids_net		
	trap_tre		
	commander_type		
	<none specified>	GroundPlayer	
	radar_cross_section	AggregateGroundPlayer	radar_cross_section
	radar_detectable		detectable
	elint_detectable		
	comint_detectable		
	ir_detectable		
	photoint_detectable		
	air_to_air_engageable		engageable
	air_to_surf_engageable		
	surf_to_air_engageable		
	surf_to_surf_engageable		
	composition		composition
	time_at_last_cse_change		last_cse_change
	lat_at_last_cse_change		
	lng_at_last_cse_change		
	alt_at_last_cse_change		
	cse_at_last_cse_change		
	hspd_at_last_cse_change		
	vspd_at_last_cse_change		
	depth		depth
	front		front
	orientation		orientation

TABLE 16: Attributes of Platoon, $Tank_1$ and $Tank_2$ (JTFp)

Entity	Original Attributes	Derived From	New Attributes
$Tank_1$	entity_name	Player	$initial_parameters_1$
	federate_id		
	affiliation		
	motion_type		
	voice_nets		
	jtids_net		
	trap_tre		
	commander_type		
	<none specified>	GroundPlayer	
	radar_cross_section	MobileGroundPlayer	$radar_cross_section_1$
	radar_detectable		$detectable_1$
	elint_detectable		
	comint_detectable		
	ir_detectable		
	photoint_detectable		
	air_to_air_engageable		$engageable_1$
	air_to_surf_engageable		
	surf_to_air_engageable		
	surf_to_surf_engageable		
	damage_state		$damage_state_1$
	entity_type		$entity_type_1$
	time_at_last_cse_change		$last_cse_change_1$
	lat_at_last_cse_change		
	lng_at_last_cse_change		
	alt_at_last_cse_change		
	cse_at_last_cse_change		
	hspd_at_last_cse_change		
	vspd_at_last_cse_change		
	role		$role_1$

TABLE 16: Attributes of Platoon, $Tank_1$ and $Tank_2$ (JTFp)

Entity	Original Attributes	Derived From	New Attributes
$Tank_2$	entity_name	Player	$initial_parameters_2$
	federate_id		
	affiliation		
	motion_type		
	voice_nets		
	jtids_net		
	trap_tre		
	commander_type		
	<none specified>	GroundPlayer	
	radar_cross_section	MobileGroundPlayer	$radar_cross_section_2$
	radar_detectable		$detectable_2$
	elint_detectable		
	comint_detectable		
	ir_detectable		
	photoint_detectable		
	air_to_air_engageable		$engageable_2$
	air_to_surf_engageable		
	surf_to_air_engageable		
	surf_to_surf_engageable		
	damage_state		$damage_state_2$
	entity_type		$entity_type_2$
	time_at_last_cse_change		$last_cse_change_2$
	lat_at_last_cse_change		
	lng_at_last_cse_change		
	alt_at_last_cse_change		
	cse_at_last_cse_change		
	hspd_at_last_cse_change		
	vspd_at_last_cse_change		
	role		$role_2$

B.4 Construct an ADG from the APT and the ART

We construct an ADG for the Platoon-Tanks MRE from the APT and the ART for JTFp. Since OMT does not support specifying relationships, we construct an example

ART for our MRE (Table 17). In practice, we expect a designer to construct an ART specific to the models executed jointly. The specification of the relationship may be accomplished formally; in Table 17, we present informal specifications in the last column.

TABLE 17: Attribute Relationship Table for Platoon-Tanks MRE in JTFp

Dependency	Type	Specification
$\text{detectable}_1 \rightarrow \text{detectable}$	Cumulative	If even one tank is detectable, the entire platoon is detectable. If the platoon is detectable, each tank is detectable.
$\text{detectable}_2 \rightarrow \text{detectable}$	Cumulative	
$\text{detectable} \rightarrow \text{detectable}_1$	Distributive	
$\text{detectable} \rightarrow \text{detectable}_2$	Distributive	
$\text{engageable}_1 \rightarrow \text{engageable}$	Cumulative	If even one tank is engageable, the platoon is engageable. If the platoon is engageable, at least one tank must be engageable.
$\text{engageable}_2 \rightarrow \text{engageable}$	Cumulative	
$\text{engageable} \rightarrow \text{engageable}_1$	Distributive	
$\text{engageable} \rightarrow \text{engageable}_2$	Distributive	
$\text{damage_state}_1 \rightarrow \text{composition}$	Cumulative	If a damage_state becomes 100%, composition reduces by one, and *vice versa*.
$\text{damage_state}_2 \rightarrow \text{composition}$	Cumulative	
$\text{composition} \rightarrow \text{damage_state}_1$	Distributive	
$\text{composition} \rightarrow \text{damage_state}_2$	Distributive	
$\text{last_cse_change}_1 \rightarrow \text{last_cse_change}$	Cumulative	Elements of the course, such as altitude, velocity and position, are vector quantities.
$\text{last_cse_change}_2 \rightarrow \text{last_cse_change}$	Cumulative	
$\text{last_cse_change} \rightarrow \text{last_cse_change}_1$	Distributive	
$\text{last_cse_change} \rightarrow \text{last_cse_change}_2$	Distributive	
$\text{radar_cross_section}_1 \rightarrow \text{radar_cross_section}$	Cumulative	The radar cross-section of the platoon encompasses the radar cross-section of its tanks.
$\text{radar_cross_section}_2 \rightarrow \text{radar_cross_section}$	Cumulative	
$\text{radar_cross_section} \rightarrow \text{radar_cross_section}_1$	Distributive	
$\text{radar_cross_section} \rightarrow \text{radar_cross_section}_2$	Distributive	
$\text{composition} \rightarrow \text{depth}$	Modelling	The composition affects the depth, front line and orientation of the platoon.
$\text{composition} \rightarrow \text{front}$	Modelling	
$\text{composition} \rightarrow \text{orientation}$	Modelling	

We construct an ADG for the Platoon-Tanks MRE. From Table 16, which was derived from the APT, we determine the nodes in the ADG. From the ART in Table 17, we determine the arcs in the ADG. The ADG is shown in Figure 67. The interaction

dependencies to each attribute exist because interactions with other entities or internal actions of the MRE may change any attribute.

Dynamic semantics of attribute relationships may be captured by weighting dependencies. Dependency classes capture static semantics, whereas weights capture dynamic semantics. For our Platoon-Tanks MRE, we assign a weight of one to each cumulative dependency, and equal weights to distributive dependencies that have the same independent attribute. We select these weights in order to keep our subsequent discussion of mapping functions simple. Other weights for these dependencies are possible.

B.5 Select Mapping Functions for Dependencies in the ART

We select mapping functions to translate attributes among concurrent representations within the Platoon-Tanks MRE. Recall from Chapter 6 that mapping functions must translate values or changes in values of attributes from one to another. Additionally, it is desirable that mapping functions complete their translations in a time-bound manner, and that they be composable and reversible.

We show mapping functions for some dependencies in Table 18. The mapping functions are presented as pseudo-code. Error-checking has been omitted for brevity. Pseudo-code in the second column of Table 18 implements specifications in the last column of Table 17. If any Tank is detectable, Platoon is detectable. Likewise, if Platoon is detectable, all Tanks are detectable. Platoon is not detectable only if both Tanks are not detectable. If any Tank is engageable, Platoon is engageable. If Platoon is engageable, at least one Tank is engageable. Platoon is not engageable only if both Tanks are not

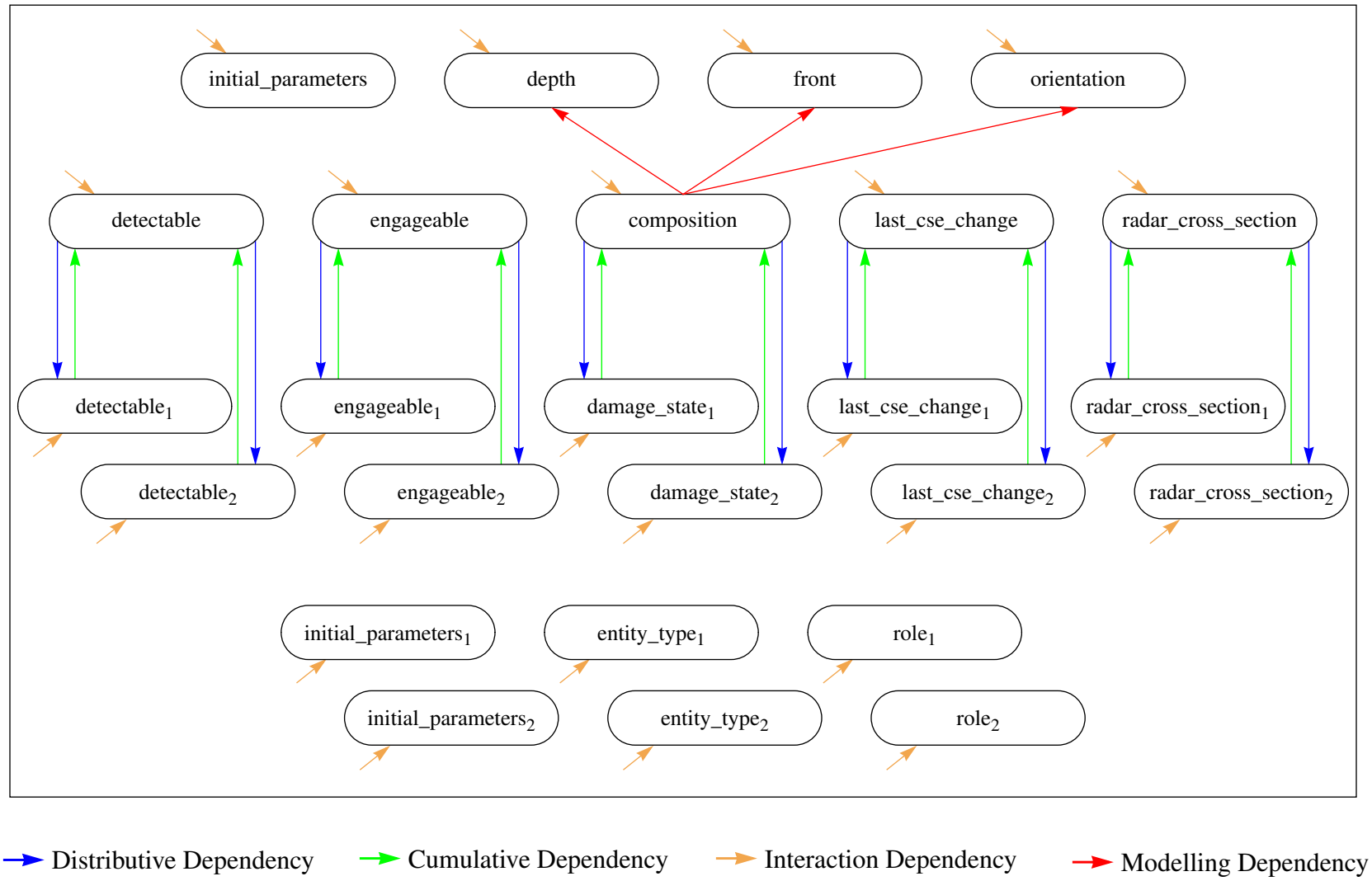


FIGURE 67: ADG for the JTFp Platoon-Tanks MRE

engageable. If the damage_state of any Tank becomes 100%, the composition of the Platoon is reduced by one. The damage_state of the Tank is changed to ∞ to ensure that composition is not reduced further subsequently. Likewise, if composition is reduced by one (δcomposition $= -1$), a Tank whose damage_state was less than 100% previously is selected and its damage_state changed to 100%. Similarly, mapping functions for other dependencies can be constructed. For the last_cse_change attribute, a designer may employ different functions for the different parts, such as lat_at_last_cse_change, time_at_last_cse_change and hspd_at_last_cse_change. For example, the Platoon-level position, consisting of lat_at_last_cse_change, lng_at_last_cse_change and alt_at_last_cse_change may be defined as the centroid of the Tank-level positions. However, the Platoon-level time, time_at_last_cse_change, may be defined as the latest of the Tank-level times. Mapping functions such as those shown in Table 18 translate values or changes in values of attributes.

TABLE 18: Mapping Functions for JTFp Platoon-Tanks MRE

Dependency	Mapping Function
$\text{detectable}_1 \rightarrow \text{detectable}$	$\text{detectable} \leftarrow f_d(\text{detectable}_1, \text{detectable}_2)$ f_d: $\text{detectable} \leftarrow \text{detectable}_1 \vee \text{detectable}_2$
$\text{detectable}_2 \rightarrow \text{detectable}$	
$\text{detectable} \rightarrow \text{detectable}_1$	$(\text{detectable}_1, \text{detectable}_2) \leftarrow g_d(\text{detectable})$ g_d: $\text{detectable}_1 \leftarrow \text{detectable}_2 \leftarrow \text{detectable}$
$\text{detectable} \rightarrow \text{detectable}_2$	
$\text{engageable}_1 \rightarrow \text{engageable}$	$\text{engageable} \leftarrow f_e(\text{engageable}_1, \text{engageable}_2)$ f_e: $\text{engageable} \leftarrow \text{engageable}_1 \vee \text{engageable}_2$
$\text{engageable}_2 \rightarrow \text{engageable}$	
$\text{engageable} \rightarrow \text{engageable}_1$	$(\text{engageable}_1, \text{engageable}_2) \leftarrow g_e(\text{engageable})$ g_e: $\text{engageable}_{\text{random}(1,\,2)} \leftarrow \text{engageable}$
$\text{engageable} \rightarrow \text{engageable}_2$	
$\text{damage_state}_1 \rightarrow$ composition	composition $\leftarrow f_c(\text{damage_state}_1, \text{damage_state}_2)$ f_c: for (i $\leftarrow$ 1 to 2) if ($\text{damage_state}_i = 100\%$) { composition––; $\text{damage_state}_i \leftarrow \infty$ }
$\text{damage_state}_2 \rightarrow$ composition	

TABLE 18: Mapping Functions for JTFp Platoon-Tanks MRE

Dependency	Mapping Function
composition → damage_state$_1$	(damage_state$_1$, damage_state$_2$) ← g_c(composition) g_c: if (δcomposition = −1) if (damage_state$_1$ < 100%) damage_state$_1$ ← ∞ elsif (damage_state$_2$ < 100%) damage_state$_2$ ← ∞
composition → damage_state$_2$	
...	

The mapping functions shown in Table 18 are composable and reversible. Moreover, since they are simple in construction, we expect that they will complete in a time-bound manner, thus ensuring that the Platoon-Tanks MRE is consistent at all observation times. When an interaction changes the value of any attribute, mapping functions propagate the change in the attribute to dependent attributes. For example, if an interaction changes the Tank-level attribute, detectable$_1$, the mapping function f_d changes the dependent Platoon-level attribute, detectable. Subsequently, the mapping function g_d changes the Tank-level attribute, detectable$_2$. Since f_d and g_d are composable, the change to detectable$_1$ eventually propagates to detectable$_2$. Since f_d and g_d are reversible, detectable$_1$ does not change again as a result of the same interaction.

When an interaction occurs, traversing the ADG in Figure 67 and applying the mapping functions in Table 18 ensures that the Platoon-Tanks MRE is consistent at all observation times. Next, we determine and resolve the effects of concurrent interactions.

B.6 Determine the Effects of Interactions from the OIT

We determine the effects of interactions on the Platoon-Tanks MRE from the OIT. We show an augmented OIT in Table 19. The first column lists the name of the interaction. The next four columns list the class and affected attributes for the sender and receiver of

the interaction. We augment each interaction in the OIT with its type (see Chapter 7): Type 0 (certain responses), Type 1 (uncertain responses), Type 2 (certain requests), and Type 3 (uncertain requests). We do not utilise the ISR (Init/Sense/React) information and the parameters of an interaction in UNIFY.

The OIT lists interactions among entities, but not internal actions of an entity. For example, the OIT does not list any interaction corresponding to our Platoon-Tanks MRE changing its course, because such an interaction is internal to the MRE. In UNIFY, internal actions are interactions. We add an internal action called ChangeCourse to the interactions in the OIT (see last row in Table 19) to show that UNIFY addresses internal actions as well as interactions with other entities. This interaction initiates a change in the course of an entity. The sender and receiver of ChangeCourse is the same entity. The class of that entity is Player. The interaction affects the attribute last_cse_change.

The last column in Table 19 lists the type of an interaction. Assigning a type requires information about the semantics of an interaction. In OMT, this information is available from the OID. For example, the OID lists the semantics of GetSeaState as a request that will be satisfied by an Environment entity. Hence GetSeaState is a Type 2 interaction. ReturnSeaState is the response to a GetSeaState. ReturnSeaState could be Type 0 or Type 1, but we assigned it to Type 1 because an entity may discard an update about the state of the sea. For the ChangeCourse interaction, we assumed that a change in the course of an entity is a request whose outcome is uncertain.

TABLE 19: Object Interaction Table for JTFp

Interaction	Sender Class	Sender Attributes	Receiver	Receiver Attributes	Interaction Parameters	ISR	Type
TBMWarning	Player	none	Player	none	send_time, comms_system, net_number	IR	1
TBMLaunchAlert	Player	none	Player	none	send_time, comms_system, net_number, launch_lat, launch_lng	IR	1

TABLE 19: Object Interaction Table for JTFp

Interaction	Sender Class	Sender Attributes	Receiver	Receiver Attributes	Interaction Parameters	ISR	Type
InitiateStrikeCommand	Player	none	Player	none	send_time, comms_system, net_number, strike_phase_name, strike_phase_number	IR	3
DetectionReport	Player	none	Player	none	send_time, comms_system, net_number, report_type, entity_id, reported_lat, reported_lng, reported_alt, reported_cse, reported_hspd, reported_vspd, reported_affiliation, reported_type, reported_raid_count, reported_damage	IR	1
RequestAirSupport	Player	none	Player	none	send_time, comms_system, net_number, requestor_id, target_id, time_on_target, target_lat, target_lng	IR	3
SituationReport	Player	none	Player	none	gfc_lat, gfc_lng, rel_to_objective, objective_name, personnel_status, equipment_status, effectiveness_status, combat_intensity	IR	1
AirToDiscreteGroundEngage	AirPlayer	none	MobileGroundPlayer	damage_state	launch_time, time_of_flight, launch_lat, launch_lng, launch_alt, weapon_type, salvo_size, aimpoint, estimated_pk_at_launch	IR	0
AirToAggregateGroundEngage	AirPlayer	none	AggregateGroundPlayer	composition	launch_time, time_of_flight, launch_lat, launch_lng, launch_alt, weapon_type, salvo_size, targeted_systems, estimated_pks_at_launch	IR	0
DiscreteGroundToAirEngage	MobileGroundPlayer	none	AirPlayer	damage_state	launch_time, time_of_flight, launch_lat, launch_lng, launch_alt, weapon_type, salvo_size, estimated_pk_at_launch	IR	0
AggregateGroundToAirEngage	AggregateGroundPlayer	none	AirPlayer	damage_state,c omposition	launch_time, time_of_flight launch_quadrant, launch_offsets, weapon_systems, ammo_types, salvo_sizes estimated_pks_at_launch	IR	0
DiscreteGroundToGroundEngage	MobileGroundPlayer	none	MobileGroundPlayer	damage_state	launch_time, time_of_flight, aim_pt_lat, aim_pt_lng, weapon_type, estimated_pk_at_launch	IR	0
TroopsHitBeach	Player	none			num_boat_sorties, num_helo_sorties, lat_of_beach_location. lng_of_beach_location	IR	0
GetLOSVisibility	Player	none	Environment	none	observation_time, sensor_lat, sensor_lng, sensor_alt, target_lat, target_lng, target_alt	IR	2
ReturnLOSVisbility	Environment	none	Player	none	LOS_visibility, relative_humidity, reason, return_id	IR	1
GetAtmosphericCondition	Player	none	Environment	none	time, observation_lat, observation_lng	IR	2
ReturnAtmosphericCondition	Environment	none	Player	none	ceiling, surface_temperature, surface_pressure, visibility, relative_humidity, total_cloud_cover, cloud1_type, cloud1_height, cloud1_amount, cloud2_type, cloud2_height, cloud2_amount, cloud3_type, cloud3_height, cloud3_amount, surface_wind_speed, surface_wind_direction, precipitation_amount, artificial_obscurants, natural_obscurants	IR	1
GetSeaState	Player	none	Environment	none	lat, lng	IR	2
ReturnSeaState	Environment	none	Player	none	state_of_sea, sea_surface_temp	IR	1
ChangeCourse	Player	last_cse_change	Player	last_cse_change	new_lat, new_lng, new_alt, new_hspd, new_vspd	IR	3

We determine the interactions that our Platoon-Tanks MRE can send and receive. In Table 20, we list the interactions that Platoon, Tank_1 and Tank_2 can send and receive. In

the first column, we list the name of an interaction as the name in the OIT along with a suffix that indicates whether Platoon, $Tank_1$ or $Tank_2$ sends or receives that interaction. For example, the interaction GetLOSVisibility can be sent by an entity of class Player. Since Player is a base class of Platoon, $Tank_1$ and $Tank_2$, we distinguish the interaction GetLOSVisibility sent by these three entities as GetLOSVisibility-P, GetLOSVisibility-T_1 and GetLOSVisibility-T_2 respectively. In the second column, we indicate whether the Platoon-Tanks MRE sends (S) or receives (R) the interaction. In the third column, we list the attributes affected by the interaction directly, i.e., we list the set *affects* for the interaction. These attributes are determined from the OIT. In the fourth column, we list the attributes affected by the interaction indirectly, i.e., we list the set $affects^+$ for the interaction. These attributes can be determined from the ADG in Figure 67. Finally, we indicate the type of the interaction.

TABLE 20: Effects of Interactions for JTFp Platoon-Tanks MRE

Interaction	S/R	*affects*	$affects^+$	Type
TBMWarning-P	S			1
TBMWarning-T_1	S			1
TBMWarning-T_2	S			1
TBMLaunchAlert-P	S			1
TBMLaunchAlert-T_1	S			1
TBMLaunchAlert-T_2	S			1
InitiateStrikeCommand-P	S			3
InitiateStrikeCommand-T_1	S			3
InitiateStrikeCommand-T_2	S			3
DetectionReport-P	S			1
DetectionReport-T_1	S			1
DetectionReport-T_2	S			1
RequestAirSupport-P	S			3

TABLE 20: Effects of Interactions for JTFp Platoon-Tanks MRE

Interaction	S/R	*affects*	*affects*$^+$	Type
RequestAirSupport-T_1	S			3
RequestAirSupport-T_2	S			3
SituationReport-P	S			1
SituationReport-T_1	S			1
SituationReport-T_2	S			1
AggregateGroundToAirEngage-P	S			0
DiscreteGroundToAirEngage-T_1	S			0
DiscreteGroundToAirEngage-T_2	S			0
DiscreteGroundToGroundEngage-T_1	S			0
DiscreteGroundToGroundEngage-T_2	S			0
TroopsHitBeach-P	S			0
TroopsHitBeach-T_1	S			0
TroopsHitBeach-T_2	S			0
GetLOSVisibility-P	S			2
GetLOSVisibility-T_1	S			2
GetLOSVisibility-T_2	S			2
GetAtmosphericCondition-P	S			2
GetAtmosphericCondition-T_1	S			2
GetAtmosphericCondition-T_2	S			2
GetSeaState-P	S			2
GetSeaState-T_1	S			2
GetSeaState-T_2	S			2
ChangeCourse-P	S	last_cse_change	last_cse_change$_1$, last_cse_change$_2$, last_cse_change	3
ChangeCourse-T_1	S	last_cse_change$_1$	last_cse_change, last_cse_change$_2$, last_cse_change$_1$	3

TABLE 20: Effects of Interactions for JTFp Platoon-Tanks MRE

Interaction	S/R	*affects*	*affects*$^+$	Type
ChangeCourse-T_2	S	last_cse_change$_2$	last_cse_change, last_cse_change$_1$, last_cse_change$_2$	3
TBMWarning-P	R			1
TBMWarning-T_1	R			1
TBMWarning-T_2	R			1
TBMLaunchAlert-P	R			1
TBMLaunchAlert-T_1	R			1
TBMLaunchAlert-T_2	R			1
InitiateStrikeCommand-P	R			3
InitiateStrikeCommand-T_1	R			3
InitiateStrikeCommand-T_2	R			3
DetectionReport-P	R			1
DetectionReport-T_1	R			1
DetectionReport-T_2	R			1
RequestAirSupport-P	R			3
RequestAirSupport-T_1	R			3
RequestAirSupport-T_2	R			3
SituationReport-P	R			1
SituationReport-T_1	R			1
SituationReport-T_2	R			1
AirToAggregateGroundEngage-P	R	composition	damage_state$_1$, damage_state$_2$, depth, front, orientation, composition	0
AirToDiscreteGroundEngage-T_1	R	damage_state$_1$	composition, damage_state$_2$, depth, front, orientation, damage_state$_1$	0

TABLE 20: Effects of Interactions for JTFp Platoon-Tanks MRE

Interaction	S/R	*affects*	*affects*$^+$	Type
AirToDiscreteGroundEngage-T_2	R	damage_state$_2$	composition, damage_state$_1$, depth, front, orientation, damage_state$_2$	0
ReturnLOSVisibility-P	R			1
ReturnLOSVisibility-T_1	R			1
ReturnLOSVisibility-T_2	R			1
ReturnAtmosphericCondition-P	R			1
ReturnAtmosphericCondition-T_1	R			1
ReturnAtmosphericCondition-T_2	R			1
ReturnSeaState-P	R			1
ReturnSeaState-T_1	R			1
ReturnSeaState-T_2	R			1
ChangeCourse-P	R	last_cse_change	last_cse_change$_1$, last_cse_change$_2$, last_cse_change	3
ChangeCourse-T_1	R	last_cse_change$_1$	last_cse_change, last_cse_change$_2$, last_cse_change$_1$	3
ChangeCourse-T_2	R	last_cse_change$_2$	last_cse_change, last_cse_change$_1$, last_cse_change$_2$	3

Any subset of the interactions in Table 20 may occur concurrently. Next, we show how to resolve the effects of concurrent interactions.

B.7 Resolve the Effects of Concurrent Interactions from the CIT

The effects of concurrent interactions can be resolved by implementing polices from the CIT. In practice, a designer constructs a CIT specific to the application. Since a CIT is unavailable in OMT, we construct an example CIT, shown in Table 21.

A designer specifies policies in the CIT for resolving the effects of concurrent interactions. The CIT consists of sets of concurrent interactions with dependent effects, policies for resolving them and conditions under which the policies are applicable. Concurrent interactions that are independent of one another can be resolved by serialization and are not specified in the CIT. Some interactions may be independent because they affect disjoint sets of attributes. Other interactions may be independent because their effects are applied in different time-steps, for example, interactions sent and received by an entity. Yet other interactions are independent because they are request-response pairs. Policies must be specified in the CIT for only the remaining interactions. Policies may be specified for classes of interactions (e.g., the last two rows in Table 21) or for instances of interactions (e.g., all the other rows in Table 21). In JTFp, many interactions do not affect any attributes. Although such interactions can be assumed independent, we do not make such an assumption. It is likely that the interactions affect internal attributes in the models. Since OMT is meant to be an interface specification, internal attributes are not listed in the APT. For consistency maintenance, a designer must list internal attributes as well in the APT. Since internal attributes are not listed, we will not assume that interactions that affect disjoint sets of attributes are independent. For

example, although InitiateStrikeCommand-P, InitiateStrikeCommand-T_1 and InitiateStrikeCommand-T_2 affect no attributes, hence affecting disjoint sets of attributes, we specify policies for resolving these interactions. An Interaction Resolver for the Platoon-Tanks MRE applies the policies in the CIT only if the effects of concurrent interactions conflict. If concurrent interactions do not conflict, they may be serialized.

TABLE 21: Concurrent Interactions Table for JTFp Platoon-Tanks MRE

Concurrent Interactions	Condition	Policy
AggregateGroundToAirEngage-P, any combination of (DiscreteGroundToAirEngage-T_1, DiscreteGroundToAirEngage-T_2, DiscreteGroundToGroundEngage-T_1, DiscreteGroundToGroundEngage-T_2)	All sent	Do not send all except AggregateGroundToAirEngage-P
DiscreteGroundToAirEngage-T_i, DiscreteGroundToGroundEngage-T_i	All sent	Do not send DiscreteGroundToAirEngage-T_i
InitiateStrikeCommand-P, any combination of (InitiateStrikeCommand-T_1, InitiateStrikeCommand-T_2)	All received	Delay all except InitiateStrikeCommand-P by one time-step
DetectionReport-P, any combination of (DetectionReport-T_1, DetectionReport-T_2)	All received	Ignore DetectionReport-P
RequestAirSupport-P, any combination of (RequestAirSupport-T_1, RequestAirSupport-T_2)	All received	Delay all except RequestAirSupport-P by one time-step
SituationReport-P, any combination of (SituationReport-T_1, SituationReport-T_2)	All received	Ignore SituationReport-P
AirToAggregateGroundEngage-P, AirToDiscreteGroundEngage-T_i	All received	Damage to $Tank_i$ less than sum of damages but greater than minimum of damages; add compensatory interaction to reduce damage
ReturnLOSVisibility-P, any combination of (ReturnLOSVisibility-T_1, ReturnLOSVisibility-T_2)	All received	Ignore ReturnLOSVisibility-P

TABLE 21: Concurrent Interactions Table for JTFp Platoon-Tanks MRE

Concurrent Interactions	Condition	Policy
ReturnAtmosphericCondition-P, any combination of (ReturnAtmosphericCondition-T_1, ReturnAtmosphericCondition-T_2)	All received	Ignore ReturnAtmosphericCondition-P
ReturnSeaState-P, any combination of (ReturnSeaState-T_1, ReturnSeaState-T_2)	All received	Ignore ReturnSeaState-P
ChangeCourse-P, any combination of (ChangeCourse-T_1, ChangeCourse-T_2)	All received	Ignore all except ChangeCourse-P
Type 0, Type 1	All received	Ignore Type 1
Type 2, Type 3	All received	Ignore Type 3
Any Interaction	Ignored or Delayed	Ignored or Delayed entirely, i.e., no partial effects permitted

B.8 Construct a Consistency Enforcer and an Interaction Resolver

A Consistency Enforcer (CE) and an Interaction Resolver (IR) for an MRE maintain consistency and resolve concurrent interactions respectively. A CE consists of an ADG and mapping functions, whereas an IR consists of policies for resolving concurrent interactions. Figure 68 shows a JTFp Platoon-Tanks MRE. The MRE can interact at multiple representation levels — the Platoon and Tank levels — concurrently. Moreover, the concurrent representations within the MRE are consistent at all observation times.

A CE consists of an ADG and application-specific mapping functions. For the Platoon-Tanks MRE, we presented an ADG in Figure 67 and mapping functions in Table 18. In Figure 34 (see Chapter 6), we presented an algorithm for implementing a CE.

In §6.3, we discussed how to traverse an ADG and apply mapping functions in order to keep an MRE internally consistent.

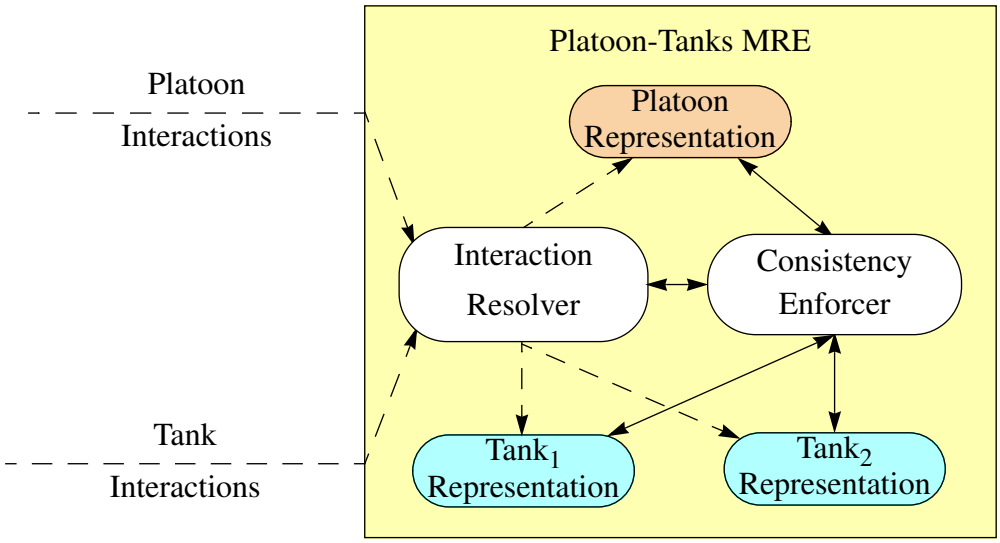


FIGURE 68: JTFp Platoon-Tanks MRE

An IR consists of application-specific policies for resolving the effects of concurrent interactions. For the Platoon-Tanks MRE, we presented policies for resolving concurrent interactions in Table 21. In Figure 47 (see Chapter 7), we presented an algorithm for implementing an IR. In §7.5, we presented a taxonomy for classifying interactions. Using this taxonomy, we presented policies for resolving the effects of concurrent interactions.

A CE and an IR ensure that an MRE is internally consistent when concurrent interactions occur. During a time-step, a number of concurrent interactions may occur. The IR determines the type of each interaction. Next, the IR applies the effect of each interaction as if the interaction occurred in isolation. In order to do so, the IR permits the interactions to take effect one at a time. When an interaction changes an attribute, the CE traverses an ADG and translates changes to dependent attributes by invoking the appropriate mapping functions. The CE maintains a list of changes for each attribute as a result of computing the effects of each interaction. Subsequently, the CE applies the

effects of all the interactions on each attribute. The CE queries the IR about policies to resolve the effects of dependent concurrent interactions whenever the CE detects conflicts in the list of changes for an entity. If the IR contains a policy for resolving conflicting changes, the CE applies the changes accordingly; otherwise, the CE assumes the changes are independent and applies them in an arbitrary order. When the changes to all attributes have been applied, the MRE is internally consistent.

"My dear Watson, try a little analysis yourself,"
said he, with a touch of impatience.
"You know my methods. Apply them,
and it will be instructive to compare results."
— Arthur Conan Doyle, The Sign of the Four

Appendix C

Joint Precision Strike Demonstration

We demonstrate how designers can employ `UNIFY` and Object Model Template (OMT) to achieve effective Multi-Representation Modelling (MRM). We incorporate `UNIFY` in Joint Precision Strike Demonstration (JPSD) [JPSD97], a military model that is part of the Department of Defence's High Level Architecture (HLA). JPSD is specified using OMT [OMT98]. From the JPSD specifications, we construct an MRE and show how to maintain consistency within this MRE when concurrent interactions occur.

We construct a Platoon-Tanks Multiple Representation Entity (MRE) from the JPSD specifications. We assume that the jointly-executing models in JPSD are a Platoon model and a Tank model. For brevity, we assume that a Platoon consists of only two Tanks, as shown in Figure 69. From the OMT tables in the JPSD specification, we determine the attributes in the representations of the Platoon and Tank models. Next, we capture the relationships among attributes using an Attribute Dependency Graph (ADG) and select mapping functions to maintain consistency in a Platoon-Tanks MRE. Finally, we select policies for resolving the effects of concurrent interactions.

In §C.1, we present the tables in OMT. In §C.2, we list steps for incorporating `UNIFY` in JPSD. We demonstrate each step in subsequent sections. In §C.3, we construct an MRE. In §C.4 and §C.5, we construct an ADG and select mapping functions for attribute dependencies in the MRE. In §C.6 and §C.7, we determine and resolve the effects of concurrent interactions. In §C.8, we construct a CE and IR for the MRE.

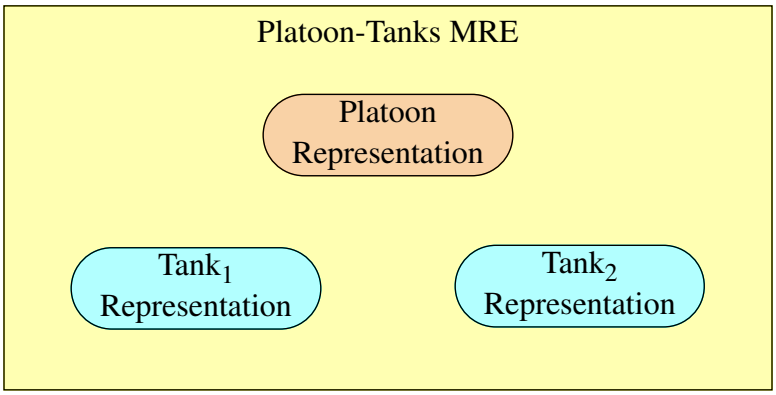


FIGURE 69: Platoon-Tanks MRE

C.1 OMT Tables

OMT consists of a number of tables for specifying parts of a model. They are:

1. Object Class Structure Table (OCST): Shows the class hierarchy along with publishable/subscribable information for each class.
2. Attribute/Parameter Table (APT): Lists object attributes and interaction parameters along their data type, cardinality, units, resolution, accuracy, accuracy condition, update type and update condition.
3. Object Interaction Table (OIT): Lists each possible interaction and associated information, such as its sender, its receiver and the attributes it affects.
4. Enumerated Data Table (EDT): Lists the values of all enumerations.
5. Complex Data Table (CDT): Lists the definitions of all structured data types.

6. Object Class Definitions (OCD): Describes the role of each entity.
7. Object Interaction Definitions (OID): Describes each interaction.
8. Attribute/Parameter Definitions (APD): Describes each object attribute and interaction parameter.

We augment the OIT with the class of each interaction. Also, we add two tables to OMT to capture attribute relationships and specify policies for concurrent interactions.

9. Attribute Relationships Table (ART): Lists each attribute dependency, its type, its mapping function and requirements and properties of the mapping function.
10. Concurrent Interactions Table (CIT): Lists policies for resolving classes and instances of concurrent interactions.

C.2 Steps

The steps for incorporating `UNIFY` in JPSD are:

1. Construct an MRE from the OCST and the APT
2. Construct an ADG from the APT and the ART
3. Select Mapping Functions for Dependencies in the ART
4. Determine the Effects of Interactions from the OIT
5. Resolve the Effects of Concurrent Interactions from the CIT
6. Construct a Consistency Enforcer and an Interaction Resolver

C.3 Construct an MRE from the OCST and the APT

We construct a Platoon-Tanks MRE to execute a Platoon model and a Tank model jointly. We modify the OCST for JPSD to make Aggregate a derived class of Entity so that

an Aggregate entity can send and receive other interactions in addition to requests to aggregate and disaggregate. Also, we do not show specific instances of derived classes, such as Tank or Aggregate. From the modified OCST for JPSD (shown in Table 22), we derive a Platoon from Aggregate. Our Platoon-Tanks MRE consists of the representations of a Platoon and two Tanks, $Tank_1$ and $Tank_2$.

TABLE 22: Object Class Structure Table for JPSD

Base Class	1st Subclass	2nd Subclass	3rd Subclass
Entity	Aggregate		
	Platform	Land	Tank
			ArmoredFightingVehicle
			SelfPropelledArtillery
			SmallWheeledUtilityVehicle
		Air	AttackHelicopter
			ElectronicWarfare
			UAV
	Munition	AntiArmor	Guided
		BattlefieldSupport	
System	TacticalSystem		
	Strike		
	BattalionCommander		
	ModSafCommander		

From the APT, we determine the attributes that are part of the concurrent representations within our Platoon-Tanks MREs. For brevity, Table 23 shows only part of the APT for JPSD. The table lists attributes only for classes or base classes of Platoon and Tank. For each attribute, the designer may specify information such as its data type, units,

resolution, accuracy, condition under which the specified accuracy is required and update type. The T/A and U/R information is not used in UNIFY.

TABLE 23: Attribute/Parameter Table for JPSD

Object/ Interaction	Attribute/Parameter	Datatype	Cardinality	Units	Resolution	Accuracy	Accuracy Condition	Update Type	Update Condition	T/A	U/R
Entity	Entity_ID_site	short		enumeration	discrete	perfect	always	static			UR
	Entity_ID_application	short		enumeration	discrete	perfect	always	static			UR
	Entity_ID_entity	short		enumeration	discrete	perfect	always	static			UR
	Force_ID	short		enumeration	discrete	perfect	always	static			UR
	Entity_Type_Kind	short		enumeration	discrete	perfect	always	static			UR
	Entity_Type_Domain	short		enumeration	discrete	perfect	always	static			UR
	Entity_Type_Country	short		enumeration	discrete	perfect	always	static			UR
	Entity_Type_Category	short		enumeration	discrete	perfect	always	static			UR
	Entity_Type_Subcategory	short		enumeration	discrete	perfect	always	static			UR
	Entity_Type_Specific	short		enumeration	discrete	perfect	always	static			UR
	Location_X	double		meters	1	10%	DR*	conditional	time-out†	TA	UR
	Location_Y	double		meters	1	10%	DR	conditional	time-out	TA	UR
	Location_Z	double		meters	1	10%	DR	conditional	time-out	TA	UR
	Velocity_X	double		meters/sec		10%	DR	conditional	time-out	TA	UR
	Velocity_Y	double		meters/sec		10%	DR	conditional	time-out	TA	UR
	Velocity_Z	double		meters/sec		10%	DR	conditional	time-out	TA	UR
	Orientation_Psi	double		radians		3 degrees	DR	conditional	time-out	TA	UR
	Orientation_Theta	double		radians		3 degrees	DR	conditional	time-out	TA	UR
	Orientation_Phi	double		radians		3 degrees	DR	conditional	time-out	TA	UR
	marking_text	string				perfect	always	static			UR

TABLE 23: Attribute/Parameter Table for JPSD

Object/ Interaction	Attribute/Parameter	Datatype	Cardinality	Units	Resolution	Accuracy	Accuracy Condition	Update Type	Update Condition	T/A	U/R
Aggregate	Aggregate_ID_site	short		enumeration	discrete	perfect	always	static			UR
	Aggregate_ID_application	short		enumeration	discrete	perfect	always	static			UR
	Aggregate_ID_entity	short		enumeration	discrete	perfect	always	static			UR
	Entity_Type_Kind	short		enumeration	discrete	perfect	always	static			UR
	Entity_Type_Domain	short		enumeration	discrete	perfect	always	static			UR
	Entity_Type_Country	short		enumeration	discrete	perfect	always	static			UR
	Entity_Type_Category	short		enumeration	discrete	perfect	always	static			UR
	Entity_Type_Subcategory	short		enumeration	discrete	perfect	always	static			UR
	Entity_Type_Specific	short		enumeration	discrete	perfect	always	static			UR
	Location_X	double		meters	1	10%	DR	periodic	0.033333333	TA	UR
	Location_Y	double		meters	1	10%	DR	periodic	0.033333333	TA	UR
	Location_Z	double		meters	1	10%	DR	periodic	0.033333333	TA	UR
	Velocity_X	double		meters/sec		10%	DR	periodic	0.033333333	TA	UR
	Velocity_Y	double		meters/sec		10%	DR	periodic	0.033333333	TA	UR
	Velocity_Z	double		meters/sec		10%	DR	periodic	0.033333333	TA	UR
	Orientation_Psi	double		radians		3 degrees	DR	periodic	0.033333333	TA	UR
	Orientation_Theta	double		radians		3 degrees	DR	periodic	0.033333333	TA	UR
	Orientation_Phi	double		radians		3 degrees	DR	periodic	0.033333333	TA	UR
	marking_text	string				perfect	always	static			UR
	Shape	short		enumeration	discrete	perfect	always	conditional	if tasking changes set shape		UR
	Num_Entities_in_Aggregate	short		enumeration	discrete	perfect	always	delta			UR
	DisaggPermitted	boolean			discrete	perfect	always	static			UR
	AggregateState	short		enumeration	discrete	perfect	always	delta			UR
	SubordinateList	sequence			discrete	perfect	always	delta			UR
Platform	Appearance_Paint_Scheme	short		enumeration	discrete	perfect	always	delta			UR
	Appearance_Smoking	short		enumeration	discrete	perfect	always	delta			UR
	Appearance_Flaming	short		enumeration	discrete	perfect	always	delta			UR
	Appearance_Trailing	short		enumeration	discrete	perfect	always	delta			UR
	Appearance_Lights	short		enumeration	discrete	perfect	always	delta			UR
	Appearance_Hatch	short		enumeration	discrete	perfect	always	delta			UR
	Damage_State_Appearance	short		enumeration	discrete	perfect	always	delta			UR
	Damage_State_Mobility	short		enumeration	discrete	perfect	always	delta			UR
	Damage_State_Fire_Power	short		enumeration	discrete	perfect	always	delta			UR
Tank	GunElevation	double		radians		0.1	DR	delta		TA	UR

* DR refers to a dead-reckoning algorithm, listed in the JPSD APT as DR(F, P, W).

† time-out refers to the JPSD APT condition: if (!accurate) or (value has changed and 5 second update interval passed)

From the OCST (Table 22) and APT (Table 23), we derive the attributes of a Tank and a Platoon. Table 24 lists the attributes of Platoon, Tank_1 and Tank_2. For brevity, we combine a number of attributes derived from the OCST and APT (second column) into

one attribute (fourth column). We combine attributes that are logically similar and that have identical accuracy condition, update type and update condition. For example, we combine the attributes Location_X, Location_Y, and Location_Z into an attribute called Location. Likewise, we combine Entity_ID_site, Entity_ID_application, Entity_ID_entity, Entity_Type_Kind, Entity_Type_Domain, Entity_Type_Country, Entity_Type_Category, Entity_Type_Subcategory, Entity_Type_Specific and marking_text into an attribute called Initial_Parameters. We combine such attributes so that we can present a simple MRE, for which an ADG will be presentable and specifying mapping functions will be manageable. Combining similar attributes is consistent with our discussion about assigning nodes of an ADG (§6.1.1). A node can be assigned to any subset of a representation for which a designer can specify how the effects of interactions must be applied. In practice, we expect designers to assign nodes to individual attributes rather than combined attributes.

C.4 Construct an ADG from the APT and the ART

We construct an ADG for the Platoon-Tanks MRE from the APT and the ART for JPSD. Since OMT does not support specifying relationships, we construct an example ART for our MRE (Table 25). In practice, we expect a designer to construct an ART specific to the models executed jointly. The specification of the relationship may be accomplished formally; in Table 25, we present informal specifications in the last column.

We construct an ADG for the Platoon-Tanks MRE. From Table 24, which was derived from the APT, we determine the nodes in the ADG. From the ART in Table 25, we determine the arcs in the ADG. The ADG is shown in Figure 70. The interaction

TABLE 24: Attributes of Platoon, $Tank_1$ and $Tank_2$ (JPSD)

Entity	Original Attributes	Derived From	New Attributes
Platoon	Aggregate_ID_site	Aggregate	Initial_Parameters
	Aggregate_ID_application		
	Aggregate_ID_entity		
	Entity_Type_Kind		
	Entity_Type_Domain		
	Entity_Type_Country		
	Entity_Type_Category		
	Entity_Type_Subcategory		
	Entity_Type_Specific		
	marking_text		
	Location_X		Location
	Location_Y		
	Location_Z		
	Velocity_X		Velocity
	Velocity_Y		
	Velocity_Z		
	Orientation_Psi		Orientation
	Orientation_Theta		
	Orientation_Phi		
	Shape		Composition
	Num_Entities_in_Aggregate		
	DisaggPermitted		
	AggregateState		
	SubordinateList		

TABLE 24: Attributes of Platoon, $Tank_1$ and $Tank_2$ (JPSD)

Entity	Original Attributes	Derived From	New Attributes
$Tank_1$	Entity_ID_site	Entity	$Initial_Parameters_1$
	Entity_ID_application		
	Entity_ID_entity		
	Force_ID		
	Entity_Type_Kind		
	Entity_Type_Domain		
	Entity_Type_Country		
	Entity_Type_Category		
	Entity_Type_Subcategory		
	Entity_Type_Specific		
	marking_text		
	Location_X		$Location_1$
	Location_Y		
	Location_Z		
	Velocity_X		$Velocity_1$
	Velocity_Y		
	Velocity_Z		
	Orientation_Psi		$Orientation_1$
	Orientation_Theta		
	Orientation_Phi		
	Appearance_Paint_Scheme	Platform	$Appearance_1$
	Appearance_Smoking		
	Appearance_Flaming		
	Appearance_Trailing		
	Appearance_Lights		
	Appearance_Hatch		
	Damage_State_Appearance		$Damage_State_1$
	Damage_State_Mobility		
	Damage_State_Fire_Power		
	GunElevation	Tank	$GunElevation_1$

TABLE 24: Attributes of Platoon, $Tank_1$ and $Tank_2$ (JPSD)

Entity	Original Attributes	Derived From	New Attributes
$Tank_2$	Entity_ID_site	Entity	$Initial_Parameters_2$
	Entity_ID_application		
	Entity_ID_entity		
	Force_ID		
	Entity_Type_Kind		
	Entity_Type_Domain		
	Entity_Type_Country		
	Entity_Type_Category		
	Entity_Type_Subcategory		
	Entity_Type_Specific		
	marking_text		
	Location_X		$Location_2$
	Location_Y		
	Location_Z		
	Velocity_X		$Velocity_2$
	Velocity_Y		
	Velocity_Z		
	Orientation_Psi		$Orientation_2$
	Orientation_Theta		
	Orientation_Phi		
	Appearance_Paint_Scheme	Platform	$Appearance_2$
	Appearance_Smoking		
	Appearance_Flaming		
	Appearance_Trailing		
	Appearance_Lights		
	Appearance_Hatch		
	Damage_State_Appearance		$Damage_State_2$
	Damage_State_Mobility		
	Damage_State_Fire_Power		
	GunElevation	Tank	$GunElevation_2$

dependencies to each attribute exist because interactions with other entities or internal actions of the MRE may change any attribute.

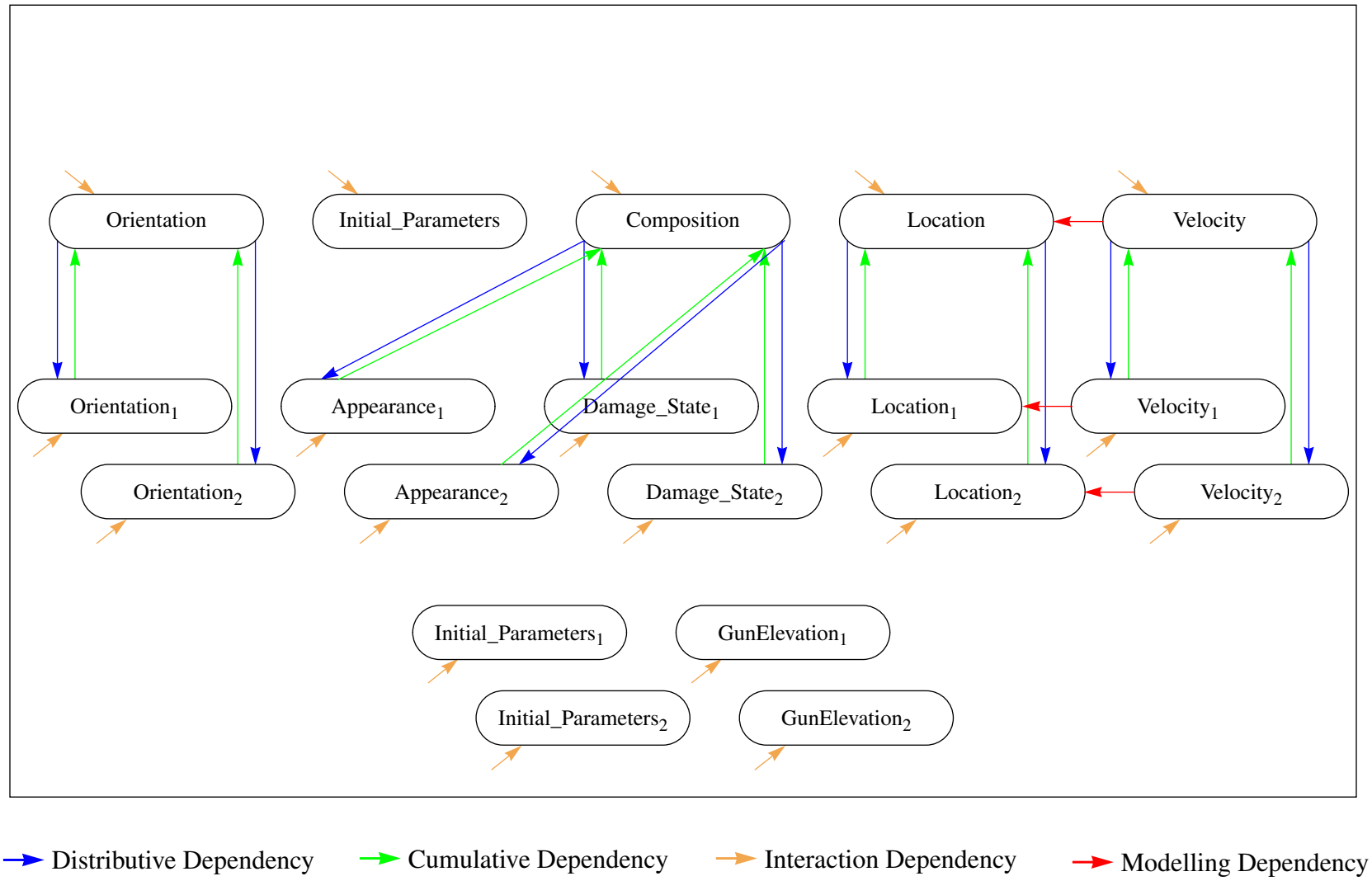


FIGURE 70: ADG for the JPSD Platoon-Tanks MRE

TABLE 25: Attribute Relationship Table for Platoon-Tanks MRE in JPSD

Dependency	Type	Specification
$\text{Location}_1 \rightarrow \text{Location}$	Cumulative	The location of the platoon is the centroid of the location of its tanks.
$\text{Location}_2 \rightarrow \text{Location}$	Cumulative	
$\text{Location} \rightarrow \text{Location}_1$	Distributive	
$\text{Location} \rightarrow \text{Location}_2$	Distributive	
$\text{Velocity}_1 \rightarrow \text{Velocity}$	Cumulative	The velocity of the platoon is the average of the velocity of its tanks.
$\text{Velocity}_2 \rightarrow \text{Velocity}$	Cumulative	
$\text{Velocity} \rightarrow \text{Velocity}_1$	Distributive	
$\text{Velocity} \rightarrow \text{Velocity}_2$	Distributive	
$\text{Orientation}_1 \rightarrow \text{Orientation}$	Cumulative	The orientation of the platoon is the average of the orientations of its tanks.
$\text{Orientation}_2 \rightarrow \text{Orientation}$	Cumulative	
$\text{Orientation} \rightarrow \text{Orientation}_1$	Distributive	
$\text{Orientation} \rightarrow \text{Orientation}_2$	Distributive	
$\text{Damage_State}_1 \rightarrow \text{Composition}$	Cumulative	If a tank is fatally damaged, Composition reduces by one, and *vice versa*.
$\text{Damage_State}_2 \rightarrow \text{Composition}$	Cumulative	
$\text{Composition} \rightarrow \text{Damage_State}_1$	Distributive	
$\text{Composition} \rightarrow \text{Damage_State}_2$	Distributive	
$\text{Appearance}_1 \rightarrow \text{Composition}$	Cumulative	The appearance of each tank determines the appearance of the platoon.
$\text{Appearance}_2 \rightarrow \text{Composition}$	Cumulative	
$\text{Composition} \rightarrow \text{Appearance}_1$	Distributive	
$\text{Composition} \rightarrow \text{Appearance}_2$	Distributive	
$\text{Velocity} \rightarrow \text{Location}$	Modelling	The location of a platoon or a tank depends on its velocity.
$\text{Velocity}_1 \rightarrow \text{Location}_1$	Modelling	
$\text{Velocity}_2 \rightarrow \text{Location}_2$	Modelling	

Dynamic semantics of attribute relationships may be captured by weighting dependencies. Dependency classes capture static semantics, whereas weights capture dynamic semantics. For our Platoon-Tanks MRE, we assign a weight of one to each cumulative dependency, and equal weights to distributive dependencies that have the same independent attribute. We select these weights in order to keep our subsequent discussion of mapping functions simple. Other weights for these dependencies are possible.

C.5 Select Mapping Functions for Dependencies in the ART

We select mapping functions to translate attributes among concurrent representations within the Platoon-Tanks MRE. Recall from Chapter 6 that mapping functions must translate values or changes in values of attributes from one to another. Additionally, it is desirable that mapping functions complete their translations in a time-bound manner, and that they be composable and reversible.

We show mapping functions for some dependencies in Table 26. The mapping functions are presented as pseudo-code. Error-checking has been omitted for brevity. Pseudo-code in the second column of Table 26 implements specifications in the last column of Table 25. The location, velocity and orientation of Platoon are averages of the location, velocity and orientation of $Tank_1$ and $Tank_2$. Similarly, mapping functions for other dependencies can be constructed. Mapping functions such as those shown in Table 26 translate values or changes in values of attributes.

TABLE 26: Mapping Functions for JPSD Platoon-Tanks MRE

Dependency	Mapping Function
$Location_1 \rightarrow Location$	$Location \leftarrow f_d(Location_1, Location_2)$
$Location_2 \rightarrow Location$	f_l: $Location_X \leftarrow (Location_1_X + Location_2_X) / 2$ $Location_Y \leftarrow (Location_1_Y + Location_2_Y) / 2$ $Location_Z \leftarrow (Location_1_Z + Location_2_Z) / 2$
$Location \rightarrow Location_1$	$(Location_1, Location_2) \leftarrow g_d(Location)$
$Location \rightarrow Location_2$	g_l: $\delta Location_1_X \leftarrow \delta Location_2_X \leftarrow \delta Location_X$ $\delta Location_1_Y \leftarrow \delta Location_2_Y \leftarrow \delta Location_Y$ $\delta Location_1_Z \leftarrow \delta Location_2_Z \leftarrow \delta Location_Z$
$Velocity_1 \rightarrow Velocity$	$Velocity \leftarrow f_d(Velocity_1, Velocity_2)$
$Velocity_2 \rightarrow Velocity$	f_v: $Velocity_X \leftarrow (Velocity_1_X + Velocity_2_X) / 2$ $Velocity_Y \leftarrow (Velocity_1_Y + Velocity_2_Y) / 2$ $Velocity_Z \leftarrow (Velocity_1_Z + Velocity_2_Z) / 2$

TABLE 26: Mapping Functions for JPSD Platoon-Tanks MRE

Dependency	Mapping Function
Velocity $\rightarrow$ Velocity$_1$	(Velocity$_1$, Velocity$_2$) $\leftarrow g_d$(Velocity) g_v: δVelocity$_1$_X $\leftarrow$ δVelocity$_2$_X $\leftarrow$ δVelocity_X δVelocity$_1$_Y $\leftarrow$ δVelocity$_2$_Y $\leftarrow$ δVelocity_Y δVelocity$_1$_Z $\leftarrow$ δVelocity$_2$_Z $\leftarrow$ δVelocity_Z
Velocity $\rightarrow$ Velocity$_2$	
Orientation$_1$ $\rightarrow$ Orientation	Orientation $\leftarrow f_d$(Orientation$_1$, Orientation$_2$) f_o: Orientation_Psi $\leftarrow$ (Orientation$_1$_Psi + Orientation$_2$_Psi) / 2 Orientation_Theta $\leftarrow$ (Orientation$_1$_Theta + Orientation$_2$_Theta) / 2 Orientation_Phi $\leftarrow$ (Orientation$_1$_Phi + Orientation$_2$_Phi) / 2
Orientation$_2$ $\rightarrow$ Orientation	
Orientation $\rightarrow$ Orientation$_1$	(Orientation$_1$, Orientation$_2$) $\leftarrow g_d$(Orientation) g_o: δOrientation$_1$_Psi $\leftarrow$ δOrientation$_2$_Psi $\leftarrow$ δOrientation_Psi δOrientation$_1$_Theta $\leftarrow$ δOrientation$_2$_Theta $\leftarrow$ δOrientation_Theta δOrientation$_1$_Phi $\leftarrow$ δOrientation$_2$_Phi $\leftarrow$ δOrientation_Phi
Orientation $\rightarrow$ Orientation$_2$	
...	

The mapping functions shown in Table 26 are composable and reversible. Moreover, since they are simple in construction, we expect that they will complete in a time-bound manner, thus ensuring that the Platoon-Tanks MRE is consistent at all observation times. When an interaction changes the value of any attribute, mapping functions propagate the change in the attribute to dependent attributes. For example, if an interaction changes the Tank-level attribute, Orientation$_1$, the mapping function f_o changes the dependent Platoon-level attribute, Orientation. Subsequently, the mapping function g_o changes the Tank-level attribute, Orientation$_2$. Since f_o and g_o are composable, the change to Orientation$_1$ eventually propagates to Orientation$_2$. Since f_o and g_o are reversible, Orientation$_1$ does not change again as a result of the same interaction.

When an interaction occurs, traversing the ADG in Figure 70 and applying the mapping functions in Table 26 ensures that the Platoon-Tanks MRE is consistent at all observation times. Next, we determine and resolve the effects of concurrent interactions.

C.6 Determine the Effects of Interactions from the OIT

We determine the effects of interactions on the Platoon-Tanks MRE from the OIT. We show an augmented OIT in Table 27. The first column lists the name of the interaction. The next four columns list the class and affected attributes for the sender and receiver of the interaction. We augment each interaction in the OIT with its type (see Chapter 7): Type 0 (certain responses), Type 1 (uncertain responses), Type 2 (certain requests), and Type 3 (uncertain requests). We do not utilise the ISR (Init/Sense/React) information and the parameters of an interaction in `UNIFY`.

The OIT lists interactions among entities, but not internal actions of an entity. For example, the OIT does not list any interaction corresponding to our Platoon-Tanks MRE changing its course, because such an interaction is internal to the MRE. In `UNIFY`, internal actions are interactions. We add an internal action called ChangeCourse to the interactions in the OIT (see last row in Table 27) to show that `UNIFY` addresses internal actions as well as interactions with other entities. This interaction initiates a change in the course of an entity. The sender and receiver of ChangeCourse is the same entity. The class of that entity is Entity. The interaction affects the attributes Location, Velocity and Orientation.

The last column in Table 27 lists the type of an interaction. Assigning a type requires information about the semantics of an interaction. For example, the semantics of Collision are that it is generated in response to a modelling event in which two entities collide. Since

the collision has occurred already and its effects on the sender and receiver are certain, Collision is a Type 0 interaction. ArtyRadioMessage is a request by a commanding officer to perform a task. Since an entity may discard the request, ArtyRadioMessage is a Type 3 interaction. For the ChangeCourse interaction, we assumed that a change in the course of an entity is a request whose outcome is uncertain.

TABLE 27: Object Interaction Table for JPSD

Interaction	Sender Class	Sender Attributes	Receiver	Receiver Attributes	Interaction Parameters	ISR	Type
Collision	Entity	Location, Velocity, Orientation, Appearance, Damage_State	Entity	Location, Velocity, Orientation, Appearance, Damage_State	Issuing_ID, Colliding_ID, Mass, Relative_Location, Event_ID, Velocity	ISR	0
Detonation	Munition	Appearance, Damage_State	Entity	Velocity, Appearance, Damage_State	Munition_ID, Location, Velocity, Firing_ID, Target_ID ,Event_ID, Detonation_Result, Burst_Descriptor	ISR	0
Weapon_Launch	Platform	none	Munition	none	Launch_Platform_ID, Weapon_ID	ISR	3
DisaggregateRequest	Aggregate	none	Aggregate	AggregateState, Subordinates	entity_ID, aggregate_ID, detection_range, aggregate_state	ISR	3
ArtyRadioMessage	ModSafCommander	none	Land	Location, Orientation, Velocity	message_type, command, gun_ID, full_message	ISR	3
ChangeCourse	Entity	Location, Velocity, Orientation	Entity	Location, Velocity, Orientation	New_Location, New_Velocity, New_Orientation	IR	3

We determine the interactions that our Platoon-Tanks MRE can send and receive. In Table 28, we list the interactions that Platoon, Tank_1 and Tank_2 can send and receive. In the first column, we list the name of an interaction as the name in the OIT along with a suffix that indicates whether Platoon, Tank_1 or Tank_2 sends or receives that interaction. For example, the interaction Collision can be sent by an entity of class Entity. Since Entity is a base class of Platoon, Tank_1 and Tank_2, we distinguish the interaction Collision sent by these three entities as Collision-P, Collision-T_1 and Collision-T_2 respectively. In the second column, we indicate whether the Platoon-Tanks MRE sends (S) or receives (R) the interaction. In the third column, we list the attributes affected by the interaction directly, i.e., we list the set *affects* for the interaction. These attributes are determined from the OIT.

In the fourth column, we list the attributes affected by the interaction indirectly, i.e., we list the set *affects*$^+$ for the interaction. These attributes can be determined from the ADG in Figure 70. Finally, we indicate the type of the interaction. Since in `UNIFY` we do not aggregate or disaggregate, we do not expect the DisaggregateRequest interaction to occur.

TABLE 28: Effects of Interactions for JPSD Platoon-Tanks MRE

Interaction	S/R	*affects*	*affects*$^+$	Type
Collision-P	S	Location, Velocity, Orientation, Composition	Location_1, Location_2, Velocity_1, Velocity_2, Orientation_1, Orientation_2, Appearance_1, Appearance_2, Damage_State_1, Damage_State_2, Location, Velocity, Orientation, Composition	0
Collision-T_1	S	Location_1, Velocity_1, Orientation_1, Appearance_1, Damage_State_1	Location, Velocity, Orientation, Composition, Location_2, Velocity_2, Orientation_2, Appearance_2, Damage_State_2, Location_1, Velocity_1, Orientation_1, Appearance_1, Damage_State_1	0
Collision-T_2	S	Location_2, Velocity_2, Orientation_2, Appearance_2, Damage_State_2	Location, Velocity, Orientation, Composition, Location_1, Velocity_1, Orientation_1, Appearance_1, Damage_State_1, Location_2, Velocity_2, Orientation_2, Appearance_2, Damage_State_2	0
Weapon_Launch-T_1	S			3
Weapon_Launch-T_2	S			3
DisaggregateRequest-P	S			3
ChangeCourse-P	S	Location, Velocity, Orientation	Location_1, Location_2, Velocity_1, Velocity_2, Orientation_1, Orientation_2, Location, Velocity, Orientation	3

TABLE 28: Effects of Interactions for JPSD Platoon-Tanks MRE

Interaction	S/R	*affects*	*affects*$^+$	Type
ChangeCourse-T_1	S	$Location_1$, $Velocity_1$, $Orientation_1$	Location, Velocity, Orientation, $Location_2$, $Velocity_2$, $Orientation_2$, $Location_1$, $Velocity_1$, $Orientation_1$	3
ChangeCourse-T_2	S	$Location_2$, $Velocity_2$, $Orientation_2$	Location, Velocity, Orientation, $Location_1$, $Velocity_1$, $Orientation_1$, $Location_2$, $Velocity_2$, $Orientation_2$	3
Collision-P	R	Location, Velocity, Orientation, Composition	$Location_1$, $Location_2$, $Velocity_1$, $Velocity_2$, $Orientation_1$, $Orientation_2$, $Appearance_1$, $Appearance_2$, $Damage_State_1$, $Damage_State_2$, Location, Velocity, Orientation, Composition	0
Collision-T_1	R	$Location_1$, $Velocity_1$, $Orientation_1$, $Appearance_1$, $Damage_State_1$	Location, Velocity, Orientation, Composition, $Location_2$, $Velocity_2$, $Orientation_2$, $Appearance_2$, $Damage_State_2$, $Location_1$, $Velocity_1$, $Orientation_1$, $Appearance_1$, $Damage_State_1$	0
Collision-T_2	R	$Location_2$, $Velocity_2$, $Orientation_2$, $Appearance_2$, $Damage_State_2$	Location, Velocity, Orientation, Composition, $Location_1$, $Velocity_1$, $Orientation_1$, $Appearance_1$, $Damage_State_1$, $Location_2$, $Velocity_2$, $Orientation_2$, $Appearance_2$, $Damage_State_2$	0
Detonation-P	R	Velocity, Composition	$Velocity_1$, $Velocity_2$, $Appearance_1$, $Appearance_2$, $Damage_State_1$, $Damage_State_2$, Location, Velocity, Composition, $Location_1$, $Location_2$	0

TABLE 28: Effects of Interactions for JPSD Platoon-Tanks MRE

Interaction	S/R	*affects*	*affects*$^+$	Type
Detonation-T_1	R	$Velocity_1$, $Appearance_1$, $Damage_State_1$	Velocity, Composition, $Location_1$, $Velocity_2$, $Velocity_1$, $Appearance_2$, $Appearance_1$,$Damage_State_2$, $Damage_State_1$, Location, $Location_2$	0
Detonation-T_2	R	$Velocity_2$, $Appearance_2$, $Damage_State_2$	Velocity, Composition, $Location_2$, $Velocity_1$, $Velocity_2$, $Appearance_1$, $Appearance_2$,$Damage_State_1$, $Damage_State_2$, Location, $Location_1$	0
DisaggregateRequest-P	R			3
ArtyRadioMessage-P	R	Location, Velocity, Orientation	$Location_1$, $Location_2$, $Velocity_1$, $Velocity_2$, $Orientation_1$, $Orientation_2$, Location, Velocity, Orientation, Composition	3
ArtyRadioMessage-T_1	R	$Location_1$, $Velocity_1$, $Orientation_1$	Location, Velocity, Orientation, $Location_2$, $Velocity_2$, $Orientation_2$, $Location_1$, $Velocity_1$, $Orientation_1$	3
ArtyRadioMessage-T_2	R	$Location_2$, $Velocity_2$, $Orientation_2$	Location, Velocity, Orientation, $Location_1$, $Velocity_1$, $Orientation_1$, $Location_2$, $Velocity_2$, $Orientation_2$	3
ChangeCourse-P	R	Location, Velocity, Orientation	$Location_1$, $Location_2$, $Velocity_1$, $Velocity_2$, $Orientation_1$, $Orientation_2$, Location, Velocity, Orientation	3
ChangeCourse-T_1	R	$Location_1$, $Velocity_1$, $Orientation_1$	Location, Velocity, Orientation, $Location_2$, $Velocity_2$, $Orientation_2$, $Location_1$, $Velocity_1$, $Orientation_1$	3

TABLE 28: Effects of Interactions for JPSD Platoon-Tanks MRE

Interaction	S/R	*affects*	*affects*$^+$	Type
ChangeCourse-T_2	R	$Location_2$, $Velocity_2$, $Orientation_2$	Location, Velocity, Orientation, $Location_1$, $Velocity_1$, $Orientation_1$, $Location_2$, $Velocity_2$, $Orientation_2$	3

Any subset of the interactions in Table 28 may occur concurrently. Next, we show how to resolve the effects of concurrent interactions.

C.7 Resolve the Effects of Concurrent Interactions from the CIT

The effects of concurrent interactions can be resolved by implementing polices from the CIT. In practice, a designer constructs a CIT specific to the application. Since a CIT is unavailable in OMT, we construct an example CIT, shown in Table 29.

A designer specifies policies in the CIT for resolving the effects of concurrent interactions. The CIT consists of sets of concurrent interactions with dependent effects, policies for resolving them and conditions under which the policies are applicable. Concurrent interactions that are independent of one another can be resolved by serialization and are not specified in the CIT. Some interactions may be independent because they affect disjoint sets of attributes. Other interactions may be independent because their effects are applied in different time-steps, for example, interactions sent and received by an entity. Yet other interactions are independent because they are request-response pairs. Policies must be specified in the CIT for only the remaining interactions. Policies may be specified for classes of interactions (e.g., the last two rows in Table 29) or

for instances of interactions (e.g., all the other rows in Table 29). An Interaction Resolver for the Platoon-Tanks MRE applies the policies in the CIT only if the effects of concurrent interactions conflict. If concurrent interactions do not conflict, they may be serialized.

TABLE 29: Concurrent Interactions Table for JPSD Platoon-Tanks MRE

Concurrent Interactions	Condition	Policy
Any combination of (Detonation-P, Detonation-T_1, Detonation-T_2), any combination of (Collision-P, Collision-T_1, Collision-T_2)	Always	Damage to Tanks less than sum of damages but greater than minimum of damages; add compensatory interaction to reduce damage
DisaggregateRequest-P	Always	Ignore
ArtyRadioMessage-P, any combination of (ArtyRadioMessage-T_1, ArtyRadioMessage-T_2)	Received, commands conflicting	Ignore all except InitiateStrikeCommand-P
ArtyRadioMessage-P, any combination of (ArtyRadioMessage-T_1, ArtyRadioMessage-T_2)	Received, commands non-conflicting	Delay all except InitiateStrikeCommand-P by one time-step
Any combination of (ArtyRadioMessage-P, ArtyRadioMessage-T_1, ArtyRadioMessage-T_2), any combination of (ChangeCourse-P, ChangeCourse-T_1, ChangeCourse-T_2)	All received	Ignore ChangeCourse-P, ChangeCourse-T_1, ChangeCourse-T_2; Resolve ArtyRadioMessage-P, ArtyRadioMessage-T_1, ArtyRadioMessage-T_2) as above
ChangeCourse-P, any combination of (ChangeCourse-T_1, ChangeCourse-T_2)	All received	Ignore all except ChangeCourse-P
Type 0, Type 1	All received	Ignore Type 1
Type 2, Type 3	All received	Ignore Type 3
Any Interaction	Ignored or Delayed	Ignored or Delayed entirely, i.e., no partial effects permitted

C.8 Construct a Consistency Enforcer and an Interaction Resolver

A Consistency Enforcer (CE) and an Interaction Resolver (IR) for an MRE maintain consistency and resolve concurrent interactions respectively. A CE consists of an ADG and mapping functions, whereas an IR consists of policies for resolving concurrent interactions. Figure 71 shows a JPSD Platoon-Tanks MRE. The MRE can interact at multiple representation levels — the Platoon and Tank levels — concurrently. Moreover, the concurrent representations within the MRE are consistent at all observation times.

A CE consists of an ADG and application-specific mapping functions. For the Platoon-Tanks MRE, we presented an ADG in Figure 70 and mapping functions in Table 26. In Figure 34 (see Chapter 6), we presented an algorithm for implementing a CE. In §6.3, we discussed how to traverse an ADG and apply mapping functions in order to keep an MRE internally consistent.

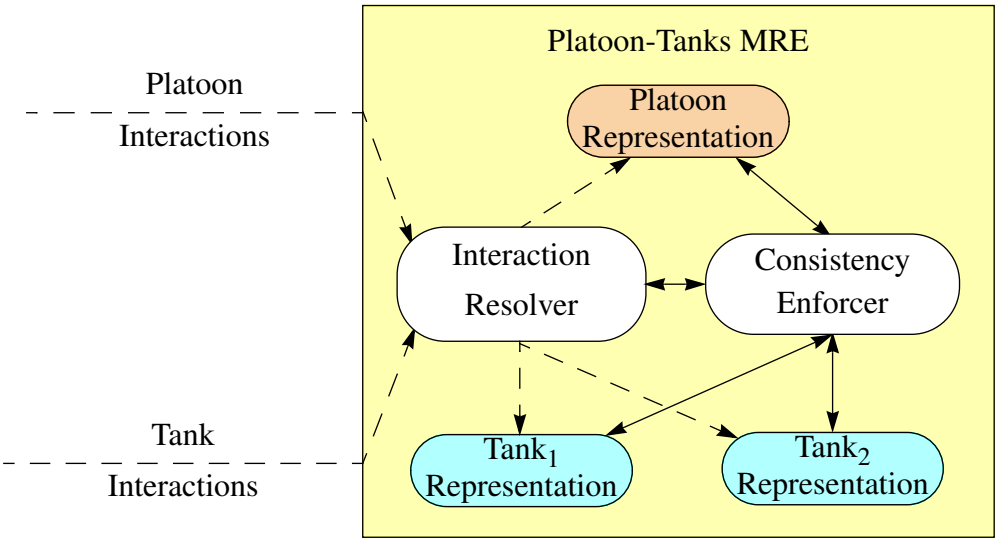


FIGURE 71: JPSD Platoon-Tanks MRE

An IR consists of application-specific policies for resolving the effects of concurrent interactions. For the Platoon-Tanks MRE, we presented policies for resolving concurrent interactions in Table 29. In Figure 47 (see Chapter 7), we presented an algorithm for implementing an IR. In §7.5, we presented a taxonomy for classifying interactions. Using this taxonomy, we presented policies for resolving the effects of concurrent interactions.

A CE and an IR ensure that an MRE is internally consistent when concurrent interactions occur. During a time-step, a number of concurrent interactions may occur. The IR determines the type of each interaction. Next, the IR applies the effect of each interaction as if the interaction occurred in isolation. In order to do so, the IR permits the interactions to take effect one at a time. When an interaction changes an attribute, the CE traverses an ADG and translates changes to dependent attributes by invoking the appropriate mapping functions. The CE maintains a list of changes for each attribute as a result of computing the effects of each interaction. Subsequently, the CE applies the effects of all the interactions on each attribute. The CE queries the IR about policies to resolve the effects of dependent concurrent interactions whenever the CE detects conflicts in the list of changes for an entity. If the IR contains a policy for resolving conflicting changes, the CE applies the changes accordingly; otherwise, the CE assumes the changes are independent and applies them in an arbitrary order. When the changes to all attributes have been applied, the MRE is internally consistent.

Knowing is not enough; we must apply. Willing is not enough; we must do.
— Goethe

Appendix D

Real-time Platform Reference

We demonstrate how designers can employ UNIFY and Object Model Template (OMT) to achieve effective Multi-Representation Modelling (MRM). We incorporate UNIFY in Real-time Platform Reference (RPR) [RPR97], a military model that is part of the Department of Defence's High Level Architecture (HLA). RPR is specified using OMT [OMT98]. From the RPR specifications, we construct an MRE and show how to maintain consistency within this MRE when concurrent interactions occur.

We construct a Platoon-Tanks Multiple Representation Entity (MRE) from the RPR specifications. We assume that the jointly-executing models in RPR are a Platoon model and a Tank model. For brevity, we assume that a Platoon consists of only two Tanks, as shown in Figure 72. From the OMT tables in the RPR specification, we determine the attributes in the representations of the Platoon and Tank models. Next, we capture the relationships among attributes using an Attribute Dependency Graph (ADG) and select mapping functions to maintain consistency in a Platoon-Tanks MRE. Finally, we select policies for resolving the effects of concurrent interactions.

In §D.1, we present the tables in OMT. In §D.2, we list steps for incorporating UNIFY in RPR. We demonstrate each step in subsequent sections. In §D.3, we construct an MRE. In §D.4 and §D.5, we construct an ADG and select mapping functions for attribute dependencies in the MRE. In §D.6 and §D.7, we determine and resolve the effects of concurrent interactions. In §D.8, we construct a CE and IR for the MRE.

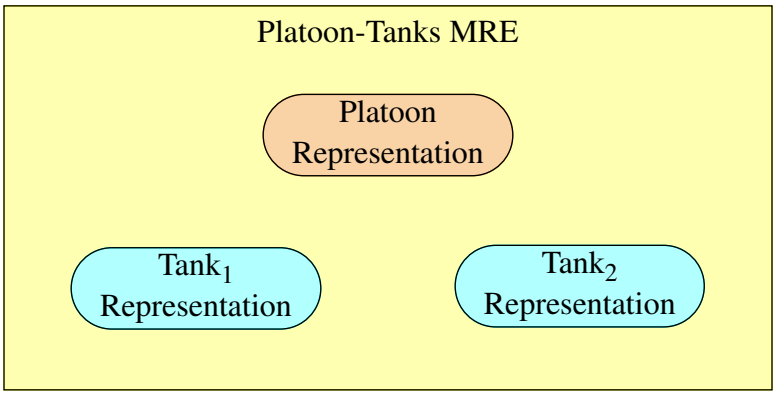


FIGURE 72: Platoon-Tanks MRE

D.1 OMT Tables

OMT consists of a number of tables for specifying parts of a model. They are:

1. Object Class Structure Table (OCST): Shows the class hierarchy along with publishable/subscribable information for each class.
2. Attribute/Parameter Table (APT): Lists object attributes and interaction parameters along their data type, cardinality, units, resolution, accuracy, accuracy condition, update type and update condition.
3. Object Interaction Table (OIT): Lists each possible interaction and associated information, such as its sender, its receiver and the attributes it affects.
4. Enumerated Data Table (EDT): Lists the values of all enumerations.
5. Complex Data Table (CDT): Lists the definitions of all structured data types.

6. Object Class Definitions (OCD): Describes the role of each entity.
7. Object Interaction Definitions (OID): Describes each interaction.
8. Attribute/Parameter Definitions (APD): Describes each object attribute and interaction parameter.

We augment the OIT with the class of each interaction. Also, we add two tables to OMT to capture attribute relationships and specify policies for concurrent interactions.

9. Attribute Relationships Table (ART): Lists each attribute dependency, its type, its mapping function and requirements and properties of the mapping function.
10. Concurrent Interactions Table (CIT): Lists policies for resolving classes and instances of concurrent interactions.

D.2 Steps

The steps for incorporating `UNIFY` in RPR are:

1. Construct an MRE from the OCST and the APT
2. Construct an ADG from the APT and the ART
3. Select Mapping Functions for Dependencies in the ART
4. Determine the Effects of Interactions from the OIT
5. Resolve the Effects of Concurrent Interactions from the CIT
6. Construct a Consistency Enforcer and an Interaction Resolver

D.3 Construct an MRE from the OCST and the APT

We construct a Platoon-Tanks MRE to execute a Platoon model and a Tank model jointly. Using the OCST for RPR (shown in Table 30), we derive a Platoon from

AggregateEntity, and a Tank from MilitaryLandPlatform. Our Platoon-Tanks MRE consists of the representations of a Platoon and two Tanks, $Tank_1$ and $Tank_2$.

TABLE 30: Object Class Structure Table for RPR

Base Class	1st Subclass	2nd Subclass	3rd Subclass	4th Subclass
BaseEntity	AggregateEntity			
	EnvironmentEntity			
	PhysicalEntity	MilitaryEntity	MilitaryPlatformEntity	MilitaryAirLandPlatform
				MilitaryAmphibiousPlatform
				MilitaryLandPlatform
				MilitarySpacePlatform
				MilitarySeaSurfacePlatform
				MilitarySubmersiblePlatform
				MilitaryMultiDomainPlatform
			MunitionEntity	
			Soldier	
		CivilPlatform	CivilAirLandPlatform	
			CivilAmphibiousPlatform	
			CivilLandPlatform	
			CivilSpacePlatform	
			CivilSeaSurfacePlatform	
			CivilSubmersiblePlatform	
			CivilMultiDomainPlatform	
		Civilian		
EmbeddedSystem	Designator			
	EmitterSystem			
	RadioReceiver			
	RadioTransmitter			
EmitterBeam	TrackJamBeam			
SimulationManager				

From the APT, we determine the attributes that are part of the concurrent representations within our Platoon-Tanks MREs. For brevity, Table 31 shows only part of the APT for RPR. The table lists attributes only for base classes of Platoon and Tank. For each attribute, the designer may specify information such as its data type, units, resolution, accuracy, condition under which the specified accuracy is required and update type.

TABLE 31: Attribute/Parameter Table for RPR

Object/ Interaction	Attribute/Parameter	Datatype	Cardinality	Units	Resolution	Accuracy	Accuracy Condition	Update Type	Update Condition
AggregateEntity	AggregateMarking	structure	1					static	
	AggregateState	enumeration	1					conditional	on change
	Dimensions	structure	1					conditional	AggSizeChange
	EntityIDs	unsigned long	0+			perfect	always	conditional	on change
	ForceID	enumeration	1					static	
	Formation	enumeration	1					conditional	on change
	NumberOfEntities	unsigned short	1		1	perfect	always	conditional	on change
	NumberOfSilentAggregates	unsigned short	1		1	perfect	always	conditional	on change
	NumberOfSilentEntities	unsigned short	1		1	perfect	always	conditional	on change
	NumberOfSubAggregates	unsigned short	1		1	perfect	always	conditional	on change
	NumberOfVariableDatums	unsigned short	1		1	perfect	always	conditional	on change
	SilentAggregates	structure	0+					conditional	on change
	SilentEntities	structure	0+					conditional	on change
	SubAggregateIDs	unsigned long	0+			perfect	always	conditional	on change
	VariableDatums	structure	0+					conditional	on change
BaseEntity	AccelerationVector	structure	1					conditional	AccelerationChange
	AngularVelocityVector	structure	1					conditional	AngVelocityChange
	DRAlgorithm	enumeration	1					conditional	on change
	EntityType	structure	1					static	
	FederateID	structure	1					static	
	IsFrozen	boolean	1	TRUE, FALSE		perfect	always	conditional	on change
	Orientation	structure	1					conditional	OrientationChange
	Position	structure	1					conditional	PositionChange
	VelocityVector	structure	1					conditional	VelocityChange
MilitaryEntity	AlternateEntityType	structure	1					static	
	CamouflageType	boolean	1	TRUE, FALSE		perfect	always	conditional	on change
	FirePowerDisabled	boolean	1	TRUE, FALSE		perfect	always	conditional	on change
	ForceID	enumeration	1			perfect	always	static	
	IsConcealed	boolean	1	TRUE, FALSE		perfect	always	conditional	on change
MilitaryPlatformEntity	AfterburnerOn	boolean	1	TRUE, FALSE		perfect	always	conditional	on change
	HasAmmunitionSupplyCap	boolean	1	TRUE, FALSE		perfect	always	static	on change
	LauncherRaised	boolean	1	TRUE, FALSE		perfect	always	conditional	on change

TABLE 31: Attribute/Parameter Table for RPR

Object/ Interaction	Attribute/Parameter	Datatype	Cardinality	Units	Resolution	Accuracy	Accuracy Condition	Update Type	Update Condition
PhysicalEntity	ArticulatedParametersArray	structure	0+					conditional	on change
	ArticulatedParametersCount	unsigned short	1		1	perfect	always	static	
	DamageState	enumeration	1					conditional	on change
	EngineSmokeOn	boolean	1	TRUE, FALSE		perfect	always	conditional	on change
	FlamesPresent	boolean	1	TRUE, FALSE		perfect	always	conditional	on change
	HasFuelSupplyCap	boolean	1	TRUE, FALSE		perfect	always	static	on change
	HasRecoveryCap	boolean	1	TRUE, FALSE		perfect	always	static	on change
	HasRepairCap	boolean	1	TRUE, FALSE		perfect	always	static	on change
	HatchState	enumeration	1					conditional	on change
	Immobilized	boolean	1	TRUE, FALSE		perfect	always	conditional	on change
	LifeformState	enumeration	1					conditional	on change
	LightsState	enumeration	1					conditional	on change
	Marking	structure	1					static	on change
	PowerPlantOn	boolean	1	TRUE, FALSE		perfect	always	conditional	on change
	RampDeployed	boolean	1	TRUE, FALSE		perfect	always	conditional	on change
	SmokePlumePresent	boolean	1	TRUE, FALSE		perfect	always	conditional	on change
	TentDeployed	boolean	1	TRUE, FALSE		perfect	always	conditional	on change
	TrailState	enumeration	1					conditional	on change

From the OCST (Table 30) and APT (Table 31), we derive the attributes of a Tank and a Platoon. Table 32 lists the attributes of Platoon, Tank_1 and Tank_2. For brevity, we combine a number of logically-similar attributes derived from the OCST and APT (second column) into one attribute (fourth column). For example, we combine the attributes IsFrozen, IsConcealed, FlamesPresent and LifeformState into an attribute called Status. We combine such attributes so that we can present a simple MRE, for which an ADG will be presentable and specifying mapping functions will be manageable. Combining similar attributes is consistent with our discussion about assigning nodes of an ADG (§6.1.1). A node can be assigned to any subset of a representation for which a designer can specify how the effects of interactions must be applied. In practice, we expect designers to assign nodes to individual attributes rather than combined attributes.

TABLE 32: Attributes of Platoon, $Tank_1$ and $Tank_2$ (RPR)

Entity	Original Attributes	Derived From	New Attributes
Platoon	AccelerationVector	BaseEntity	Acceleration
	AngularVelocityVector		AngularVelocity
	DRAlgorithm		DRAlgorithm
	EntityType		InitialParameters
	FederateID		
	IsFrozen		Status
	Orientation		Orientation
	Position		Position
	VelocityVector		Velocity
	AggregateMarking	AggregateEntity	InitialParameters
	ForceID		
	Dimensions		Dimensions
	Formation		Formation
	EntityIDs		Composition
	AggregateState		
	NumberOfEntities		
	NumberOfSilentAggregates		
	NumberOfSilentEntities		
	NumberOfSubAggregates		
	NumberOfVariableDatums		
	SilentAggregates		
	SilentEntities		
	SubAggregateIDs		
	VariableDatums		

TABLE 32: Attributes of Platoon, $Tank_1$ and $Tank_2$ (RPR)

Entity	Original Attributes	Derived From	New Attributes
$Tank_1$	AccelerationVector	BaseEntity	$Acceleration_1$
	AngularVelocityVector		$AngularVelocity_1$
	DRAlgorithm		$DRAlgorithm_1$
	EntityType		$InitialParameters_1$
	FederateID		
	IsFrozen		$Status_1$
	Orientation		$Orientation_1$
	Position		$Position_1$
	VelocityVector		$Velocity_1$
	ArticulatedParametersArray	PhysicalEntity	$ArticulatedParameters_1$
	ArticulatedParametersCount		
	DamageState		$Status_1$
	EngineSmokeOn		
	FlamesPresent		
	HasFuelSupplyCap		
	HasRecoveryCap		
	HasRepairCap		
	HatchState		
	Immobilized		
	LifeformState		
	LightsState		
	Marking		
	PowerPlantOn		
	RampDeployed		
	SmokePlumePresent		
	TentDeployed		
	TrailState		
	AlternateEntityType	MilitaryEntity	$InitialParameters_1$
	ForceID		
	CamouflageType		$Status_1$
	FirePowerDisabled		
	IsConcealed		
	AfterburnerOn	MilitaryPlatformEntity	
	HasAmmunitionSupplyCap		
	LauncherRaised		
	<none>	MilitaryLandPlatform	

TABLE 32: Attributes of Platoon, $Tank_1$ and $Tank_2$ (RPR)

Entity	Original Attributes	Derived From	New Attributes
$Tank_2$	AccelerationVector	BaseEntity	$Acceleration_2$
	AngularVelocityVector		$AngularVelocity_2$
	DRAlgorithm		$DRAlgorithm_2$
	EntityType		$InitialParameters_2$
	FederateID		
	IsFrozen		$Status_2$
	Orientation		$Orientation_2$
	Position		$Position_2$
	VelocityVector		$Velocity_2$
	ArticulatedParametersArray	PhysicalEntity	$ArticulatedParameters_2$
	ArticulatedParametersCount		
	DamageState		$Status_2$
	EngineSmokeOn		
	FlamesPresent		
	HasFuelSupplyCap		
	HasRecoveryCap		
	HasRepairCap		
	HatchState		
	Immobilized		
	LifeformState		
	LightsState		
	Marking		
	PowerPlantOn		
	RampDeployed		
	SmokePlumePresent		
	TentDeployed		
	TrailState		
	AlternateEntityType	MilitaryEntity	$InitialParameters_2$
	ForceID		
	CamouflageType		$Status_2$
	FirePowerDisabled		
	IsConcealed		
	AfterburnerOn	MilitaryPlatformEntity	
	HasAmmunitionSupplyCap		
	LauncherRaised		
	<none>	MilitaryLandPlatform	

D.4 Construct an ADG from the APT and the ART

We construct an ADG for the Platoon-Tanks MRE from the APT and the ART for RPR. Since OMT does not support specifying relationships, we construct an example ART for our MRE (Table 33). In practice, we expect a designer to construct an ART specific to the models executed jointly. The specification of the relationship may be accomplished formally; in Table 33, we present informal specifications in the last column.

TABLE 33: Attribute Relationship Table for Platoon-Tanks MRE in RPR

Dependency	Type	Specification
$\text{Position}_1 \rightarrow \text{Position}$	Cumulative	The position of the platoon is the centroid of the position of its tanks.
$\text{Position}_2 \rightarrow \text{Position}$	Cumulative	
$\text{Position} \rightarrow \text{Position}_1$	Distributive	
$\text{Position} \rightarrow \text{Position}_2$	Distributive	
$\text{Velocity}_1 \rightarrow \text{Velocity}$	Cumulative	The velocity of the platoon is the average of the velocity of its tanks.
$\text{Velocity}_2 \rightarrow \text{Velocity}$	Cumulative	
$\text{Velocity} \rightarrow \text{Velocity}_1$	Distributive	
$\text{Velocity} \rightarrow \text{Velocity}_2$	Distributive	
$\text{Orientation}_1 \rightarrow \text{Orientation}$	Cumulative	The orientation of the platoon is the average of the orientations of its tanks.
$\text{Orientation}_2 \rightarrow \text{Orientation}$	Cumulative	
$\text{Orientation} \rightarrow \text{Orientation}_1$	Distributive	
$\text{Orientation} \rightarrow \text{Orientation}_2$	Distributive	
$\text{Status}_1 \rightarrow \text{Composition}$	Cumulative	The composition of the platoon changes if tanks are fatally damaged.
$\text{Status}_2 \rightarrow \text{Composition}$	Cumulative	
$\text{Composition} \rightarrow \text{Status}_1$	Distributive	
$\text{Composition} \rightarrow \text{Status}_2$	Distributive	
$\text{Velocity} \rightarrow \text{Position}$	Modelling	The position of a platoon or a tank depends on its velocity.
$\text{Velocity}_1 \rightarrow \text{Position}_1$	Modelling	
$\text{Velocity}_2 \rightarrow \text{Position}_2$	Modelling	
$\text{Acceleration} \rightarrow \text{Velocity}$	Modelling	The velocity of a platoon or a tank depends on its acceleration.
$\text{Acceleration}_1 \rightarrow \text{Velocity}_1$	Modelling	
$\text{Acceleration}_2 \rightarrow \text{Velocity}_2$	Modelling	
...		

We construct an ADG for the Platoon-Tanks MRE. From Table 32, which was derived from the APT, we determine the nodes in the ADG. From the ART in Table 33, we determine the arcs in the ADG. The ADG is shown in Figure 73. The interaction dependencies to each attribute exist because interactions with other entities or internal actions of the MRE may change any attribute.

Dynamic semantics of attribute relationships may be captured by weighting dependencies. Dependency classes capture static semantics, whereas weights capture dynamic semantics. For our Platoon-Tanks MRE, we assign a weight of one to each cumulative dependency, and equal weights to distributive dependencies that have the same independent attribute. We select these weights in order to keep our subsequent discussion of mapping functions simple. Other weights for these dependencies are possible.

D.5 Select Mapping Functions for Dependencies in the ART

We select mapping functions to translate attributes among concurrent representations within the Platoon-Tanks MRE. Recall from Chapter 6 that mapping functions must translate values or changes in values of attributes from one to another. Additionally, it is desirable that mapping functions complete their translations in a time-bound manner, and that they be composable and reversible. We show mapping functions for some dependencies in Table 34. The mapping functions are presented as pseudo-code. Error-checking has been omitted for brevity. Pseudo-code in the second column of Table 34 implements specifications in the last column of Table 33. Mapping functions such as those shown in Table 34 translate values or changes in values of attributes.

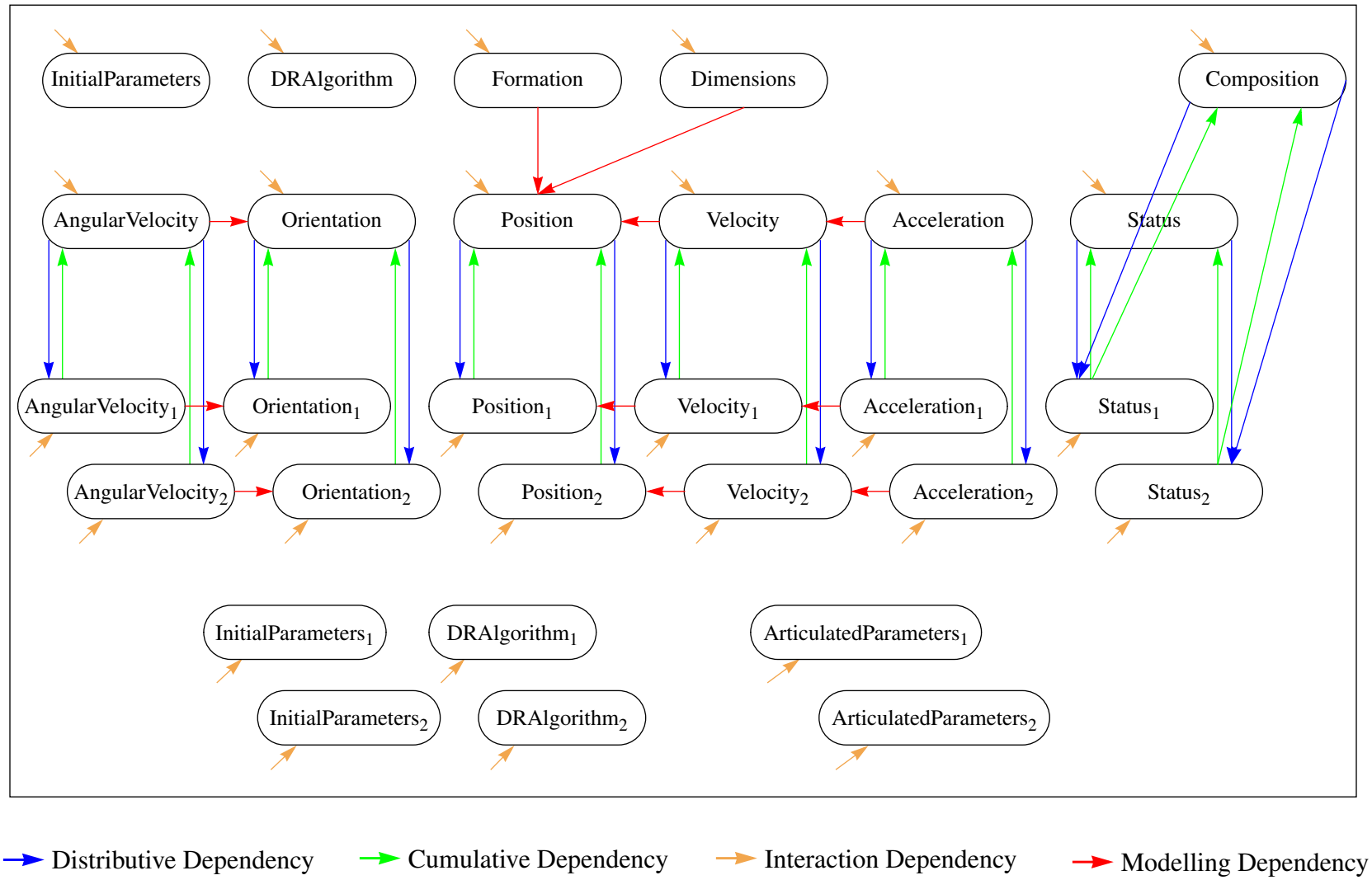


FIGURE 73: ADG for the RPR Platoon-Tanks MRE

TABLE 34: Mapping Functions for RPR Platoon-Tanks MRE

Dependency	Mapping Function
$\text{Position}_1 \rightarrow \text{Position}$ $\text{Position}_2 \rightarrow \text{Position}$	$\text{Position} \leftarrow f_d(\text{Position}_1, \text{Position}_2)$ f_l: $\text{Position.X} \leftarrow (\text{Position}_1\text{.X} + \text{Position}_2\text{.X}) / 2$ $\text{Position.Y} \leftarrow (\text{Position}_1\text{.Y} + \text{Position}_2\text{.Y}) / 2$ $\text{Position.Z} \leftarrow (\text{Position}_1\text{.Z} + \text{Position}_2\text{.Z}) / 2$
$\text{Position} \rightarrow \text{Position}_1$ $\text{Position} \rightarrow \text{Position}_2$	$(\text{Position}_1, \text{Position}_2) \leftarrow g_d(\text{Position})$ g_l: $\delta\text{Position}_1\text{.X} \leftarrow \delta\text{Position}_2\text{.X} \leftarrow \delta\text{Position.X}$ $\delta\text{Position}_1\text{.Y} \leftarrow \delta\text{Position}_2\text{.Y} \leftarrow \delta\text{Position.Y}$ $\delta\text{Position}_1\text{.Z} \leftarrow \delta\text{Position}_2\text{.Z} \leftarrow \delta\text{Position.Z}$
$\text{Velocity}_1 \rightarrow \text{Velocity}$ $\text{Velocity}_2 \rightarrow \text{Velocity}$	$\text{Velocity} \leftarrow f_d(\text{Velocity}_1, \text{Velocity}_2)$ f_v: $\text{Velocity.X} \leftarrow (\text{Velocity}_1\text{.X} + \text{Velocity}_2\text{.X}) / 2$ $\text{Velocity.Y} \leftarrow (\text{Velocity}_1\text{.Y} + \text{Velocity}_2\text{.Y}) / 2$ $\text{Velocity.Z} \leftarrow (\text{Velocity}_1\text{.Z} + \text{Velocity}_2\text{.Z}) / 2$
$\text{Velocity} \rightarrow \text{Velocity}_1$ $\text{Velocity} \rightarrow \text{Velocity}_2$	$(\text{Velocity}_1, \text{Velocity}_2) \leftarrow g_d(\text{Velocity})$ g_v: $\delta\text{Velocity}_1\text{.X} \leftarrow \delta\text{Velocity}_2\text{.X} \leftarrow \delta\text{Velocity.X}$ $\delta\text{Velocity}_1\text{.Y} \leftarrow \delta\text{Velocity}_2\text{.Y} \leftarrow \delta\text{Velocity.Y}$ $\delta\text{Velocity}_1\text{.Z} \leftarrow \delta\text{Velocity}_2\text{.Z} \leftarrow \delta\text{Velocity.Z}$
$\text{Orientation}_1 \rightarrow \text{Orientation}$ $\text{Orientation}_2 \rightarrow \text{Orientation}$	$\text{Orientation} \leftarrow f_d(\text{Orientation}_1, \text{Orientation}_2)$ f_o: $\text{Orientation.Psi} \leftarrow$ $(\text{Orientation}_1\text{.Psi} + \text{Orientation}_2\text{.Psi}) / 2$ $\text{Orientation.Theta} \leftarrow$ $(\text{Orientation}_1\text{.Theta} + \text{Orientation}_2\text{.Theta}) / 2$ $\text{Orientation.Phi} \leftarrow$ $(\text{Orientation}_1\text{.Phi} + \text{Orientation}_2\text{.Phi}) / 2$
$\text{Orientation} \rightarrow \text{Orientation}_1$ $\text{Orientation} \rightarrow \text{Orientation}_2$	$(\text{Orientation}_1, \text{Orientation}_2) \leftarrow g_d(\text{Orientation})$ g_o: $\delta\text{Orientation}_1\text{.Psi} \leftarrow \delta\text{Orientation}_2\text{.Psi} \leftarrow$ $\delta\text{Orientation.Psi}$ $\delta\text{Orientation}_1\text{.Theta} \leftarrow \delta\text{Orientation}_2\text{.Theta} \leftarrow$ $\delta\text{Orientation.Theta}$ $\delta\text{Orientation}_1\text{.Phi} \leftarrow \delta\text{Orientation}_2\text{.Phi} \leftarrow$ $\delta\text{Orientation.Phi}$
...	

The mapping functions shown in Table 34 are composable and reversible. Moreover, since they are simple in construction, we expect that they will complete in a time-bound manner, thus ensuring that the Platoon-Tanks MRE is consistent at all observation times. When an interaction changes the value of any attribute, mapping functions propagate the change in the attribute to dependent attributes. For example, if an interaction changes the

Tank-level attribute, Orientation_1, the mapping function f_o changes the dependent Platoon-level attribute, Orientation. Subsequently, the mapping function g_o changes the Tank-level attribute, Orientation_2. Since f_o and g_o are composable, the change to Orientation_1 eventually propagates to Orientation_2. Since f_o and g_o are reversible, Orientation_1 does not change again as a result of the same interaction.

When an interaction occurs, traversing the ADG in Figure 73 and applying the mapping functions in Table 34 ensures that the Platoon-Tanks MRE is consistent at all observation times. Next, we determine and resolve the effects of concurrent interactions.

D.6 Determine the Effects of Interactions from the OIT

We determine the effects of interactions on the Platoon-Tanks MRE from the OIT. We show an augmented OIT in Table 35. The first column lists the name of the interaction. The next four columns list the class and affected attributes for the sender and receiver of the interaction. We augment each interaction in the OIT with its type (see Chapter 7): Type 0 (certain responses), Type 1 (uncertain responses), Type 2 (certain requests), and Type 3 (uncertain requests). We do not utilise the ISR (Init/Sense/React) information and the parameters of an interaction in `UNIFY`.

The OIT lists interactions among entities, but not internal actions of an entity. For example, the OIT does not list any interaction corresponding to our Platoon-Tanks MRE changing its course, because such an interaction is internal to the MRE. In `UNIFY`, internal actions are interactions. We add an internal action called ChangeCourse to the interactions in the OIT (see last row in Table 35) to show that `UNIFY` addresses internal actions as well as interactions with other entities. This interaction initiates a change in the course of an

entity. The sender and receiver of ChangeCourse is the same entity. The class of that entity is Player. The interaction affects the attributes Position, Velocity and Orientation.

The last column in Table 35 lists the type of an interaction. Assigning a type requires information about the semantics of an interaction. For example, the semantics of CreateObjectRequest could be that the SimulationManager requests an AggregateEntity to create a new entity as its constituent. If such a request must always be satisfied by an AggregateEntity, CreateObjectRequest is a Type 2 interaction. CreateObjectResult is the response to a CreateObjectRequest. CreateObjectResult could be Type 0 or Type 1, but we assigned it to Type 1 because the SimulationManager may discard an update about the created object. For the ChangeCourse interaction, we assumed that a change in the course of an entity is a request whose outcome is uncertain.

TABLE 35: Object Interaction Table for RPR

Interaction	Sender Class	Sender Attributes	Receiver	Receiver Attributes	Interaction Parameters	ISR	Type
ActionRequest	SimulationManager	none	AggregateEntity	none	ObjectCount, ObjectIDs, Action	IR	2
ActionResult	AggregateEntity	none	SimulationManager	none	ActionResult	IR	1
AttributeChangeRequest	SimulationManager	none	AggregateEntity	none	ObjectCount, ObjectIDs, AttributeValueSet	IR	2
AttributeChangeResult	AggregateEntity	none	SimulationManager	none	ObjectID, AttributeChangeResult, AttributeValueSet	IR	1
Collision	PhysicalEntity	Acceleration, AngularVelocity, Status, Orientation, Position, Velocity	PhysicalEntity	Acceleration, AngularVelocity, Status, Orientation, Position, Velocity	CollidingObjectID, CollidingObjectMass, CollidingObjectVelocity, CollisionType, CollisionLocation, EventID, IssuingObjectID	IR	0
CreateObjectRequest	SimulationManager	none	AggregateEntity	none	ObjectClass, AttributeValueSet	IR	2
CreateObjectResult	AggregateEntity	none	SimulationManager	none	CreateObjectResult	IR	1
MunitionDetonation	MilitaryPlatformEntity	none	PhysicalEntity	Acceleration, AngularVelocity, Status, Orientation, Position, Velocity	ArticulatedPartsArray, ArticulatedPartsCount, DetonationLocation, DetonationResult, EventID, FiringObjectID, FinalVelocityVector, FuseType, MunitionObjectID, MunitionType, QuantityFired, RateOfFire, RelativeDetonationLocation, TargetObjectID	IR	0
RemoveObjectRequest	SimulationManager	none	AggregateEntity	none	ObjectCount, ObjectIDs	IR	2
RemoveObjectResult	AggregateEntity	none	SimulationManager	none	RemoveObjectResult	IR	1
WeaponFire	MilitaryEntity	none		none	EventID, FireControlSolutionRange, FireMissionIndex, FiringLocation, FiringObjectID, FuseType, InitialVelocityVector, MunitionObjectID, MunitionType, QuantityFired, RateOfFire, TargetObjectID, WarheadType	IR	0
ChangeCourse	BaseEntity	Position, Velocity, Orientation	BaseEntity	Position, Velocity, Orientation	New_Location, New_Velocity, New_Orientation	IR	3

We determine the interactions that our Platoon-Tanks MRE can send and receive. In Table 36, we list the interactions that Platoon, $Tank_1$ and $Tank_2$ can send and receive. In the first column, we list the name of an interaction as the name in the OIT along with a suffix that indicates whether Platoon, $Tank_1$ or $Tank_2$ sends or receives that interaction. For example, the interaction ChangeCourse can be sent by an entity of class BaseEntity. Since BaseEntity is a base class of Platoon, $Tank_1$ and $Tank_2$, we distinguish the interaction ChangeCourse sent by these three entities as ChangeCourse-P, ChangeCourse-T_1 and ChangeCourse-T_2 respectively. In the second column, we indicate whether the Platoon-Tanks MRE sends (S) or receives (R) the interaction. In the third column, we list the attributes affected by the interaction directly, i.e., we list the set *affects* for the interaction. These attributes are determined from the OIT. In the fourth column, we list the attributes affected by the interaction indirectly, i.e., we list the set $affects^+$ for the interaction. These attributes can be determined from the ADG in Figure 73. Finally, we indicate the type of the interaction.

TABLE 36: Effects of Interactions for RPR Platoon-Tanks MRE

Interaction	S/R	*affects*	$affects^+$	Type
ActionResult-P	S			1
AttributeChangeResult-P	S			1
CreateObjectResult-P	S			1
RemoveObjectResult-P	S			1

TABLE 36: Effects of Interactions for RPR Platoon-Tanks MRE

Interaction	S/R	*affects*	*affects*$^+$	Type
Collision-T_1	S	$Acceleration_1$, $AngularVelocity_1$, $Status_1$, $Velocity_1$, $Orientation_1$, $Position_1$	Acceleration, Status, AngularVelocity, Velocity, Orientation, Position, Composition, $Acceleration_2$, $Status_2$, $AngularVelocity_2$, $Velocity_2$, $Orientation_2$, $Position_2$, $Acceleration_1$, $Status_1$, $AngularVelocity_1$, $Velocity_1$, $Orientation_1$, $Position_1$	0
Collision-T_2	S	$Acceleration_2$, $AngularVelocity_2$, $Status_2$, $Velocity_2$, $Orientation_2$, $Position_2$	Acceleration, Status, AngularVelocity, Velocity, Orientation, Position, Composition, $Acceleration_1$, $Status_1$, $AngularVelocity_1$, $Velocity_1$, $Orientation_1$, $Position_1$, $Acceleration_2$, $Status_2$, $AngularVelocity_2$, $Velocity_2$, $Orientation_2$, $Position_2$	0
WeaponFire-T_1	S			0
WeaponFire-T_2	S			0
ChangeCourse-P	S	Position, Velocity, Orientation	$Position_1$, $Velocity_1$, $Orientation_1$, $Position_2$, $Velocity_1$, $Orientation_2$, Position, Velocity, Orientation	3
ChangeCourse-T_1	S	$Position_1$, $Velocity_1$, $Orientation_1$	Position, Velocity, Orientation, $Position_2$, $Velocity_2$, $Orientation_2$, $Position_1$, $Velocity_1$, $Orientation_1$	3
ChangeCourse-T_2	S	$Position_2$, $Velocity_2$, $Orientation_2$	Position, Velocity, Orientation, $Position_1$, $Velocity_1$, $Orientation_1$, $Position_2$, $Velocity_2$, $Orientation_2$	3
ActionRequest-P	R			2

TABLE 36: Effects of Interactions for RPR Platoon-Tanks MRE

Interaction	S/R	*affects*	*affects*$^+$	Type
AttributeChangeRequest-P	R			2
CreateObjectRequest-P	R			2
RemoveObjectRequest-P	R			2
Collision-T_1	R	$Acceleration_1$, $AngularVelocity_1$, $Status_1$, $Velocity_1$, $Orientation_1$, $Position_1$	Acceleration, Status, AngularVelocity, Velocity, Orientation, Position, Composition, $Acceleration_2$, $Status_2$, $AngularVelocity_2$, $Velocity_2$, $Orientation_2$, $Position_2$, $Acceleration_1$, $Status_1$, $AngularVelocity_1$, $Velocity_1$, $Orientation_1$, $Position_1$	0
Collision-T_2	R	$Acceleration_2$, $AngularVelocity_2$, $Status_2$, $Velocity_2$, $Orientation_2$, $Position_2$	Acceleration, Status, AngularVelocity, Velocity, Orientation, Position, Composition, $Acceleration_1$, $Status_1$, $AngularVelocity_1$, $Velocity_1$, $Orientation_1$, $Position_1$, $Acceleration_2$, $Status_2$, $AngularVelocity_2$, $Velocity_2$, $Orientation_2$, $Position_2$	0
MunitionDetonation-T_1	R	$Acceleration_1$, $AngularVelocity_1$, $Status_1$, $Velocity_1$, $Orientation_1$, $Position_1$	Acceleration, Status, AngularVelocity, Velocity, Orientation, Position, Composition, $Acceleration_2$, $Status_2$, $AngularVelocity_2$, $Velocity_2$, $Orientation_2$, $Position_2$, $Acceleration_1$, $Status_1$, $AngularVelocity_1$, $Velocity_1$, $Orientation_1$, $Position_1$	0

TABLE 36: Effects of Interactions for RPR Platoon-Tanks MRE

Interaction	S/R	*affects*	*affects*$^+$	Type
MunitionDetonation-T_2	R	$Acceleration_2$, $AngularVelocity_2$, $Status_2$, $Velocity_2$, $Orientation_2$, $Position_2$	Acceleration, Status, AngularVelocity, Velocity, Orientation, Position, Composition, $Acceleration_1$, $Status_1$, $AngularVelocity_1$, $Velocity_1$, $Orientation_1$, $Position_1$, $Acceleration_2$, $Status_2$, $AngularVelocity_2$, $Velocity_2$, $Orientation_2$, $Position_2$	0
ChangeCourse-P	R	Position, Velocity, Orientation	$Position_1$, $Velocity_1$, $Orientation_1$, $Position_2$, $Velocity_1$, $Orientation_2$, Position, Velocity, Orientation	3
ChangeCourse-T_1	R	$Position_1$, $Velocity_1$, $Orientation_1$	Position, Velocity, Orientation, $Position_2$, $Velocity_2$, $Orientation_2$, $Position_1$, $Velocity_1$, $Orientation_1$	3
ChangeCourse-T_2	R	$Position_2$, $Velocity_2$, $Orientation_2$	Position, Velocity, Orientation, $Position_1$, $Velocity_1$, $Orientation_1$, $Position_2$, $Velocity_2$, $Orientation_2$	3

Any subset of the interactions in Table 36 may occur concurrently. Next, we show how to resolve the effects of concurrent interactions.

D.7 Resolve the Effects of Concurrent Interactions from the CIT

The effects of concurrent interactions can be resolved by implementing polices from the CIT. In practice, a designer constructs a CIT specific to the application. Since a CIT is unavailable in OMT, we construct an example CIT, shown in Table 37.

A designer specifies policies in the CIT for resolving the effects of concurrent interactions. The CIT consists of sets of concurrent interactions with dependent effects, policies for resolving them and conditions under which the policies are applicable. Concurrent interactions that are independent of one another can be resolved by serialization and are not specified in the CIT. Some interactions may be independent because they affect disjoint sets of attributes. Other interactions may be independent because their effects are applied in different time-steps, for example, interactions sent and received by an entity. Yet other interactions are independent because they are request-response pairs. Policies must be specified in the CIT for only the remaining interactions. Policies may be specified for classes of interactions (e.g., the last two rows in Table 37) or for instances of interactions (e.g., all the other rows in Table 37). In RPR, many interactions do not affect any attributes. Although such interactions can be assumed independent, we do not make such an assumption. It is likely that the interactions affect internal attributes in the models. Since OMT is meant to be an interface specification, internal attributes are not listed in the APT. For consistency maintenance, a designer must list internal attributes as well in the APT. Since internal attributes are not listed, we will not assume that interactions that affect disjoint sets of attributes are independent. For

example, although ActionRequest-P and RemoveObjectRequest-P affect no attributes, hence affecting disjoint sets of attributes, we specify policies for resolving these interactions. An Interaction Resolver for the Platoon-Tanks MRE applies the policies in the CIT only if the effects of concurrent interactions conflict. If concurrent interactions do not conflict, they may be serialized.

TABLE 37: Concurrent Interactions Table for RPR Platoon-Tanks MRE

Concurrent Interactions	Condition	Policy
MunitionDetonation-T_i, Collision-T_i	Always	Damage to $Tank_i$ less than sum of damages but greater than minimum of damages; add compensatory interaction to reduce damage
RemoveObjectRequest-P, any combination of (ActionRequest-P, AttributeChangeRequest-P, CreateObjectRequest-P)	Same object	Order all before RemoveObjectRequest-P
CreateObjectRequest-P, any combination of (ActionRequest-P, AttributeChangeRequest-P, RemoveObjectRequest-P)	Same object	Order all after CreateObjectRequest-P
ChangeCourse-P, any combination of (ChangeCourse-T_1, ChangeCourse-T_2)	All received	Ignore all except ChangeCourse-P
Type 0, Type 1	All received	Ignore Type 1
Type 2, Type 3	All received	Ignore Type 3
Any Interaction	Ignored or Delayed	Ignored or Delayed entirely, i.e., no partial effects permitted

D.8 Construct a Consistency Enforcer and an Interaction Resolver

A Consistency Enforcer (CE) and an Interaction Resolver (IR) for an MRE maintain consistency and resolve concurrent interactions respectively. A CE consists of an ADG and mapping functions, whereas an IR consists of policies for resolving concurrent interactions. Figure 74 shows an RPR Platoon-Tanks MRE. The MRE can interact at multiple representation levels — the Platoon and Tank levels — concurrently. Moreover, the concurrent representations within the MRE are consistent at all observation times.

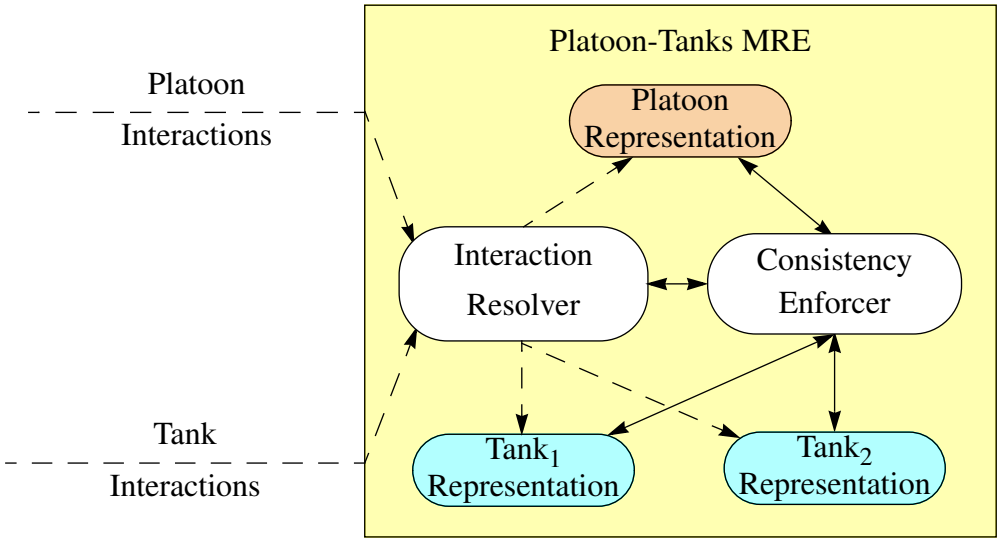


FIGURE 74: RPR Platoon-Tanks MRE

A CE consists of an ADG and application-specific mapping functions. For the Platoon-Tanks MRE, we presented an ADG in Figure 73 and mapping functions in Table 34. In Figure 34 (see Chapter 6), we presented an algorithm for implementing a CE. In §6.3, we discussed how to traverse an ADG and apply mapping functions in order to keep an MRE internally consistent.

An IR consists of application-specific policies for resolving the effects of concurrent interactions. For the Platoon-Tanks MRE, we presented policies for resolving concurrent interactions in Table 37. In Figure 47 (see Chapter 7), we presented an algorithm for implementing an IR. In §7.5, we presented a taxonomy for classifying interactions. Using this taxonomy, we presented policies for resolving the effects of concurrent interactions.

A CE and an IR ensure that an MRE is internally consistent when concurrent interactions occur. During a time-step, a number of concurrent interactions may occur. The IR determines the type of each interaction. Next, the IR applies the effect of each interaction as if the interaction occurred in isolation. In order to do so, the IR permits the interactions to take effect one at a time. When an interaction changes an attribute, the CE traverses an ADG and translates changes to dependent attributes by invoking the appropriate mapping functions. The CE maintains a list of changes for each attribute as a result of computing the effects of each interaction. Subsequently, the CE applies the effects of all the interactions on each attribute. The CE queries the IR about policies to resolve the effects of dependent concurrent interactions whenever the CE detects conflicts in the list of changes for an entity. If the IR contains a policy for resolving conflicting changes, the CE applies the changes accordingly; otherwise, the CE assumes the changes are independent and applies them in an arbitrary order. When the changes to all attributes have been applied, the MRE is internally consistent.

One is always a long way from solving a problem
until one actually has the answer.
— Stephen Hawking

Appendix E

Hierarchical Autonomous Agents

We demonstrate how designers can apply UNIFY to achieve effective Multi-Representation Modelling (MRM) in hierarchical autonomous agents (HAA) [WAS98B]. Hierarchical autonomous agents employ multiple models to achieve a goal. Examples of the models are a planning model that selects actions that an agent can perform, and a perception-action model that senses and changes an agent's surroundings. A HAA may execute multiple models jointly.

The HAA model we considered is part of a research project undertaken by the Vision group at the University of Virginia. The agent, Marcus, has been programmed to construct complex arrangements such as archways from basic building blocks. Marcus is a hierarchical autonomous agent that has two models — a planner model and a perception-action (PA) model. Typically, the planner maintains long-term or abstract representation, whereas the PA system maintains immediate and detailed representation. Each model may have its own representation of the world in which Marcus operates. Accordingly, each model may represent building blocks, partially-completed arrangements, obstacles, doors

and pathways by a number of relevant attributes such as position, orientation and colour. Marcus considers relationships among blocks that are stacked or placed next to each other as an arrangement. We construct an MRE for Marcus and show how to maintain consistency within this MRE when concurrent interactions occur.

The jointly-executing models in Marcus are a planner model and a PA model. We determine the attributes in the planner and PA representations and construct a Multiple Representation Entity (MRE) for Marcus, as shown in Figure 75. Next, we capture the relationships among attributes using an Attribute Dependency Graph (ADG) and select mapping functions to maintain consistency in the Marcus MRE. Finally, we select policies for resolving the effects of concurrent interactions.

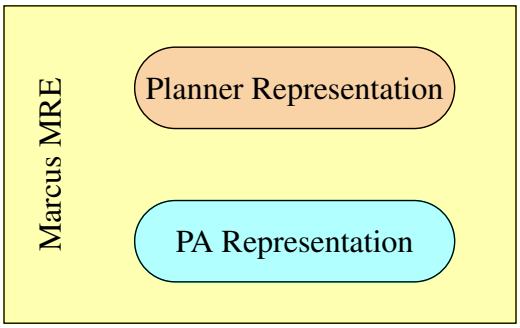


FIGURE 75: Marcus MRE

In §E.1, we list steps for incorporating `UNIFY` in Marcus. We demonstrate each step in subsequent sections. In §E.2, we construct an MRE. In §E.3 and §E.4, we construct an ADG and select mapping functions for attribute dependencies in the MRE. In §E.5 and §E.6, we determine and resolve the effects of concurrent interactions. In §E.7, we construct a CE and IR for the MRE.

E.1 Steps

The steps for incorporating `UNIFY` in Marcus are:

1. Construct an MRE from Planner and PA Representations
2. Construct an ADG for the MRE

3. Select Mapping Functions for Dependencies in the ADG
4. Determine the Effects of Interactions
5. Resolve the Effects of Concurrent Interactions
6. Construct a Consistency Enforcer and an Interaction Resolver

E.2 Construct an MRE from Planner and PA Representations

In order to construct an MRE for Marcus, we must determine what constitutes the representations of the planner and PA models. In HAAs, the representation of a planner or PA consists of attributes that capture properties of objects that are that are important for the current goal. In order to achieve a goal, an agent decomposes the goal into tasks and sub-tasks. Different models in a hierarchical agent architecture view an agent's tasks at different levels of abstraction. Deciding what tasks an agent must execute will be based on the agent's goals and capabilities. Marcus's goal is to build an archway out of coloured blocks scattered throughout a room. This goal can be refined to the planner tasks of *build-tower* and *span-towers-with-block*. The PA model accomplishes these tasks by executing tasks such as *goto-block*, *pick-up-block*, *put-down-block*, *stack-block-on-block* and *span-blocks-with-block*.

After the tasks and sub-tasks have been identified, the representation for each model can be constructed by identifying the objects relevant to the agent's tasks. These objects are parts of the environment that are affected by a task. For example, in the *build-tower* task, *tower* is a relevant object. Likewise, in the *stack-block-on-block task*, two *block*s are the relevant objects. A model represents objects relevant to its current tasks. Attributes are

properties of these objects described using traditional data structures. During the execution of a task, the representation for a model may consist of many attributes for each object in the task. Given Marcus goal of constructing an archway from blocks, we list the attributes of objects represented by the PA and planner (Table 38).

TABLE 38: Attributes of planner and PA (Marcus)

Entity	Object	Attributes
Planner	$Tower_1$	T_1.position, T_1.orientation, T_1.height, T_1.width, T_1.stacked, …
	$Tower_2$	T_2.position, T_2.orientation, T_2.height, T_2.width, T_2.stacked, …
	Arch	A.position, A.orientation, A.height, A.width, A.connected, …
PA	$Block_1$	B_1.position, B_1.orientation, B_1.height, B_1.width, B_1.colour, …
	$Block_2$	B_2.position, B_2.orientation, B_2.height, B_2.width, B_2.colour, …
	$Block_3$	B_3.position, B_3.orientation, B_3.height, B_3.width, B_3.colour, …
	$Block_4$	B_4.position, B_4.orientation, B_4.height, B_4.width, B_4.colour, …
	$Block_5$	B_5.position, B_5.orientation, B_5.height, B_5.width, B_5.colour, …

Wasson addresses how tasks can be decomposed and representation identified for jointly-executing models in HAAs [WAS99].

E.3 Construct an ADG for the MRE

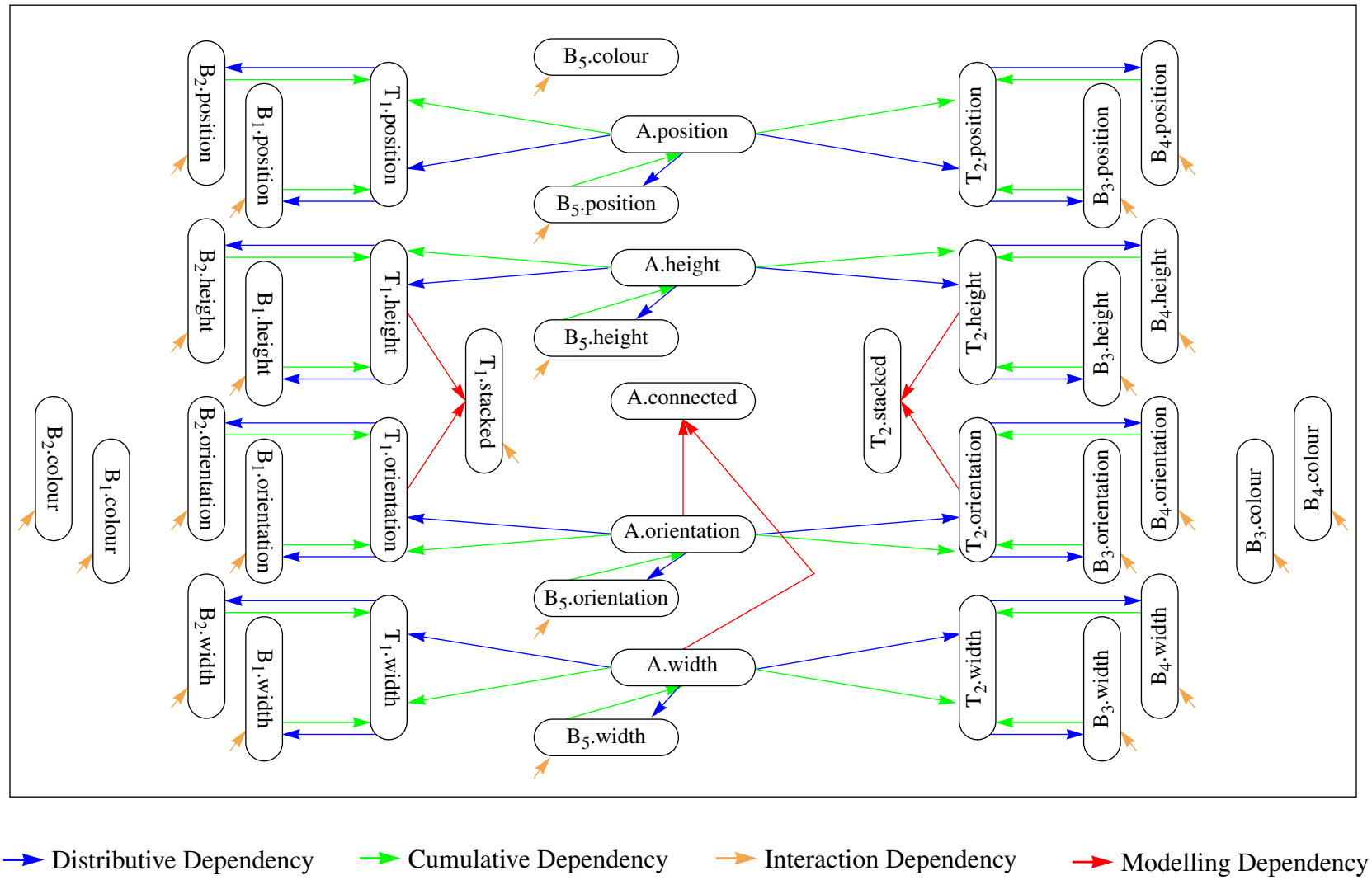
We construct an Attribute Relationship Table (ART) for the attributes in the planner and PA representations. We construct an example ART for our MRE (Table 39); in practice designers provide ARTs for their applications. The specification of the relationship may be accomplished formally; in Table 39, we present informal specifications in the last column.

TABLE 39: Attribute Relationship Table for Marcus MRE

Dependency	Type	Specification
B_1.position $\rightarrow$ T_1.position	Cumulative	The positions of blocks determine the position of a tower and *vice versa*.
B_2.position $\rightarrow$ T_1.position	Cumulative	
T_1.position $\rightarrow$ B_1.position	Distributive	
T_1.position $\rightarrow$ B_2.position	Distributive	
B_1.orientation $\rightarrow$ T_1.orientation	Cumulative	The orientations of blocks determine the orientation of a tower and *vice versa*.
B_2.orientation $\rightarrow$ T_1.orientation	Cumulative	
T_1.orientation $\rightarrow$ B_1.orientation	Distributive	
T_1.orientation $\rightarrow$ B_2.orientation	Distributive	
B_1.height $\rightarrow$ T_1.height	Cumulative	The heights of blocks determine the orientation of a tower and *vice versa*.
B_2.height $\rightarrow$ T_1.height	Cumulative	
T_1.height $\rightarrow$ B_1.height	Distributive	
T_1.height $\rightarrow$ B_2.height	Distributive	
T_1.height $\rightarrow$ T_1.stacked	Modelling	A tower with indeterminate height or orientation is not stacked.
T_1.orientation $\rightarrow$ T_1.stacked	Modelling	
T_1.position $\rightarrow$ A.position	Cumulative	The positions of towers determine the position of an arch and *vice versa*.
T_2.position $\rightarrow$ A.position	Cumulative	
A.position $\rightarrow$ T_1.position	Distributive	
A.position $\rightarrow$ T_2.position	Distributive	
A.width $\rightarrow$ A.stacked	Modelling	A tower with indeterminate width or orientation is not connected.
A.orientation $\rightarrow$ A.stacked	Modelling	
...		

We construct an ADG for the Marcus MRE. From Table 38, we determine the nodes in the ADG. From the ART in Table 39, we determine the arcs in the ADG. The ADG is shown in Figure 76. The interaction dependencies to some attributes exist because interactions with the environment may change those attributes.

The ADG in Figure 76 is constructed by Marcus as it progresses towards its goal. Initially, the ADG in Marcus consists of only nodes corresponding to the attributes at all representation levels. As Marcus stacks blocks to construct a tower or spans towers to construct an arch, it adds arcs to its ADG. If a previously-stacked tower falls apart, a CE in



Distributive Dependency Cumulative Dependency Interaction Dependency Modelling Dependency

FIGURE 76: ADG for the Marcus MRE

Marcus can detect that a relationship among the constituent blocks of the tower no longer holds. Subsequently, the CE changes the values of attributes in the ADG to denote that the tower is no longer stacked. At a future time, the planner model in Marcus can attempt to reconstruct the tower by stacking blocks.

E.4 Select Mapping Functions for Dependencies in the ADG

We select mapping functions to translate attributes among concurrent representations within the Marcus MRE. Recall from Chapter 6 that mapping functions must translate values or changes in values of attributes from one to another. Additionally, it is desirable that mapping functions complete their translations in a time-bound manner, and that they be composable and reversible.

We show mapping functions for some dependencies in Table 40. The mapping functions are presented as pseudo-code. Error-checking has been omitted for brevity. Pseudo-code in the second column of Table 40 implements specifications in the last column of Table 39. The position of a tower is identical to the position of its lowermost block. Conversely, the position of the lowermost block in a tower is identical to the position of the tower. If the positions of other blocks are desired, then appropriate dependencies must be specified, for example, dependencies between the height of the lowermost block and the position of the block immediately above it. If either the height or the orientation of a tower is invalid, the tower is not stacked. The orientation of a tower can become invalid if its constituent blocks are oriented differently. A similar condition may be specified for the height of a tower. Similar mapping functions for other

dependencies can be constructed. Mapping functions such as those shown in Table 40 translate values or changes in values of attributes.

TABLE 40: Mapping Functions for Marcus MRE

Dependency	Mapping Function
B_1.position $\rightarrow$ T_1.position	T_1.position $\leftarrow f_p$(B_1.position, B_2.position) f_p: T_1.position $\leftarrow$ B_1.position
B_2.position $\rightarrow$ T_1.position	
T_1.position $\rightarrow$ B_1.position	(B_1.position, B_2.position) $\leftarrow g_p$(T_1.position) g_p: B_1.position $\leftarrow$ T_1.position
T_1.position $\rightarrow$ B_2.position	
B_1.orientation $\rightarrow$ T_1.orientation	T_1.orientation $\leftarrow f_o$(B_1.orientation, B_2.orientation) f_o: T_1.orientation $\leftarrow$ B_1.orientation
B_2.orientation $\rightarrow$ T_1.orientation	
T_1.orientation $\rightarrow$ B_1.orientation	(B_1.orientation, B_2.orientation) $\leftarrow g_o$(T_1.orientation) g_o: B_1.orientation $\leftarrow$ B_2.orientation $\leftarrow$ T_1.orientation
T_1.orientation $\rightarrow$ B_2.orientation	
T_1.orientation $\rightarrow$ T_1.stacked	T_1.stacked $\leftarrow p_1$(T_1.orientation) p_1: if (T_1.orientation = invalid) T_1.stacked = false
T_1.height $\rightarrow$ T_1.stacked	T_1.stacked $\leftarrow p_2$(T_1.height) p_2: if (T_1.height = invalid) T_1.stacked = false
...	

The mapping functions shown in Table 40 are composable and reversible. Moreover, since they are simple in construction, we expect that they will complete in a time-bound manner, thus ensuring that the Marcus MRE is consistent at all observation times. When an interaction changes the value of any attribute, mapping functions propagate the change in the attribute to dependent attributes. For example, if an interaction changes the PA-level attribute, B_1.orientation, the mapping function f_o changes the dependent planner-level attribute, T_1.orientation. Subsequently, the mapping function g_o changes the PA-level attribute, B_2.orientation. Since f_o and g_o are composable, the change to B_1.orientation eventually propagates to B_2.orientation. Since f_o and g_o are reversible, B_1.orientation does not change again as a result of the same interaction.

When an interaction occurs, traversing the ADG in Figure 76 and applying the mapping functions in Table 40 ensures that the Marcus MRE is consistent at all observation times. Next, we determine and resolve the effects of concurrent interactions.

E.5 Determine the Effects of Interactions

We determine the effects of interactions that Marcus can send and receive. Marcus can interact with its environment only at the PA level. The planner model in Marcus does not interact with the environment apart from initially receiving a goal and finally reporting on the success or failure in achieving the goal. However, the planner interacts with the PA level in order to specify sub-tasks. In Table 41, we list the interactions that the planner and PA levels can send and receive. In the first column, we list the name of an interaction. In the second and third columns, we indicate the sender and receiver of an interaction. A sender or receiver that is not "planner" or "PA", is external to Marcus and is part of Marcus' environment. In the fourth column, we list the attributes affected by the interaction directly, i.e., we list the set *affects* for the interaction. We do not list the set $affects^+$ for the interaction because this set changes as Marcus adds arcs to its ADG while progressing towards its goal. Finally, we indicate the type of the interaction.

We augment each interaction with its type (see Chapter 7): Type 0 (certain responses), Type 1 (uncertain responses), Type 2 (certain requests), and Type 3 (uncertain requests). Assigning a type requires information about the semantics of an interaction. In Marcus, PA-level interactions are assumed to be certain. For example, the PA model in Marcus is assumed to be able to sense the position of an object correctly. Likewise, if the PA model requests the underlying hardware to move Marcus, the hardware will not fail to do so.

Hence, interactions between the PA model and the processor are Type 0 or Type 2. In contrast, planner-level interactions are assumed to be uncertain. For example, Marcus may not be able to pick up a block as per the planner's request. Hence, interactions between the planner model and the PA model are Type 1 or Type 3. Classifying the interactions in this manner reflects the design philosophy of "trusting sensors more than memory".

TABLE 41: Interactions sent and received by the Marcus MRE

Interaction	Sender	Receiver	*affects*	Type
SenseObject(X)	PA	processor		2
UpdateObject(X)	processor	PA	X.position, X.orientation, X.height, X.width, …	0
Move	PA	processor		2
Swivel	PA	processor		2
Turn	PA	processor		2
MoveStatus	processor	PA		0
SwivelStatus	processor	PA		0
TurnStatus	processor	PA		0
PickObject(X)	planner	PA	X.position, X.orientation	3
PutDownObject(X)	planner	PA	X.position, X.orientation	3
StackObject(X, Y)	planner	PA	Y.position, Y.orientation	3
SpanObject(X, Y, Z)	planner	PA	Z.position, Z.orientation	3
GoThroughDoor	planner	PA		3
Travel	planner	PA		3
ActionStatus	PA	planner		1

Any subset of the interactions in Table 41 may occur concurrently. Next, we show how to resolve the effects of concurrent interactions.

E.6 Resolve the Effects of Concurrent Interactions

The effects of concurrent interactions can be resolved by implementing application-specific polices. In practice, a designer selects policies specific to the application. We select example policies, shown in Table 42. A designer specifies policies in a Concurrent Interactions Table (CIT) for resolving the effects of concurrent interactions. The CIT consists of sets of concurrent interactions with dependent effects, policies for resolving them and conditions under which the policies are applicable. Concurrent interactions that are independent of one another can be resolved by serialization and are not specified in the CIT. Some interactions may be independent because they affect disjoint sets of attributes. Other interactions may be independent because their effects are applied in different time-steps, for example, interactions sent and received by an entity. Yet other interactions are independent because they are request-response pairs. Policies must be specified in the CIT for only the remaining interactions. An Interaction Resolver for the Marcus MRE applies the policies in the CIT only if the effects of concurrent interactions conflict. If concurrent interactions do not conflict, they may be serialized.

TABLE 42: Concurrent Interactions Table for Marcus MRE

Concurrent Interactions	Condition	Policy
Type 0, Type 1		Ignore Type 1
Type 2, Type 3		Delay Type 3
Any Interaction	Ignored or Delayed	Ignored or Delayed entirely, i.e., no partial effects permitted

E.7 Construct a Consistency Enforcer and an Interaction Resolver

A Consistency Enforcer (CE) and an Interaction Resolver (IR) for an MRE maintain consistency and resolve concurrent interactions respectively. A CE consists of an ADG and mapping functions, whereas an IR consists of policies for resolving concurrent interactions. Figure 77 shows an MRE for Marcus. The MRE can interact at multiple representation levels — the planner and PA levels — concurrently. Moreover, the concurrent representations within the MRE are consistent at all observation times.

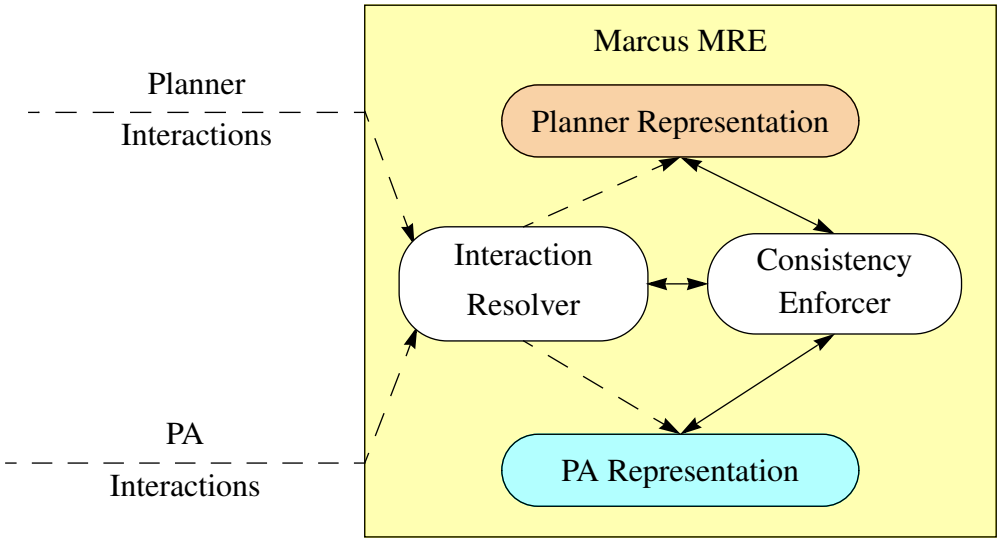


FIGURE 77: Marcus MRE

A CE consists of an ADG and application-specific mapping functions. We presented an ADG for Marcus in Figure 76 and mapping functions in Table 40. In Figure 34 (see Chapter 6), we presented an algorithm for implementing a CE. In §6.3, we discussed how to traverse an ADG and apply mapping functions to keep an MRE internally consistent.

An IR consists of application-specific policies for resolving the effects of concurrent interactions. For the Marcus MRE, we presented policies for resolving concurrent

interactions in Table 42. In Figure 47 (see Chapter 7), we presented an algorithm for implementing an IR. In §7.5, we presented a taxonomy for classifying interactions. Using this taxonomy, we presented policies for resolving the effects of concurrent interactions.

A CE and an IR ensure that an MRE is internally consistent when concurrent interactions occur. During a time-step, a number of concurrent interactions may occur. The IR determines the type of each interaction. Next, the IR applies the effect of each interaction as if the interaction occurred in isolation. In order to do so, the IR permits the interactions to take effect one at a time. When an interaction changes an attribute, the CE traverses an ADG and translates changes to dependent attributes by invoking the appropriate mapping functions. The CE maintains a list of changes for each attribute as a result of computing the effects of each interaction. Subsequently, the CE applies the effects of all the interactions on each attribute. The CE queries the IR about policies to resolve the effects of dependent concurrent interactions whenever the CE detects conflicts in the list of changes for an entity. If the IR contains a policy for resolving conflicting changes, the CE applies the changes accordingly; otherwise, the CE assumes the changes are independent and applies them in an arbitrary order. When the changes to all attributes have been applied, the MRE is internally consistent.

They do certainly give very strange and new-fangled names to diseases.
— *Plato,* The Republic

Indexed Glossary*

Aggregate Model **41**

A model at low resolution or high abstraction.

Aggregation 22, **42**

Composition of a collection of HREs into a single LRE.

Aggregation-disaggregation 9, **20**, 54, 76, 85, 97, 172

An MRM approach in which representation levels are transitioned.

Attribute **33**

A property of an entity, which can be used to refer to the entity and manipulate its behavior.

* In the spirit of UNIFY, this chapter is a glossary as well as an index. The numbers to the right of each term denote pages in which the term is discussed. The bold number refers to the page in which we define the term. Below every index entry is an informal definition for the term.

Attribute Dependency Graph 7, 27, **96**, 172

A graph with attributes as nodes and dependencies among attributes as arcs.

Behavior of an Entity **37**

The sequence of states for a particular entity.

Behavior of a Model **37**

The sequence of states of a model.

Certain Interaction **142**, 149

An interaction whose outcome is predictable.

Chain Disaggregation 23, **56**, 86, 197

Forcible disaggregation of many entities because of LRE-HRE interactions.

Compatible Time-steps **44**, 51, 68, 159, 192

If multiple entities never violate any assumptions made by one other during any time-step, they execute at *compatible time-steps*.

Concurrent Interactions 7, **39**, 51, 81, 136

Interactions that occur at overlapping times in the simulation.

Concurrent Representations 2, **42**, 71

Representations of different simulation entities of the same object or process that execute jointly and allow interaction at all representation levels.

Consistency Cost 9, 28, 76, **174**, 177

Cost of maintaining consistency among jointly-executing models.

Consistency Enforcer 6, 27, **77**, 95, 118, 172

Consists of an Attribute Dependency Graph and appropriate mapping functions for maintaining internal consistency in a Multiple Representation Entity.

Consistency Maintenance 2, **43**

Correlating the multiple entity states for the same object or process so that relationships among attributes hold.

Cost-effectiveness (R3) **9**, 49, 173

Simulation and consistency costs should be low.

Cross-level Interactions **58,** 73, 87

Interactions whose sender and receiver are at different representation levels.

Cumulative Dependencies 96, **102**, 106

Attribute dependencies wherein the value of a single attribute depends jointly on the value of many other attributes.

Dependency **37**

A static relationship between two attributes.

Dependent Interactions 7, **39**, 64, 70, 136

Interactions whose effects are dependent on one another.

Disaggregate Model **42**

A model at high resolution or low abstraction.

Disaggregation 22, **42**

Decomposition of an LRE into its constituent HREs.

Distributive Dependencies 96, **102**, 107

Attribute dependencies wherein the value of a single attribute influences the value of many other attributes jointly.

Effective Joint Execution 2, 28, **46**, 70, 170

The joint execution of multiple models that satisfies the requirements of multi-representation interaction, multi-representation consistency and cost-effectiveness.

Effects of an Interaction **33**

The changes caused in the representations of the sender and receivers because of the interaction.

Entity 3, **32**

A description of an object or process in a simulation.

Entity Representation **32**

A collection of the attributes of one entity described using the notation of data structures.

Environment of a Model **33**

Objects and processes external to a model.

Executing a Model **36**

Simulating the objects and processes that are part of a phenomenon.

Execution of a Multi-model 3

The joint execution of multiple models.

Fundamental Observations 5, 53

Observations that relate the causes of problems in jointly-executing models to a failure in maintaining consistency among the model representations.

Ghosting **71**

With multiple models, executing *only one model* and reflecting changes from that model to other models.

Guidelines for Designers of Multi-models 10, 161

With multiple models, executing *only one model* and reflecting changes from that model to other models.

Hierarchical Models 103

Models that bear a relationship of being the composition-decomposition or abstraction-refinement of one another.

High Resolution Entity (HRE) 22, **42**, 54, 86

An entity at a low level of abstraction, or high decomposition.

Independent Interactions **39**, 144

Interactions whose effects are the same whether they occur in isolation or concurrently.

Interaction 6, 33, **37**, 126

The means by which entities exchange information or influence one another.

Interaction Dependencies 96, **103**

Attribute dependencies that denote the effects of interactions.

Interaction Resolver 8, 28, **77**, 150

Resolves the effects of concurrent interactions on a Multiple Representation Entity by means of policies based on semantic characteristics of interactions.

Internal Consistency with an MRE **71**

Attribute dependencies that denote the effects of interactions.

Joint Execution of Multiple Models 3, **42**

Execution of multiple models at overlapping times, possibly with the exchange of information among the models.

Low Resolution Entity (LRE) 22, **42**, 54, 86

An entity at a high level of abstraction, or high composition.

Mapping Consistency **8**, 80

When entity properties common to different models are translated such that repeated translations in a given period do not cause abnormal behavior in the entity during that period, the models exhibit mapping consistency.

Mapping Inconsistency **55**, 76, 84, 197

When repeated translations among attributes cause intolerable discontinuities in the behavior of different models, the models exhibit mapping inconsistency.

Mapping Functions 7, 27, **43**, 50, 70, 110, 159, 172

Methods used to correlate the multiple representations of the same object or process.

Model 3, **36**

An abstraction of some phenomenon that incorporates the behavior of objects and processes participating in that phenomenon.

Modeling 3, **32**

A method to study real-life phenomena.

Modeling Dependencies 96, 102, **103**

Attribute dependencies inherent in the nature of the object or process being modeled.

Multi-model 3, 41, **42**

For some phenomenon, the union of multiple models that may differ in execution and representation.

Multiple Representation Entity (MRE) 6, 27, **70**, 92, 95

A *conceptual* entity that can interact at multiple representation levels concurrently by maintaining attributes at different representation levels.

Multi-representation Consistency (R2) **8**, 48, 172

Jointly-executing models must be related and consistent with one another.

Multi-representation Interaction (R1) **8**, 47, 76, 171

Entities in each model may initiate and receive interactions that may be concurrent and dependent.

Multi-representation Modeling (MRM) 3, 32, **42**, 159

The joint execution of multiple models.

Network Flooding **57**, 87, 197

High consumption of network resources because of a large number of messages.

Receiver of an Interaction **33**

The entity that receives an interaction.

Relationship between Attributes **33**

Indicates that if the value of one attribute changes, the value of the other is likely to change.

Representation 3, **32**, 191

A collection of the objects and processes participating in a phenomenon described using some rigorous notation.

Representation Level **41**

The conceptual context in which a model executes.

Request Interaction **141**, 146, 149

An interaction concerned with actions that may take place in the future.

Resolving Concurrent Interactions **40**, 62, 81, 93, 144, 148, 159, 192

Computing the effects of concurrent, possibly dependent, interactions.

Resolution **41**

A representation level in a hierarchical model.

Resolution Level **41**

Resolution.

Response Interaction **141**, 146, 149

An interaction concerned with actions that have taken place in the past.

Reversible Mapping Functions **44**, 80, 83, 115

Mapping functions that return the original attribute when composed.

Selective Viewing 9, **20**, 76, 85, 97, 172

An MRM approach in which the most detailed model is simulated at all times.

Sender of an Interaction **33**

The entity that initiates an interaction.

Simulation 3, **37**

A method to execute a model, usually on a computer with some combination of executing code, control/display interface hardware and interfaces to real-world equipment.

Simulation Cost 9, 28, 76, **174**, 179

Cost of simulating jointly-executing models.

Taxonomy of Interactions 7, 28, **139**

A classification of interactions according to semantic characteristics of interactions.

Temporal Consistency **8**, 78

When multiple entities have consistent views of another entity at overlapping simulation times, the entities exhibit temporal consistency.

Temporal Inconsistency 91, 197

When multiple entities have differing views of an entity at overlapping simulation times, they exhibit temporal inconsistency.

Thrashing **57**, 86, 197

Repeated transitions by an entity because it transitions representation levels frequently.

Time-step **36**, 65

The duration of time between two consecutive observation times of a model.

Transition Latency **56**, 86, 197

Delay encountered when performing an aggregation or disaggregation.

Uncertain Interaction **142**, 149

An interaction whose outcome is unpredictable.

`UNIFY` **8**, 5, 47, 54, 70, 159, 191

A framework for effective MRM.

QUOTATION, n.
The act of repeating erroneously the words of another. The words erroneously repeated.
— *Ambrose Bierce,* The Devil's Dictionary

References

ABADI95 Abadi, M., Lamport, L., *Conjoining Specifications*, ACM Transactions on Programming Languages and Systems, Vol. 17, No. 3, May 1995.

ACK82 Ackerman, W. B., *Data Flow Languages*, IEEE Computer, Vol. 15, No. 2, February 1982.

AGRE87 Agre, P. E., Chapman, D., *Pengi: An Implementation of a Theory of Activity*, American Association for Artifical Intelligence Conference, 1987.

ALBUS97 Albus, J. S., *The NIST Real-time Control System (RCS): an approach to intelligent systems research*, Journal of Experimental and Theoretical Artificial Intelligence, Vol. 9, 1997.

ALHIR98 Alhir, S. S., *UML in a Nutshell*, O'Reilly & Associates Inc., ISBN 1-56592-448-7, 1998.

ALLEN92 Allen, P. D., *Combining Deterministic and Stochastic Elements in Variable Resolution Models*, Conference on Variable-Resolution Modeling, Washington, DC, May 1992.

ALLEN96 Allen, P. M., Valle, A. N., *An approach to managing dissimilar unit interactions in constructive/virtual simulation linkage*, 14th DIS Workshop on Standards for the Interoperability of Distributed Simulations, Orlando, Florida, September 1996.

ALLEN98 Allen, R. J., Garlan, D., Ivers, J., *Formal Modeling and Analysis of the HLA Component Integration Standard*, ACM SIGSOFT, Florida, November 1998.

AMG95 Architecture Management Group, *Preliminary Definition*, High Level Architecture Briefings, Defense Modeling and Simulation Office (DMSO), Alexandria, Virginia, March 1995.

AMO94 Amoroso, E. D., *Fundamentals of Computer Security Technology*, Prentice Hall PTR, ISBN 0-13-108929-3, 1994.

ARCH86 Archibald, J., Baer, J. L., *Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model*, ACM Transactions on Computer Systems, Vol. 4, No. 4, November 1986.

ASTRA76 Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., Eswaran, K. P., Gray, J. N., Griffiths, P. P., King, W. F., Lorie, R. A., McJones, P. R., Mehl, J. W., Putzolu, G. R., Traiger, I. L., Wade, B. W., Watson, V., *System R: Relational Approach to Database Management*, ACM Transactions on Database Systems, Vol. 1, No. 2, June 1976.

BADRI92 Badrinath, B. R., Ramamritham, K., *Semantics-Based Concurrency Control: Beyond Commutativity*, ACM Transactions on Database Systems, Vol. 17, No. 1, March 1992.

BALZER85 Balzer, R., *Automated Enhancement of Knowledge Representations*, International Joint Conference on Artifical Intelligence, 1985.

BAN81 Bancilhon, F., Spyratos, N., *Update Semantics of Relational Views*, ACM Transactions on Database Systems, Vol. 6, No. 4, December 1981.

BARG91 Barghouti, N. S., Kaiser, G. E., *Concurrency Control in Advanced Database Applications*, ACM Computing Surveys, Vol. 23, No. 3, September 1991.

BARNES80 Barnes, J. G. P., *An Overview of ADA*, Software Practice and Experience, Vol. 10, 1980.

BERM94 Berman, D. F., Bartell, J. T., Salesin, D. H., *Multiresolution Painting and Compositing*, ACM Computer Graphics Proceedings Annual Conference Series, 1994.

BERN81 Bernstein, P. A., Goodman, N., *Concurrency Control in Distributed Database Systems*, ACM Computing Surveys, Vol. 13, No. 2, June 1981.

BERN87 Bernstein, P. A., Hadzilacos, V., Goodman, N., *Concurrency Control and Recovery in Database Systems*, Addison Wesley Publishing Company Inc., ISBN 0-201-10715-5, 1987.

BESH85 Beshers, G., Campbell, R., *Maintained and Constructor Attributes*, ACM SIGPLAN 85 Symposium on Language Issues in Programming Environments, June 1985.

BIRT73 Birtwistle, G. M., Dahl, O-J, Myhrhaug, B. Nygaard, K., *Simula Begin*, Studentlitteratur and Auerbach Publishers, ISBN 91-44-06211-7, 1973.

BON97 Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D. P., Slack, M. G., *Experiences with an Architecture for Intelligent Reactive Agents*, Journal of Experimental and Theoretical Artificial Intelligence, Vol. 9, No. 2, 1997.

BORN82 Borning, A. H., Ingalls, D. H. H., *Multiple Inheritance in Smalltalk-80*, American Association for Artifical Intelligence Conference, August 1982.

BRINCH78 Brinch Hansen, P., *Distributed Processes: A Concurrent Programming Concept*, Communications of the ACM, Vol. 21, No. 11, November 1978.

BRAHMA90 Brahmadathan, K., Ramarao, K. V. S., *On the Management of Long-Living Transactions*, Journal of Systems Software, Vol. 11, 1990.

BRILL96 Brill, F., *Representation of Local Space in Perception/Action Systems: Behaving Appropriately in Difficult Situations*, Ph.D. Dissertation, Department of Computer Science, University of Virginia, 1996.

BRILL98 Brill, F., Wasson, G., Ferrer, G., Martin W., *The Effective Field of View Paradigm: Adding Representation to a Reactive System*, Engineering Applications of Artificial Intelligence issue on Machine Vision for Intelligent Vehicles and Autonomous Robots, Vol. 11, 1998.

BROOKS86 Brooks, R. A., *A Robust Layered Control System For A Mobile Robot*, IEEE Journal of Robotics and Automation, Vol. RA-2, No. 1, March 1986.

BURD95 Burdick, C. D., *Interoperability of Simulations with Different Levels of Resolution*, Defense Modeling and Simulation Office (DMSO) Workshop, Alexandria, Virginia, November 1995.

BURNS98 Burns, A., Prasad, D., Bondavalli, A., Di Giandomenico, F., Ramamritham, K., Stankovic, J., Strigini, L., *The Meaning and Role of Value in Scheduling Flexible Real-Time Systems*, Technical Report CS-98-29, Department of Computer Science, University of Virginia, 1998.

CALD95A Calder, R. B., Peacock, J. C., Panagos, J., Johnson, T. E., *Integration of Constructive, Virtual, Live, and Engineering Simulations in the JPSD CLCGF*, 5th Conference on Computer Generated Forces & Behavioral Representation, Orlando, Florida, May 1995.

CALD95B Calder, R. B., Peacock, J. C., Wise, B. P. Jr., Stanzione, T., Chamberlain, F., Panagos, J., *Implementation of a Dynamic Aggregation/Deaggregation Process in the JPSD CLCGF*, 5th Conference on Computer Generated Forces & Behavioral Representation, Orlando, Florida, May 1995.

CARD85 Cardelli, L., Wegner, P., *On Undertanding Types, Data Abstractions, and Polymorphism*, ACM Computing Surveys, Vol. 17, No. 4, December 1985.

CHAM75 Chamberlin, D. D., Gray, J. N., Traiger, I. L., *Views, Authorization, and Locking in a Relational Database System*, American Federation of Information Processing Societies Conference, 1975.

CHEN76 Chen, P. P., *The Entity-Relationship Model — Toward a Unified View of Data*, ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976.

CLARK76 Clark, J. H., *Hierarchical Geometric Models for Visible Surface Algorithms*, Communications of the ACM, Vol. 19, No. 10, October 1976.

CLARK94 Clark, K. J., Brewer, D., *Bridging the Gap Between Aggregate Level and Object Level Exercises*, 4th Conference on Computer Generated Forces & Behavioral Representation, Orlando, Florida, May 1994.

CODD70 Codd, E. F., *A Relational Model of Data for Large Shared Data Banks*, Communications of the ACM, Vol. 13, No. 6, June 1970.

CORMEN89 Cormen, T. H., Leiserson, C. E., Rivest, R. L., *Introduction to Algorithms*, MIT Press, ISBN 0-262-03141-8, 1989.

COX95 Cox, A., Maybury, J., Weeden, N., *Aggregation Disaggregation Research — A UK Approach*, 13th DIS Workshop on Standards for the Interoperability of Distributed Simulations, Orlando, Florida, September 1995.

DAH95 Dahmann, J., Wood, D. C., *editors*, Special Issue of IEEE Distributed Interactive Simulation, Vol. 83, No. 8, August 1995.

DAHL66 Dahl, O-J., Nygaard, K., *Simula — An Algol-Based Simulation Language*, Communications of the ACM, Vol. 9, No. 9, September 1966.

DATE95 Date, C. J., *An Introduction to Database Systems (Sixth Edition)*, Addison Wesley Publishing Company Inc., ISBN 0-201-54329-X, 1995.

DAVIS82 Davis, A. L., Keller, R. M., *Data Flow Program Graphs*, IEEE Computer, Vol. 15, No. 2, February 1982.

DAVIS92 Davis, P. K., *An Introduction to Variable-Resolution Modeling and Cross-Resolution Model Connection*, Conference on Variable-Resolution Modeling, Washington, DC, May 1992.

DAVIS93 Davis, P. K., Hillestad, R. J., *Families of Models that Cross Levels of Resolution: Issues for Design, Calibration and Management*, Winter Simulation Conference, 1993.

DAVIS98 Davis, P. K., Bigelow, J. H., *Experiments in Multiresolution Modeling (MRM)*, Prepared for the Defense Advanced Research Projects Agency by RAND's National Defense Research Institute, ISBN 0-8330-2653-4, 1998.

DEMERS85 Demers, A., Rogers, A., Zadeck, F. K., *Attribute Propagation by Message Passing*, ACM SIGPLAN 85 Symposium on Language Issues in Programming Environments, June 1985.

DENNIS80 Dennis, J. B., *Data Flow Supercomputers*, IEEE Computer, Vol. 13, No. 1, November 1980.

DIS93 DIS Steering Committee, *The DIS Vision, A Map to the Future of Distributed Simulation*, Comment Draft, October 1993.

DOD94 Under Secretary of Defense (Acquisition and Technology), *Modeling and Simulation (M&S) Master Plan*, Dept. of Defense, September 1994.

EPST85 Epstein, J. M., *The Calculus of Conventional War: Dynamic Analysis Without Lanchester Theory*, The Brookings Institute, 1985.

ERMAN80 Erman, L. D., Hayes-Roth, F., Lesser, V. R., Reddy, D. R., *The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty*, ACM Computing Surveys, Vol. 12, No. 2, June 1980.

ESWA76 Eswaran, K. P., Gray, J. N., Lorie, R. A., Traiger, I. L., *The Notions of Consistency and Predicate Locks in a Database System*, Communications of the ACM, Vol. 19, No. 11, November 1976.

FIRBY87 Firby, R. J., *An Investigation into Reactive Planning in Complex Domains*, American Association for Artifical Intelligence Conference, 1987.

FOWLER97 Fowler, M., Scott, K., *UML Distilled*, Addison Wesley Longman Inc., ISBN 0-201-32563-2, 1997.

FRANCE93 Franceschini, R. W., *Intelligent Placement of Disaggregated Entities*, Institute for Simulation and Training, 1993.

FREE90 Freeman-Benson, B. N., Maloney, J., Borning, A., *An Incremental Constraint Solver*, Communications of the ACM, Vol. 33, No. 1, January 1990.

FRÜH92A Frühwirth, T., Herold, A., Küchenhoff, V., Le Provost, T., Lim, P., Monfroy, E., Wallace, M., *Constraint Logic Programming — An Informal Introduction*, Logic Programming in Action, LNCS 636, Springer-Verlag, 1992, Technical Report ECRC-92-6i, European Computer-Industry Research Centre, July 1992.

FRÜH92B Frühwirth, T., *Constraint Simplification Rules*, Technical Report ECRC-92-18, European Computer-Industry Research Centre, July 1992.

FUJI90 Fujimoto, R. M., *Parallel Discrete Event Simulation*, Communications of the ACM, Vol. 33, No. 10, October 1990.

GAJSKI82 Gajski, D. D., Padua, D. A., Kuck, D. J., Kuhn, R. H., *A Second Opinion on Data Flow Machines and Languages*, IEEE Computer, Vol. 15, No. 2, February 1982.

GAR87 Garlan, D., *Views for Tools in Integrated Environments*, Ph.D. Dissertation, Technical Report CMU-CS-87-147, School of Computer Science, Carnegie Mellon University, 1987.

GAR95 Garland, M., Heckbert. P. S., *Fast Polygonal Approximations of Terrains and Height Fields*, Technical Report CMU-CS-95-181, School of Computer Science, Carnegie Mellon University, September 1995.

GARCIA83 Garcia-Molina, H., *Using Semantic Knowledge for Transaction Processing in a Distributed Database*, ACM Transactions on Database Systems, Vol. 8, No. 2, June 1983.

GARCÍA93 García de la Banda, M., Hermenegildo, M., Marriott, K., *Independence in Constraint Logic Programs*, International Logic Programming Symposium, MIT Press, 1993.

GAT92 Gat, E., *Integrating Planning and Execution in a Heterogeneous Asychronous Architecture for Controlling Real-World Mobile Robots*, American Association for Artifical Intelligence Conference, 1992.

GIORGI90 Giorgi, F., *Simulation of Regional Climate Using a Limited Area Model Nested in a General Circulation Model*, Journal of Climate, Vol. 3, September 1990.

GIORGI91 Giorgi, F., Mearns, L. O., *Approaches to the Simulation of Regional Climate: A Review*, Reviews of Geophysics, Vol. 29, No. 2, May 1991.

GOLD80 Goldstein, I. P., Bobrow, D. G., *Descriptions for a Programming Environment*, First Annual Conference of the National Association for Artificial Intelligence, August 1980, Xerox Technical Report CSL-81-3.

GOOD75 Goodenough, J. B., *Exception Handling: Issues and a Proposed Notation*, Communications of the ACM, Vol. 18, No. 12, December 1975.

GRIM93 Grimshaw, A. S., Strayer, W. T., Narayan P., *Dynamic, Object-Oriented Parallel Processing*, IEEE Parallel and Distributed Technology, May 1993.

HAER83 Haerder, T., Reuter, A., *Principles of Transaction-Oriented Database Recovery*, ACM Computing Surveys, Vol. 15, No. 4, December 1983.

HANKS90 Hanks, S., Firby, R. J., *Issues and Architectures for Planning and Execution*, DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control, San Mateo, CA, November 1990.

HARDY94 Hardy, D., Healy, M., *Constructive & Virtual Interoperation: A Technical Challenge*, 4th Conference on Computer Generated Forces & Behavioral Representation, Orlando, Florida, May 1994.

HARSH92 Harshberger, E. R., Bennett, B. E., Frelinger, D. R., *An Approach to Hierarchies of Models: Process Independence*, Conference on Variable-Resolution Modeling, Washington, DC, May 1992.

HECK94 Heckbert, P. S., Garland, M., *Multiresolution Modeling for Fast Rendering*, Graphics Interface, Banff, Canada, May 1994.

HECK97 Heckbert, P. S., Garland, M., *Survey of Polygonal Surface Simplification Algorithms*, Multiresolution Surface Modeling Course, ACM Computer Graphics Proceedings Annual Conference Series, May 1997.

HENN96 Hennessey, J. L., Patterson, D. A., *Computer Architecture: A Quantitative Approach (Second Edition)*, Morgan Kaufmann Publishers, ISBN 1-55860-329-8, 1996.

HILL92A Hillestad, R. J., Juncosa, M. L., *Cutting Some Trees to See the Forest: On Aggregation and Disaggregation in Combat Models*, Conference on Variable-Resolution Modeling, Washington, DC, May 1992, Naval Research Logistics, Vol. 42, 1995.

HILL92B Hillestad, R. J., Owen, J., Blumenthal, D., *Experiments in Variable Resolution Combat Modeling*, Conference on Variable-Resolution Modeling, Washington, DC, May 1992, Naval Research Logistics, Vol. 42, 1995.

HOFER95 Hofer, R. C., Loper, M. L., *DIS Today*, Proceedings of the IEEE, Vol. 83, No. 8, August 1995.

HOP79 Hopcroft, J. E., Ullman, J. D., *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley Publishing Company Inc., ISBN 0-201-02988-X, 1979.

HOR86 Horwitz, S., Teitelbaum, T., *Generating Editing Environments Based on Relations and Attributes*, ACM Transactions on Programming Languages and Systems, Vol. 8, No. 4, October 1986.

HORR92 Horrigan, T. J., *The "Configuration Problem" and Challenges for Aggregation*, Conference on Variable-Resolution Modeling, Washington, DC, May 1992.

HOW97 Howard, J. D., *An Analysis of Security Incidents on the Internet 1989-1995*, Ph.D. Dissertation, Engineering and Public Policy, Carnegie Mellon University, 1997.

HUMBEL96 Humbel, S., Sieber, S., Morokuma, K., *The IMOMO method: Integration of different levels of molecular orbital approximations for geometry optimization of large systems: Test for* n*-butane confirmation and* S_N2 *reaction: RCl+Cl*$^-$, Journal of Chemical Physics, Vol. 105, No. 5, August 1996.

JAFFAR92 Jaffar, J., Michaylov, S., Stuckey, P., Yap, R. H. C., *The CLP($\Re$) Language and System*, ACM Transactions on Programming Languages and Systems, Vol. 14, No. 3, July 1992.

JAFFAR94 Jaffar, J., Maher, M. J., *Constraint Logic Programming: A Survey*, Journal of Logic Programming, Vol. 19, May 1994.

JEFF85 Jefferson, D. R., *Virtual Time*, ACM Transactions on Programming Languages and Systems, Vol. 7, No. 3, July 1985.

JPSD97 —, *Federate Object Model for Joint Precision Strike Demonstration*, OMT v1.3, 1997.

JTFP97 —, *Federate Object Model for Joint Task Force Prototype*, OMT v1.3, 1997.

KARR83 Karr, A. F., *Lanchester Attrition Processes and Theater-Level Combat Models*, Mathematics of Conflict, Elsevier Science Publishers B.V. (North-Holland), ISBN: 0 444 86678 7, 1983.

KARR94 Karr, C. R., Root, E., *Integrating Aggregate and Vehicle Level Simulations*, 4th Conference on Computer Generated Forces & Behavioral Representation, Orlando, Florida, 1994.

KERN88 Kernighan, B. W., Ritchie, D. M., *The C Programming Language (Second Edition)*, Prentice Hall Inc., ISBN 0-13-110370-9, 1988.

KNUTH68 Knuth, D. E., *Semantics of Context-free Languages*, Mathematical Systems Theory, Vol. 2, No. 2, June 1968.

KNUTH71 Knuth, D. E., *Semantics of Context-free Languages: Correction*, Mathematical Systems Theory, Vol. 5, No. 1, March 1971.

KORTH88 Korth, H. F., Speegle, G. D., *Formal Model of Correctness without Serializability*, ACM SIGMOD Record, Vol. 17, No. 3, September 1988.

LAIRD91 Laird, J., Yager, E., Hucka, M., Tuck, C., *Robo-Soar: An Integration of External Interaction, Planning, and Learning using Soar*, Robotics and Autonomous Systems, Vol. 8, 1991.

LAM78 Lamport, L., *Time, Clocks, and the Ordering of Events in a Distributed System*, Communications of the ACM, Vol. 21, No. 7, July 1978.

LAM94 Lamport, L., *The Temporal Logic of Actions*, ACM Transactions on Programming Languages and Systems, Vol. 16, No. 3, May 1994.

LEE98 Lee, A. W. F., Sweldens, W., Schröder, P., Cowsar, L., Dobkin, D., *MAPS: Multiresolution Adaptive Parameterization of Surfaces*, Computer Graphics Proceedings Annual Conference Series, 1998.

LINTON84 Linton, M. A., *Implementing Relational Views of Programs*, ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, April 1984.

LISKOV79 Liskov, B. H., Snyder, A., *Exception Handling in CLU*, IEEE Transactions on Software Engineering, Vol. SE-5, No. 6, November 1979.

LUEBKE97 Luebke, D., *Survey of Polygonal Simplification Algorithms*, Technical Report TR97-045, Department of Computer Science, University of North Carolina, 1997.

LYNCH83 Lynch, N. A., *Multilevel atomicity: a new correctness criterion for database concurrency control*, ACM Transactions on Database Systems, Vol. 8, No. 4, December 1983.

MAD74 Madnick, S. E., Donovan, J. J, *Operating Systems*, McGraw-Hill Inc., ISBN 0-07-039455-5, 1974.

MARR93 Marriott, K., Stuckey, P. J., *The 3 R's of Optimizing Constraint Logic Programs: Refinement, Removal and Reordering*, 20th Annual ACM Symposium on Principles of Programming Languages, 1993.

MATSU96 Matsubara, T., Maseras, F., Koga, N., Morokuma, K., *Application of the New "Integrated MO + MM" (IMOMM) Method to the Organometallic Reaction* $Pt(PR_3)_2 + H_2$ *(R = H, Me,* t-*Bu, and Ph)*, Journal of Physical Chemistry, Vol. 100, No. 7, 1996.

MATT89 Mattern, F., *Virtual Time and Global States of Distributed Systems*, Parallel and Distributed Algorithms, Elsevier Science Publishers B.V. (North-Holland), 1989.

MILLER95 Miller, D. C., Thorpe, J. A., *SIMNET: The Advent of Simulator Networking*, Proceedings of the IEEE, Vol. 83, No. 8, August 1995.

MUNSON96 Munson, J., Dewan, P., *A Concurrency Control Framework for Collaborative Systems*, ACM Conference on Computer Supported Cooperative Work, 1996.

NAT95 Natrajan, A., Nguyen-Tuong, A., *To disaggregate or not to disaggregate, that is not the question*, ELECSIM, Internet, April-June, 1995, Technical Report CS-95-18, Department of Computer Science, University of Virginia, 1995.

NAT96 Natrajan, A., Reynolds Jr., P. F., Srinivasan, S., *Consistency Maintenance using* `UNIFY`, Technical Report CS-95-28, Department of Computer Science, University of Virginia, 1996, Part of Grant Proposal to DMSO, 1995-1996.

NAT97 Natrajan, A., Reynolds Jr., P. F., Srinivasan, S., *A Flexible Approach to Multi-Resolution Modeling*, Parallel and Distributed Simulation, June 1997.

NAT99 Natrajan, A., Reynolds Jr., P. F., *Resolving Concurrent Interactions*, 3rd International Workshop on Distributed Interactive Simulation and Real Time Applications, October 1999.

NRC97 Committee on Modeling and Simulation: Opportunities for Collaboration between the Defense and Entertainment Research Communities, *Modeling and Simulation: Linking Entertainment and Defense*, National Research Council, October 1993.

OMT98 U.S. Department of Defense, *High Level Architecture Object Model Template Specification Version 1.3*, IEEE P1516.2, Standard for Modeling and Simulation, April 1998.

PAPA86 Papadimitriou, C. H., *The Theory of Database Concurrency Control*, Computer Science Press, ISBN 0-88175-027-1, 1986.

PETER77 Peterson, J. L., *Petri Nets*, ACM Computing Surveys, Vol. 9, No. 3, September 1977.

PETRI62 Petri, C. A., *Kommunikation mit Automaten*, Ph.D. dissertation, Schriften des Rheinisch-Westfalischen Institutes für Instrumentelle Mathematik an der Universitat Bonn, 1962.

PETTY94 Petty, M. D., *The Turing Test as an Evaluation Criterion for Computer Generated Forces*, 4th Conference on Computer Generated Forces & Behavioral Representation, Orlando, Florida, May 1994.

PETTY95 Petty, M. D., Franceschini, R. W., *Disaggregation Overload and Spreading Disaggregation in Constructive+Virtual Linkages*, 5th Conference on Computer Generated Forces & Behavioral Representation, Orlando, Florida, May 1995.

PRATT95 Pratt, D. R., Johnson, M. A., *Constructive and Virtual Model Linkage*, Winter Simulation Conference, 1995.

PULLEN95 Pullen, J. M., Wood, D. C., *Networking Technology and DIS*, Proceedings of the IEEE, Vol. 83, No. 8, August 1995.

PUPPO97 Puppo, E., Scopigno, R., *Simplification. LOD and Multiresolution Principles and Applications*, Eurographics, Vol. 16, No. 3, 1997.

REDDY95 Reddy, R., Garrett, R., *Future Technology Challenges in Distributed Interactive Simulation*, Proceedings of the IEEE, Vol. 83, No. 8, August 1995.

REPS84 Reps, T., Teitelbaum, T., *The Synthesizer Generator*, ACM SIGSOFT/ SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Software Engineering Notes, Vol. 9, No. 3, SIGPLAN Notices, Vol. 19, No. 5, May 1984.

REPS86 Reps, T., Marceau, C., Teitelbaum, T., *Remote attribute updating for language-based editors*, 13th Annual ACM Symposium on Principles of Programming Languages, 1986.

REYN94 Reynolds Jr., P. F., *DISorientation*, ELECSIM 94, Internet, April-June, 1994.

REYN97 Reynolds Jr., P. F., Natrajan, A., Srinivasan, S., *Consistency Maintenance in Multi-Resolution Simulations*, ACM Transactions on Modeling and Computer Simulation, Vol. 7, No. 3, July 1997.

RISBEY96 Risbey, J. S., Stone, P. H., *A Case Study of the Adequacy of GCM Simulations for Input to Regional Climate Change Assessments*, Journal of Climate, Vol. 9, July 1996.

ROBKIN92 Robkin, M., *A proposal to Modify the Distributed Interactive Simulation Aggregate PDU*, Hughes Training Inc., February 1992.

ROSSER82 Rosser, J. B., *Highlights of the history of the lambda-calculus*, Conference Record of 1982 ACM Symposium on Lisp and Functional Programming, 1992.

RPR97 —, *Federate Object Model for Real-time Platform Reference*, OMT v1.3, September 1997.

RUM91 Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., *Object-Oriented Modeling and Design*, Prentice Hall PTR, ISBN 0-13-629841-9, 1991.

SACER74 Sacerdoti, E. D., *Planning in a Hierarchy of Abstraction Spaces*, Artificial Intelligence, Vol. 5, 1974.

SARAS91 Saraswat, V. A., Rinard, M., Panangaden, P., *Semantic Foundations of concurrent constraint programming*, 18th Annual ACM Symposium on Principles of Programming Languages, 1991.

SEIDEL95 Seidel, D. W., King, B. C., Burke, C. D., *AIM Approach to Simulation Interoperability,* The MITRE Corporation, Preliminary Draft, July 1995.

SHER92 Sherman, R., Butler, B., *Segmenting the Battlefield*, Loral WDL, June 1992.

SHLAER92 Shlaer, S., Mellor, S. J., *Object Lifecycles: Modeling the World in States*, Prentice Hall PTR, ISBN 0-13-629940-7, 1992.

SILB91 Silberschatz, A., Peterson, J. L., Galvin, P., *Operating System Concepts (Third Edition)*, Addison Wesley Publishing Company Inc., ISBN 0-201-51379-X, 1991.

SIM94 Simmons, R., *Structured Control for Autonomous Robots*, IEEE Transactions on Robotics and Automation, Vol. 10, No. 1, February 1994.

SMITH94 Smith, R. D., *Invited speaker*, Department of Computer Science, University of Virginia, December 1994.

SMITH95 Smith, R. D., *The Conflict Between Heterogeneous Simulation and Interoperability*, 17th Inter-Service/Industry Training, Simulation, and Education Conference (I/ITSEC) Proceedings, November 1995.

STEFIK86 Stefik, M., Bobrow, D. G., *Object-Oriented Programming: Themes and Variations*, AI Magazine, Vol. 6, No. 4, 1986.

STEIN94 Steinman, J. S., Wieland, F., *Parallel Proximity Detection and the Distribution List Algorithm*, 1994.

STOBER95 Stober, D. R., Kraus, M. K., Foss, W. F., Franceschini, R. W., Petty, M. D., *Survey of Constructive+Virtual Linkages*, 5th Conference on Computer Generated Forces & Behavioral Representation, Orlando, Florida, May 1995.

STONE76 Stonebraker, M. R., Wong, E., Kreps, P., Held, G., *The Design and Implementation of Ingres*, ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976.

STROU91 Stroustrup, B., *The C++ Programming Language (Second Edition)*, Addison Wesley Publishing Company Inc., ISBN 0-201-53992-6, 1991.

SULL94 Sullivan, K. J., *Mediators: Easing the Design and Evolution of Integrated Systems*, Ph.D. Dissertation, Technical Report 94-08-01, Department of Computer Science and Engineering, University of Washington, 1984.

SVEN96A Svensson, M., Humbel, S., Froese, R. D. J., Matsubara, T., Sieber, S., Morokuma, K., *ONIOM: A Multilayered Integrated MO + MM Method for Geometry Optimizations and Single Point Energy Predictions. A Test for Diels-Alder Reactions and* $Pt(P(\text{t-}Bu)_3)_2 + H_2$ *Oxidative Addition*, Journal of Physical Chemistry, Vol. 100, No. 50, 1996.

SVEN96B Svensson, M., Humbel, S., Morokuma, K., *Energetics using the single point IMOMO (integrated molecular orbital + molecular orbital) calculations: Choices of computational levels and model system*, Journal of Chemical Physics, Vol. 105, No. 9, September 1996.

TANEN92 Tanenbaum, A. S., *Modern Operating Systems*, Prentice Hall Inc., ISBN 0-13-595752-4, 1992.

TEXEL97 Texel, P. P., Williams, C. B., *Use Cases combined with Booch/OMT/UML: Process and Products*, Prentice Hall PTR, ISBN 0-13-727405-X, 1997.

THOM98 Thomasin, A., *Concurrency Control: Methods, Performances and Analysis*, ACM Computing Surveys, Vol. 30, No. 1, March 1998.

TURING50 Turing, A. M., *Computing Machinery and Intelligence*, Mind, Vol. 59, October 1950.

VAN96 Van Hentenryck, P., Saraswat, V. A., *et al*, *Strategic Directions in Constraint Programming*, ACM Computing Surveys, Vol. 28, No. 4, December 1996.

WAS98A Wasson, G., Martin, W., *Multi-tiered Representation for Autonomous Robots*, SPIE Conference on Mobile Robots and Autonomous Systems, November 1998.

WAS98B Wasson, G. S., Natrajan, A., Gunderson, J. P., Ferrer, G. J., Martin, W. N., Reynolds Jr., P. F., *Consistency Maintenance in Autonomous Agent Representations*, Technical Report CS-98-06, Department of Computer Science, University of Virginia, 1998.

WAS99 Wasson, G. S., *Design of Representation Systems for Autonomous Agents*, Ph.D. Dissertation, Department of Computer Science, University of Virginia, 1999.

WEAT93 Weatherly, R. M., Wilson, A. L., Griffin, S. P., *ALSP — Theory, Experience and Future Directions*, Winter Simulation Conference, 1993.

WEIHL88 Weihl, W. E., *Commutativity-Based Concurrency Control for Abstract Data Types*, IEEE Transactions on Computers, Vol. 37, No. 12, December 1988.

WILL93 Williams, C. C., *Concurrency Control in Asynchronous Computations*, Ph.D. Dissertation, Department of Computer Science, University of Virginia, 1998.

WIM86 Wimsatt, W. C., *Heuristics and the Study of Human Behavior*, in Fiske, D., Shweder, R., eds., Meta-Theory in the Social Sciences: Pluralisms and Subjectives, University of Chicago Press, 1986.

YEMINI85 Yemini, S., Berry, D. M., *A Modular Verifiable Exception-Handling Mechanism*, ACM Transactions on Programming Languages and Systems, Vol. 7, No. 2, April 1985.

ZORIN97 Zorin, D., Schröder, P., Sweldens, W., *Interactive Multiresolution Mesh Editing*, ACM Computer Graphics Proceedings Annual Conference Series, May 1997.