

Winter 2011

# Understanding the Elements of Executable Architectures Through a Multi-Dimensional Analysis Framework

Edwin A. Shuman IV  
*Old Dominion University*

Follow this and additional works at: [https://digitalcommons.odu.edu/msve\\_etds](https://digitalcommons.odu.edu/msve_etds)

Part of the [Computer Sciences Commons](#)

---

## Recommended Citation

Shuman, Edwin A.. "Understanding the Elements of Executable Architectures Through a Multi-Dimensional Analysis Framework" (2011). Doctor of Philosophy (PhD), dissertation, Modeling Simul & Visual Engineering, Old Dominion University, DOI: 10.25777/v5k3-0128  
[https://digitalcommons.odu.edu/msve\\_etds/40](https://digitalcommons.odu.edu/msve_etds/40)

This Dissertation is brought to you for free and open access by the Modeling, Simulation & Visualization Engineering at ODU Digital Commons. It has been accepted for inclusion in Modeling, Simulation & Visualization Engineering Theses & Dissertations by an authorized administrator of ODU Digital Commons. For more information, please contact [digitalcommons@odu.edu](mailto:digitalcommons@odu.edu).

**UNDERSTANDING THE ELEMENTS OF EXECUTABLE ARCHITECTURES  
THROUGH A MULTI-DIMENSIONAL ANALYSIS FRAMEWORK**

by

Edwin A. Shuman IV  
B.A. August 1980, University of Virginia  
M.S. March 1990, Naval Postgraduate School

A Dissertation Submitted to the Faculty of  
Old Dominion University in Partial Fulfillment of the  
Requirement for the Degree of

DOCTOR OF PHILOSOPHY


MODELING AND SIMULATION


OLD DOMINION UNIVERSITY  
December 2011

Approved by:

 Andreas Tolk (Director)

 Frederic D. McKenzie (Member)

Charles Keating  (Member)

 Thomas Pawlowski (Member)

UMI Number: 3492412

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

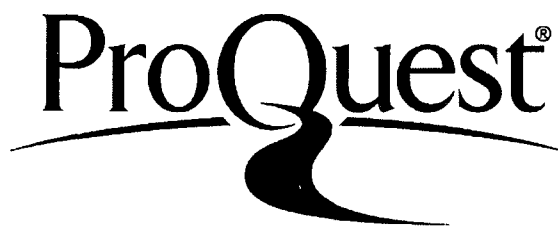
In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI 3492412

Copyright 2012 by ProQuest LLC.

All rights reserved. This edition of the work is protected against unauthorized copying under Title 17, United States Code.



ProQuest LLC  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106-1346

## **ABSTRACT**

### **UNDERSTANDING THE ELEMENTS OF EXECUTABLE ARCHITECTURES THROUGH A MULTI-DIMENSIONAL ANALYSIS FRAMEWORK**

Edwin A. Shuman IV  
Old Dominion University, 2011  
Director: Andreas Tolk

The objective of this dissertation study is to conduct a holistic investigation into the elements of executable architectures. Current research in the field of Executable Architectures has provided valuable solution-specific demonstrations and has also shown the value derived from such an endeavor. However, a common theory underlying their applications has been missing.

This dissertation develops and explores a method for holistically developing an Executable Architecture Specification (EAS), i.e., a meta-model containing both semantic and syntactic information, using a conceptual framework for guiding data coding, analysis, and validation. Utilization of this method resulted in the description of the elements of executable architecture in terms of a set of nine information interrogatives: an executable architecture information ontology. Once the detail-rich EAS was constructed with this ontology, it became possible to define the potential elements of executable architecture through an intermediate level meta-model. The intermediate level meta-model was further refined into an interrogative level meta-model using only the nine information interrogatives, at a very high level of abstraction.

Copyright, 2011, by Edwin A. Shuman, All Rights Reserved.

I dedicate this dissertation to my beloved wife Nancy White Hammonds Shuman.

## ACKNOWLEDGEMENTS

There are many persons who have contributed to the successful completion of this dissertation. I extend many thanks to my parents Ned and Sue Shuman, for their hopeful encouragement, and to my children: Michael, Renée and Robert, for their patience and understanding over the past seven years as I have poured myself into my graduate studies. I thank my committee members for their guidance and advice on this research. I thank Dr. Chuck Keating and Dr. Rick McKenzie for serving on my committee and for their support and guidance. I thank Dr. Kathleen Mayfield for her support and encouragement. I thank the M&S Brown Bag members, especially Dr. Saikou Diallo and Dr. José Padilla for their constructive critique of my work. Three persons deserve particular recognition: my dissertation advisor, Dr. Andreas Tolk for his sense of humor and his patient, careful, learned and generous guidance over the past two years; committee member and colleague Dr. Tom Pawlowski for his careful technical review and encouragement; and my wife Nancy Shuman, who provided indispensable support as my editor, consultant and best friend.

## TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION .....	1
1.1 Definitions .....	1
1.2 Definitions for Executable Architecture .....	2
1.3 Importance of Executable Architectures .....	4
1.4 Purpose of the Study (Gap in Body of Knowledge) and Proposal.....	6
2 LITERATURE REVIEW .....	7
2.1 Architecture Elements .....	7
2.2 Modeling Languages .....	9
2.2.1 Structured Analysis.....	9
2.2.2 Object Oriented Languages.....	10
2.2.3 Business Process Modeling Language (BPMN).....	12
2.3 Modeling and Simulation Formalisms .....	14
2.3.1 Coloured Petri Nets.....	14
2.3.2 Discrete Event System Specification (DEVS).....	16
2.4 Themes .....	19
2.4.1 Architecture Description Language (ADL) .....	21
2.4.2 Structured Architecture Development and Executable Architectures .....	21
2.4.3 Object Oriented Architecture Development .....	23
2.4.4 Object Oriented to DEVS .....	26
2.4.5 Executable Extensions to Combat Simulations .....	29
2.4.6 Literature Analysis, Synthesis and Conclusions.....	30
2.4.7 Insight: At the Language Level No Common Concept for Executable Architectures .....	33
2.5 Theoretical Framework .....	36
2.6 Executable Architecture Concept Triangle .....	37
2.7 Transition from Theory to Method.....	43
3 RESEARCH METHODS (QUALITATIVE RESEARCH).....	44
3.1 Type of Design and Underlying Assumptions .....	44
3.2 Grounded Theory Background.....	45
3.2.1 Sampling (theoretical versus purposeful) .....	46
3.2.2 Creativity and Reflexivity (Interaction between the researcher and the world being studied).....	46

3.2.3	Literature Review (beginning or end).....	47
3.2.4	Lack of Precision .....	47
3.2.5	Conclusions with respect to Grounded Theory Criticisms .....	47
3.3	Data Collection, Coding and Analysis, and Theory Development .....	48
3.4	Data Collection and Analysis: Sources & Tools.....	49
3.5	Delimitations and Study Boundaries.....	53
3.6	EACT Process Flow Chart .....	54
3.7	Data Collection and Analysis of Architecture Elements.....	55
3.8	Validity.....	57
4	DATA COLLECTION AND ANALYSIS.....	59
4.1	Data Analysis and Findings High Level .....	59
4.2	Data Analysis and Findings: Detail Level.....	62
4.2.1	Identification of Descriptive Categories (Open Coding) .....	62
4.2.2	Selection of Baseline Architecture Framework .....	64
4.2.3	UPDM Target Set .....	66
4.2.4	Modeling Languages.....	68
4.3	Code Organization.....	69
4.3.1	Population of Individual Data Structures (Ontologies and Compositions) .....	73
4.3.2	Development of Meta-Models through Alignment of Code Database and Visual Views .....	76
4.4	Data Element Analysis Roadmap Execution .....	83
4.4.1	Step 1: Code, Classify & Build UPDM Meta-models .....	83
4.4.2	Step 2: Build Group UPDM Meta-model Maps & Adjust coding .....	85
4.4.3	Step 3: Build Common UPDM Meta-model map & Adjust coding.....	87
4.4.4	Step 4: Code, Classify & Build Language Meta-model Maps.....	89
4.4.5	Step 5: Building Group Meta-model Maps.....	95
4.4.6	Step 6: Build Common Meta-model Map & Adjusting codes.....	102
4.4.7	Step 7: Compare Composite Maps (UPDM & Language) .....	105
4.4.8	Step 8: Coding Model Simulation Formalisms.....	110
4.4.9	Step 9: Compare Simulation Formalism Elementals to Composite Meta-model .....	114
4.4.10	Discrete Event System Specification (DEVS).....	119
4.4.11	Step 10: Define Elementals as Sets Based on Ontologies & Interrogatives.....	126
4.5	EAS Intermediate-level Model.....	132

4.5.1	Static Elements.....	135
4.5.2	Dynamic Elements.....	138
4.6	Meta-model Use Case .....	145
4.7	EAS – Interrogative Meta-model .....	146
4.8	Chapter 4 Conclusions .....	148
5	CONCLUSIONS AND RECOMMENDATIONS .....	149
5.1	Synopsis of Research Results.....	150
5.2	Potential Elements of Executable Architectures (EA) .....	155
5.3	Recommendations .....	158
5.4	Over-specification Concerns .....	158
5.5	Significance of Study .....	159
5.5.1	Practical Implementations and Significance .....	159
5.5.2	DODAF 3.0.....	160
5.5.3	SysML (Next Generation).....	161
5.5.4	Tool Mediation.....	161
5.6	Conclusion.....	161
6	REFERENCES .....	163
	APPENDIXES .....	167
	A. Element Comparison Tables.....	167
	B. Dissertation Electronic Files.....	171
	VITA .....	172

## LIST OF FIGURES

Figure	Page
1 - Taxonomy of UML Structure and Behavior Diagrams .....	10
2 - SysML Diagrams .....	11
3 - Executable Architecture Literature Thematic Map .....	20
4 - Bifurcated Model .....	28
5 - Literature Themes to Gaps to Architecture Framework Map.....	35
6 - Building Theory .....	36
7 - Simplified Executable Architecture Concept Triangle.....	38
8 - Idea for Executable Architecture Specification.....	40
9 - Relationships Explored.....	41
10 - Executable Architecture Concept Triangle.....	42
11 - Transition from Theory to Method .....	43
12 - Data Collection and Analysis .....	49
13 - MAXQDA Data Collection Windows.....	51
14 - MAXQDA MAXMAPS Window .....	53
15 - EACT Process Flow Chart .....	55
16 - Data Collection and Analysis of Architecture Elements.....	56
17 - Data Collection and Analysis – High level .....	59
18 - Data Collection and Analysis .....	62
19 - OV-5b Meta-Model (OMG, 2009a) .....	65
20 - Top Level of Code Organization in MAXQDA .....	70
21 - Code Categories.....	72
22 - Elemental Composition and Generalization Relationships (in MAXQDA Code System) .....	73
23 - Sample OV-2 Composition and Generalization Relationships (in MAXQDA MAXMAPS) .....	76
24 - Data Element Analysis Roadmap: across Similar Meta-model (m-m) Types.....	80
25 - Executable Architecture Triangle (Architecture Elements Guidepost).....	83
26 - Meta-Models.....	84
27 - MAXMAPS with Mouse-Over Memo Display (Needline Definition) .....	85
28 - Building Composite UPDM Group Function Model .....	87
29 - UPDM Composite OV-5 & OV-6a & OV-6b & OV-2 & OV-6c & OV-4.....	88
30 - Executable Architecture Triangle (Modeling Language Guidepost) .....	89
31 - SysML Activity Diagram .....	91
32 - SysML Activity (-Implementation).....	93
33 - Composite Functional Group UPDM-Language.....	97
34 - EAS Meta-model .....	104
35 - EAS and UPDM Comparison.....	107
36 - Executable Architecture Triangle (Modeling Formalisms Guidepost) .....	110
37 - Non-hierarchical CP-net.....	113
38 - CP-net Hierarchical Elements and Similar Composite Elements.....	114
39 - CP-net Relationships .....	116
40 - Composite DEVS .....	123

41 - EAS with Formalism Traces.....	125
42 - Parent-Child Depiction .....	127
43 - EAS Intermediate (EASI).....	132
44 - Meta-model Use Case.....	145
45 - Interrogative Meta-Model.....	147
46 - EACT Summary .....	151
47 - EAS Summary .....	152
48 - Process Element Node Tree Summary .....	153
49 - EAS-I Summary .....	154
50 - EAS - Interrogative Summary .....	155

## LIST OF TABLES

Table	Page
1 - Definitions of Executable Architectures.....	3
2 - Modeling Language and DODAF Alignments.....	13
3 - CP-net Elements .....	16
4 - Classic DEVS Elements .....	17
5 - Parallel DEVS Elements.....	18
6 - Parallel DEVS Processor with a Buffer.....	18
7 - Classic Coupled DEVS Elements.....	19
8 - UML to CP-net Mapping.....	25
9 - Literature Topics, Findings and Gaps (1).....	34
10 - Color and Interrogative Classifications .....	64
11 - UPDM Views .....	67
12 - UPDM Target Set.....	68
13 - UPDM Views and Modeling Language Alignment .....	69
14 - Sample Coding of OV-2 Elementals .....	74
15 - UML Relationships and MAXMAPS Links Equivalences .....	77
16 - Data Element Analysis Roadmap Steps .....	79
17 - Composite Groups .....	86
18 - Group - Language Meta-model Alignment .....	90
19 - SysML Activity Diagram Implementation Detail Elements .....	92
20 - SysML Non-Implementation Detail Element Augmentations (over UML).....	94
21 - UPDM-Language Model Group Composites .....	95
22 - Code Query Sets.....	96
23 - Comparison Classifications .....	98
24 - BPMN Elements.....	100
25 - Functional Group Elemental Comparisons (Events).....	101
26 - Process Group Comparisons (Activities) .....	101
27 - Composite UPDM-Language Member Models.....	103
28 - Comparisons 1 .....	108
29 - Comparisons 2 .....	109
30 - Augmentation Synopsis and Categorization.....	109
31 - CP-net Elements .....	112
32 - CP-net Cross Model Comparison .....	115
33 - CP-net to Composite Relationship Comparisons .....	118
34 - Classic DEVS Elements .....	119
35 - Parallel DEVS Elements.....	120
36 - DEVS Processor with a Buffer.....	121
37 - Classic Coupled DEVS Elements.....	122
38 - Composite DEVS Elements.....	123
39 - DEVS Element Comparisons .....	126
40 - Elements 1 (Executable Architecture Tree).....	129
41 - Elements 2 (Executable Architecture Element Tree) .....	130
42 - The Nine Information and Knowledge Interrogatives.....	131

43 - EAS Intermediate Static Elements .....	134
44 - Intermediate Level Dynamic Elements .....	136
45 - Static Elements of EA.....	156
46 - Dynamic Elements EA .....	156
47 - Activity Comparison Table .....	167
48 - Product Comparison Group.....	168
49 - Rule Comparison Group.....	168
50 - Time Comparison Group.....	169
51 - Control Node Comparison Group.....	169
52 - Node Comparison Group.....	170

# CHAPTER 1

## INTRODUCTION

The objective of this dissertation study has been to conduct a holistic investigation into the elements of executable architectures, in an effort to address a significant gap in the literature, contributing to a theory of executable architectures.

This dissertation has explored a method for developing Executable Architecture Specifications, using the Executable Architecture Concept Triangle (EACT) as a framework for guiding data triangulation. The Executable Architecture Concept Triangle was first described in “Understanding Executable Architectures Through an Examination of Language Model Elements” (Shuman, 2010); it was developed based on observations from the literature that suggest a method for data collection and analysis. The EACT was explored and refined through a qualitative analysis study leading to the *development of a method* for constructing meta-models for executable architecture, and to the *development of meta-models* describing an Executable Architecture Specification. Application of this method in the development of meta-models has enabled a holistic investigation into the potential elements of executable architectures.

### 1.1 Definitions

There are a number of definitions that are presented in this section that are foundational to the concepts presented in this paper.

1. **Architecture:** structure of components, their relationships, and the principles and guidelines governing their design and evolution over time (DOD, 2007a);
2. **Architecture Framework:** guidance and rules for structuring, classifying, and organizing architectures (DOD, 2007a);
3. **Graphical modeling language:** a language for visualizing, specifying, constructing and documenting a system (definition derived from UML definition (Booch, Rumbaugh, & Jacobson, 1999));
4. **Holistic:** looking at the system as a whole -- a unifying approach to methodological development, whereby approaches are linked or integrated into a system; related to System Holism Principle; a System has holistic properties

possessed by none of its parts; each of the system parts has properties not possessed by the system as a whole (Clemson, 1984);

5. **Meta-Model:** a model that defines the components of a conceptual model, process, or system (Booch, et al., 1999); a special kind of model that specifies the abstract syntax of a modeling language (meta-model, 2011);
6. **Necessary:** “adj. That which is needed. a. Indispensable, vital, essential; requisite, citation from the Oxford English Dictionary (necessary, 2011);
7. **Necessary condition:** n. A fact, proposition, etc., on which another thing is dependent or contingent; a prerequisite (necessary, 2011);
8. **Potential:** adj. possible as opposed to actual; having or showing the capacity to develop into something in the future; latent; prospective; **etymology:** post-classical Latin *potentialis* possible as opposed to actual (4th cent.), classical Latin *potential*, *potence* n.+ *-ālis* –al suffix; compare Middle French *potencial*, *potenciel*, Middle French, relating to power or ability (late 15th cent).

## 1.2 Definitions for Executable Architecture

Table 1 provides a snapshot of definitions of the term *executable architecture* from the perspective of previous investigators. Levis drew attention to the need for understanding relationships. Wagenhals emphasized behavioral analysis. Pawlowski described it as a dynamic model of sequenced activities with organization, using resources; in this context he focused on model composability in the context of a combat simulation. Zeigler highlighted the importance of translation of models with sufficient fidelity. Renzhong focused on the development of Colored Petri Nets (CP-NETs) from general systems static UML models. Risco-Martin focused on executable UML models. Mittal described an executable architecture as the use of dynamic simulation software to evaluate architecture models.

All investigators cited in Table 1 described executable architectures as an extension of static architecture modeling into the domain of executable process modeling. Their focus was on what they could solve in the context of specific use cases. This study starts with what they have in common. The perspective or definition used in this study is as follows: **executable architecture supports executable process modeling as a component part of an integrated Architecture Framework (e.g., the US Department of**

*Defense Architecture Framework (DODAF) or UK Ministry of Defence Architecture Framework (MODAF)) that enables behavioral and performance analysis and extends static architecture modeling into the domain of executable process modeling.* This description is derived from Wagenhals, Haider and Levis (2002), Pawlowski (2004), and Mittal (2006).

**Table 1 - Definitions of Executable Architectures**

Author	Year	Title	Executable Architecture Description
Levis (Levis & Wagenhals, 2000)	2000	C4ISR Architectures: I. Developing a Process for C4ISR Architecture Design (Levis & Wagenhals, 2000)	A Dynamic Model, used for understanding relationships and to analyze the properties of the architecture
Wagenhals et al (Wagenhals, et al., 2002)	2002	Synthesizing Executable Models of Object Oriented Architectures (Wagenhals, et al., 2002)	An Executable model based on C4ISR Framework that enables behavioral and performance analysis
Pawlowski, T. (Pawlowski III, et al., 2004)	2004	Executable Architecture Methodology for Analysis, FY04 Final Report (Pawlowski III, et al., 2004)	A dynamic model of sequenced activities with organization, using resources to produce and consume information
Zeigler and Mittal (Zeigler & Mittal, 2005)	2005	Enhancing DODAF with a DEVS-Based System Lifecycle Development Process (Zeigler & Mittal, 2006)	Translation of DODAF compliant architectures into models with sufficient fidelity
Mittal, S., Risco, J. & Zeigler, B., (Mittal, Risco, & Zeigler, 2007)	2007	DEVS-based simulation web services for net-centric T&E” (Mittal, et al., 2007)	Use of dynamic simulation software to evaluate architecture models
Renzhong, W. (Renzhong & Dagli, 2008)	2008	Executable System Architecting Using SysML in Conjunction with CP-net (Renzhong & Dagli, 2008)	Development of CP-net from general systems static models
Risco-Martin (Risco-Martin, De La Cruz, Mittal, & Zeigler, 2009)	2009	EuDEVS: Executable UML with DEVS Theory of Modeling and Simulation (Risco-Martin, et al., 2009)	Executable UML models

This dissertation has examined those architecture elements that have potential to produce executable process models, in the context of an integrated Architecture

Framework. The elements are used across the architecture artifacts. In executable process modeling, processes, change, and causality are evaluated over time. In other words, a static model, having been expressed using some modeling language, is further explored and analyzed through modeling elaborations supported by simulation. From this perspective, the static modeling perspective is expanded to include time, resources, control logic, and behavior, such that there is an elaboration from the two dimensional to the three, with the addition of time, resources, uncertainty and even the possibility of emergent behavior patterns.

### **1.3 Importance of Executable Architectures**

The utility of executable architectures has been addressed at length by Wagenhals and Levis (2000), (2002), Zeigler and Mittal (2005), (2006), and Pawlowski (2004). They cited the importance of executable architectures as a vehicle for providing a more holistic, integrated solution for evaluation of designed architectures. Executable architectures or models can provide a vehicle for evaluation of the logical, behavioral, and performance characteristics of a dynamic system that has been described through static models. Additionally, executable architectures can be used to support test and evaluation of complex architectures, at the system of systems and enterprise system level.

From the perspective of the DOD, Modeling and Simulation is described as one of the key usages of architecture data (DOD, 2007a) to enable evaluation of the logical, behavioral, resource, and performance characteristics of systems; from a cost perspective there is good reason to enable this capability up front rather than it being an afterthought requiring re-work. Tremendous resources are invested in the development of static architectures, which are later reconstructed or rebuilt as executables. DODAF is widely used to build static architectures and models in support of systems analysis and design. However, DODAF has not been explicitly designed with the perspective of extension into the dynamic modeling domain (it will be shown that some simulation elements are present, some are not). Defining the potential elements of executable architectures should enable the development of future architecture frameworks to support a design that could enable dynamic modeling. In addition, in this study, identifying the elements that are useful, and deriving them in general, contributes to theory building by analyzing what

has been done specifically in practice, and then applying analysis methods to understand what is generally theoretically possible.

The DOD Architecture Framework (DODAF) is widely used across the spectrum of capability and systems development in the Department of Defense and is an integral part of the DOD Joint Capabilities Integration and Development System (JCIDS) (CJCSI, 2009) codifies those operational and systems views that should be delivered as part of the definition of systems capabilities and requirements.

Military experimentation (Alberts, 2002) is a critical and complex endeavor that is made possible through model-based systems engineering. This involves system of systems integration between both command and control (C2) and combat simulations. This is similarly the case in training environments. Technical management for engineering prototypical efforts such as Joint Capabilities Technology Demonstrations (JCTD) is realizing the importance of developing Architecture views hand in hand with systems integration in order to facilitate new capabilities exploration and development. These products and views run the full spectrum of models and often have a very data centric focus, thereby facilitating or enabling systems integration.

In order to assess the behavior and performance of complex architectures, static architecture models must be extended into the domain of simulation. For simple process models, the implications for performance and resource utilization can be intuitively determined *a priori*. However, in more complex models where processing is non-deterministic and where resources are not fixed, performance analysis requires the use of simulation techniques to determine measures of performance.

Executable models or simulations serve a number of purposes. One basic function is model logic verification. Is the model logically correct? Model validity is a second purpose which addresses fidelity to the modeled domain and business processes, and may be addressed through model inspection in both a static and dynamic environment. Model process modification and what-if alternative analysis is a third function of executable models. Model process may be altered or refined based on insights gained as a result of dynamic model analysis, which provides an examination of timing. In general, executable models provide measures of performance, but the executable process itself helps in model validation, verification, and experimentation.

#### **1.4 Purpose of the Study (Gap in Body of Knowledge) and Proposal**

This dissertation has been built upon the current body of knowledge surrounding executable architectures. Among the main contributors in this domain in particular Levis, Mittal, Pawlowski, Wagenhals, Zeigler, and Zinn have investigated the transformation of static DODAF architectures into executable architectures. Each researcher investigated some dimension of executable architectures through a particular use case developmental effort; each approached the development of executable architectures in a similar way, starting with a particular static modeling language translated into some particular target dynamic implementation; they all investigated the problem space at an elemental level of translation, from static to dynamic. All researchers provided valuable solution-specific demonstrations of translations from static to dynamic modeling and also showed the value derived from such an endeavor. However, a common theory underlying their applications is still missing. No one has attempted to conduct a holistic investigation into the theoretical elements of executable architectures. This is the gap in the body of knowledge which will be addressed in this dissertation study.

**The proposal of this study was articulated as follows: to conduct a holistic investigation into the possible elements of executable architectures by means of a qualitative investigative study. This study will develop a theoretical framework for inquiry into the dimensions of executable architectures. In the course of this study, the theoretical framework for inquiry will be used to further investigate the elements that have potential for executable architectures.**

The following main contributions have been realized:

- A refined theoretical framework and method for analysis of architecture frameworks in light of the foundational requirement for executable architectures has been developed;
- Through the utilization of the theoretical framework, a description of the theoretical elements and their relationships has been derived.

## **CHAPTER 2**

### **LITERATURE REVIEW**

An overview of Architecture Elements, Modeling Languages, and Modeling and Simulation Formalisms is provided as a foundation for the literature review and to lay the ground work for further discussion of these topics throughout this dissertation. The use of these three main categories has been positively evaluated by peers and has been successfully presented and discussed with experts in the community (Shuman, 2010; Tolk, Garcia, Shuman, 2010):

- Architecture elements focus on static elements and concepts and their attributes;
- Modeling language describe the behavior of such elements;
- Formalisms ensure that the elements and their behavior are captured consistently.

All three categories contribute to the holistic understanding of executable architectures. They will be described in detail in the following sections.

#### **2.1 Architecture Elements**

Architecture Elements are the components of and defined by architectures. Architecture (DOD, 2007a) is defined as the structure of Architecture Elements, their relationships, and the principles and guidelines governing their design and evolution over time. An architecture framework, such as DODAF (DOD, 2007a) “provides the guidance and rules for developing, representing, and understanding architectures.” An architecture framework defines the architecture elements and their relationships to each other in the context of various models or views, and further describes model to model relationships (DOD, 2007a). Architecture frameworks are important because they provide for consistency of model constructs and for interoperability between models from both a syntactic and semantic point of view. Commonality of model syntax and semantics is essential to information sharing. Semantics defines the elemental information sets and their meanings. Syntax defines the relationship of elements to each other. DODAF is used widely across the United States Department of Defense. It was one of the earliest architecture frameworks to be developed and was originally designated the C4ISR Framework. The C4ISR Framework drew heavily from both structured analysis and the Zachman Framework (Zachman, 1999) with its focus on the interrogatives.

DODAF 2.0 (DOD, 2009) is the most recent version of DODAF. Apart from a slightly different model organizational structure and the addition of some very useful views, such as capability views, the main difference between it and DODAF 1.5 is the point of view with respect to data and view. In DODAF 1.5, views drive data. In DODAF 2.0 data drives views.

There are a number of other architecture frameworks, such as the Ministry of Defense Architecture Framework (MODAF), which was developed in the United Kingdom; the NATO Architecture Framework (NAF), which was developed to support NATO; and the Department of National Defence (DND AF), which is the Canadian architecture framework. The TOGAF is a framework for enterprise architecture that was developed and supported by the Open Group which is a global business standards consortium.

Unified Profile for DODAF and MODAF is bilateral: a hybrid of both DODAF and MODAF that is based on a UML modeling language implementation. Unified Profile for DODAF and MODAF (UPDM) (OMG, 2009) was developed by the OMG in partnership with the US Department of Defense (DOD) and the United Kingdom Ministry of Defence (MOD). UPDM specifies a UML 2, and optional SysML, profile to enable practitioners to express DODAF and MODAF model elements and to organize them in a set of views that support the modeling needs of stakeholders. OMG asserts that UPDM will significantly enhance the quality, productivity, and effectiveness of enterprise and system of system models (OMG, 2009a).

In the development of architectures, various approaches are utilized. As DODAF was developed and refined, it was demonstrated that UML implementations of the architecture framework were possible (Bienvenu, Shin, & Levis, 2000). In spite of its roots in structured analysis, DODAF is described as language and implementation neutral. More recently, the OMG has developed specifications for SysML, which is an extension of UML for the systems engineering domain (OMG, 2006). In addition, OMG has developed Business Process Modeling Notation (BPMN) (OMG, 2009) as a modeling language supporting B2B, SOA-based, system of systems modeling. The domain experience of the author has shown that BPMN is increasingly viewed as a means to develop architectures, although in a somewhat limited way. Because it is implemented

by various vendors as an executable process model (e.g., iGrafx), it provides a powerful means for developing executable process models.

## 2.2 Modeling Languages

As stated in Chapter 1, modeling languages provide models with graphical, symbolic, and standard notations designed to address various kinds of inquiry. An architecture framework describes the models or views that are part of that given framework. In the case of DODAF (DOD, 2007a), model language implementation neutrality is asserted as a premise, such that models may be developed using Structured (e.g., IDEF, Data Flow Diagrams, etc.) or Object Oriented language approaches (e.g. UML and SysML). As will be discussed in the literature review, the viability of both Structured and Object Oriented architecture implementations has been demonstrated. A newer modeling language, Business Process Modeling Notation (BPMN), is increasingly used for partial implementation of DODAF views. Key language models of relevance to executable architecture development are IDEF0, UML, SysML and BPMN. It is apparent from the literature review that these languages are the standard languages used to describe executable architectures and they are the primary languages used in practice today.

### 2.2.1 Structured Analysis

Structured Analysis includes a loose collection of modeling and analysis techniques that were developed in the 1960s, 70s, and 80s. Structured Analysis modeling includes the Integrated Definition or IDEF (IDEF, 2010) models, e.g., IDEF 0, IDEF1X, and IDEF 3, the Data Flow Diagram, and the Entity Relationship Diagrams. Volume II of DODAF 2.0 (DOD, 2009) is replete with examples. Of particular interest to process modeling is IDEF 0, which is used extensively in process or behavior modeling. The IDEF 0 models is described in terms of *Input* flows, *Output* flows, *Control* flows and *Mechanism* flows, and the term ICOM was coined as an acronym to describe these flows. The use of IDEF 0 in architecture development is well documented in the literature and in practice; of note the work of Wagenhals is described later under Structured Implementations. IDEF 1X (IDEF, 2010) is a data modeling technique that affords generalization, composition, and association relationships; it is a powerful tool for

describing data entities and their relationships. IDEF 3 (IDEF, 2010), a process model that is less commonly used in practice today, provides a way to model activities, rule constraints, and resource allocations; it is similar to UML Activity Diagrams and BPMN (described in the following sections). Data Flow Diagrams (DfDs) (DeMarco, 1979) are a simple but very powerful modeling technique for describing systems functions and related data flows.

### 2.2.2 Object Oriented Languages

According to the object oriented perspective, the main building block of all software systems is the object or the class (Booch, et al., 1999). Object oriented modeling languages follow this perspective. UML is an Object Oriented language or notation intended for analyzing, describing and documenting all aspects of a software system. It supports modeling various structures using object oriented principles. The current version is UML 2.2. It is comprised of seven Behavior and seven Structure diagrams. The Structure Diagrams are used to depict the static structure of a system, whereas the Behavior diagrams show the dynamic behavior of the objects in a system. Figure 1 shows the UML diagram taxonomy (OMG, 2009). The UML Activity, State Machine and Interaction Diagrams are key diagrams of relevance to process and behavior modeling and for this reason will be discussed extensively in Chapter 4.

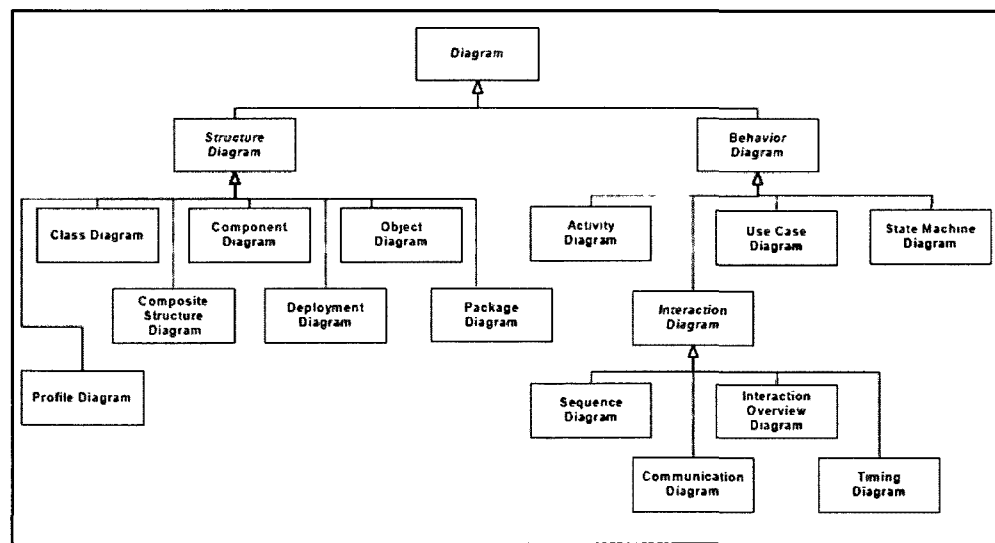
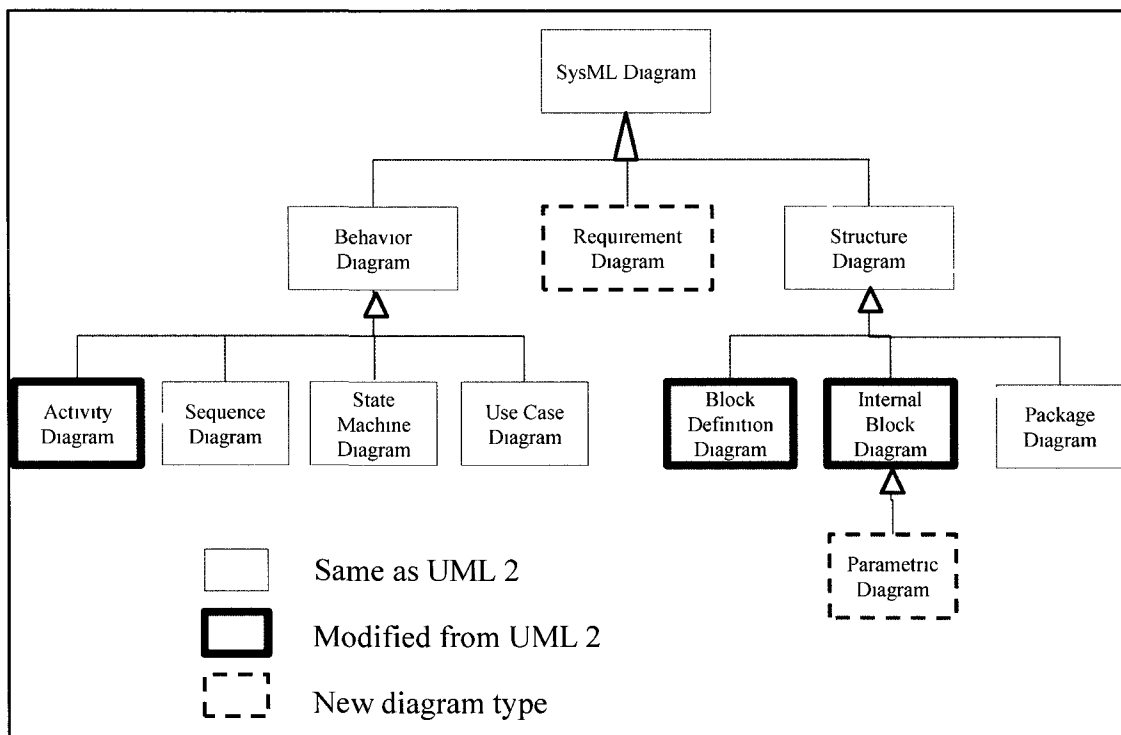


Figure 1 - Taxonomy of UML Structure and Behavior Diagrams

SysML is a UML profile, which is a domain-specific systems engineering modeling language that is used for specifying, analyzing, designing and verifying complex systems, including hardware, software, information flow, people, processes, and resources. SysML reuses seven of UML's thirteen diagrams, augmenting 3 of them, and adds two new diagrams (the Requirements and Parametric diagram) for a total of nine diagram types (OMG, 2006). SysML also supports allocation tables which have a tabular format that can be dynamically derived from SysML allocation relationships. Figure 2 shows SysML diagrams and the legend indicates relationships to UML2.



**Figure 2 - SysML Diagrams**

The significant changes to SysML from UML were described in Shuman (2010) and are provided here as a point of reference. The key diagrams of relevance to executable architectures are the Activity Diagram and the Block Diagram which will be discussed further in Chapter 4. The Activity Diagram is a Behavior Diagram that emphasizes inputs, outputs, sequences, and conditions for coordinating behaviors.

Modifications of the UML Activity Diagram (to the SysML Activity Diagram) includes the addition of data controls and edge extensions (having output parameter sets, probabilities, or parameter value replacement and discarding), all of which were investigated for relevance to executable architectures. Block Diagrams and Internal Block Diagrams provide for blocks or modular units that are used to describe system components and describe their relationships to each other.

Both the Requirements and Parametrics Diagrams add what is needed for Systems Engineering in terms of requirements definition and hard systems performance verification. The SysML Requirements Diagram is new. It supports system requirements engineering and capability taxonomies; however, the focus of this study is soft systems (Flood & Carson, 1993) or process and behavior modeling, which involves the human element. SysML Parametric Diagrams are a new type of diagram which includes constraint blocks for constraining the properties of other blocks; they provide a means to precisely define performance and quantitative constraints such as maximum acceleration, minimum curb weight, and total air conditioning capacity. The ability to define system component attribute constraints is essential to the precise definition of *hard system* (i.e., physical systems) performance but has not been the focus of this investigation.

### **2.2.3 Business Process Modeling Language (BPMN)**

BPMN was developed by the Business Process Management Initiative (BPMI) as a standard for business process modeling. It provides a modeling method that is based on flow charting principles, is similar to UML Activity Diagrams, and is generally described as straightforward and useful for communication of business process descriptions to business and management-oriented stakeholders (OMG, 2009). It is managed by the Object Management Group (OMG), with version 1.1 released in February of 2008. It is comprised of four basic categories of elements: flow objects, connecting objects, swim lanes, and artifacts. Flow objects consist of Events, Activities, and Gateways. There are three connecting objects: Sequence Flow, Message Flow, and Associations. Pools are comprised of Swim Lanes, i.e., participants or entities in a process. Artifacts are comprised of Data Objects required or produced by activities, Groups for documentation, and Annotations providing additional text information (OMG, 2009).

Table 2, developed by Shuman (2010) in “Understanding Executable Architectures Through An Examination of Language Model Elements,” provides a table of comparisons between DODAF models and the four groups of modeling languages previously described: Structured, UML, SysML, and BPMN. Horizontal alignment of models indicates model similarity. The “Fishwick Category” column refers to a taxonomy of model types developed by Fishwick (1995). Some of these similarities have been investigated in this dissertation, as will be discussed in Chapter 4.

**Table 2 - Modeling Language and DODAF Alignments**

Viewpoint	Fishwick Category	DODAF Model Number	Model Type	Structured Modeling Language	UML	SysML	BPMN
<b>Data Viewpoint</b>	<b>Conceptual</b>	<b>DIV-2 (OV-7)</b>	Logical Data Model	IDEF 1X, ERD	Class Diagram	Block Diagram	
<b>Operational Viewpoint</b>	<b>Conceptual</b>	<b>OV-1</b>	Concept Diagram		Use Case	Use Case	
	<b>Functional</b>	<b>OV-2</b>	Operational Node Connectivity Diagram		Communication Diagram	Block Diagram	BPMN
	<b>Functional</b>	<b>OV-3</b>	Information Exchange Matrix			Allocation Tables	
	<b>Conceptual</b>	<b>OV-4</b>	Operational Relationships Diagram		Class Diagram	Block Diagram Package Diagram	
	<b>Functional</b>	<b>OV-5a</b>	Hierarchical Activity Diagram			Block Diagram Package Diagram	
	<b>Functional</b>	<b>OV-5b</b>	Activity Diagram	IDEF 0	Activity Diagram	Activity Diagram	BPMN
	<b>Functional</b>	<b>OV-6c</b>	Operational Event Trace Diagram		Sequence Diagram	Sequence Diagram	BPMN
	<b>Declarative</b>	<b>OV-6b</b>	Operational State Transition Description		State Diagram	State Diagram	
	<b>Functional</b>	<b>OV-6a</b>	Operational Rules Diagram	IDEF 3		Activity Diagram	BPMN
<b>Systems / Services Viewpoints</b>	<b>Functional</b>	<b>SV-1</b>	System to System Node Connectivity Diagram		Communication Diagram	Block Diagram	
	<b>Functional</b>	<b>SV-4</b>	Data Flow Diagram	Data Flow Diagram	Communication Diagram	Activity Diagram	BPMN
	<b>Functional</b>	<b>SV-6</b>	System Data Exchange Matrix			Allocation Tables	
	<b>Functional</b>	<b>SV-7</b>	Systems Measures Matrix	IDEF3		Parameter Diagram	BPMN
	<b>Functional</b>	<b>SV-10c</b>	Systems and Services Event-Trace Description		Sequence Diagram	Sequence Diagram	BPMN
	<b>Declarative</b>	<b>SV-10b</b>	Systems and Services State Transition Description		State Diagram	State Diagram	
	<b>Functional</b>	<b>SV-10a</b>	System Rules Model	IDEF 3		Activity Diagram	BPMN

## 2.3 Modeling and Simulation Formalisms

A modeling formalism for executable architectures should holistically describe the elements and the rules of an executable architecture using a standard mathematical notation. In addition, a modeling formalism should tie the elements together in a consistent and complete way and provide the mathematical framework to demonstrate that all functions are provided and correctly interconnected. Similarly, the elements of an executable architecture should be describable using a modeling formalism, which would in turn provide validating evidence of executable architecture holism (Tolk, Garcia, & Shuman, 2010). Colored Petri Nets (CP-net) and the DEVS formalism are two extensively referenced and used Modeling and Simulation formalisms.

### 2.3.1 *Coloured Petri Nets*

Coloured Petri nets (CP-net) are in wide usage for many practical purposes. As described by Jensen, the main reason for the success of CP-nets is their graphical representation and well-defined semantics, which support formal analysis (Jensen, 1992a). The Coloured Petri net is an offshoot of Place Transition Nets, or “Petri nets.” In his bibliographical remarks, Jensen (1992a) explains the foundation for the Petri net, called the Condition/Event net (CE-net), which was first described by Carl Adam Petri in his doctoral thesis (Petri, 1962). As stated by Jensen (1992a), “A Petri net is state and action oriented at the same time.” States are indicated by ellipses, called places. Each place may contain a dynamically varying number of tokens. The distribution of tokens on the places is called the marking. Actions are indicated by rectangles, which are the transitions. The places and transitions make up the nodes. Directed arrows or arcs are connected between places and transitions. An arc may have an arc expression associated with it.

The Coloured Petri Net (CP-net) is an elaboration on the Petri net, in that it provides for the marking of tokens with associated data values, which are indicated by the token colours. Colour sets determine the possible values of tokens. In Coloured Petri-nets, arc expressions, which evaluate to multi-sets, specify the collection of tokens, each with a well-defined token colour. In CP-nets, token marking of a given place is indicated by a small circle with an integer for the number of tokens, and a text string that specifies a multi-set which describes the token colors in terms of their coefficients

(Jensen, 1992). Jensen (1992a) attributes the wide use and success of Petri nets to having “a graphical representation and a well-defined semantics, allowing formal analysis.” He lists twelve advantages to using CP-nets:

1. CP-nets have a graphical representation.
2. CP-nets have a well-defined semantics which unambiguously defines the behaviour of each CP-net.
3. CP-nets are very general and can be used to describe a large variety of different systems.
4. CP-nets have very few, but powerful, primitives.
5. CP-nets have an explicit description of both states and actions.
6. CP-nets have a semantics which builds upon true concurrency, instead of interleaving.
7. CP-nets offer hierarchical descriptions.
8. CP-nets integrate the description of control and synchronization with the description of data manipulation.
9. CP-nets are stable towards minor changes of the modeled system.
10. CP-nets offer interactive simulations where the results are presented directly on the CP-net diagram.
11. CP-nets have a large number of formal analysis methods by which properties of CP-nets can be proved.
12. CP-nets have computer tools supporting their drawing, simulation and formal analysis (Jensen, 1992).

Table 3 provides CP-net elements, formal definitions and simple verbal descriptions. CP-net elements will be further described and used as a validating source in Chapter 4.

Table 3 - CP-net Elements

Code	Formal Definition	Interpretation
<b>Transitory Objects</b>		Ephemeral objects (messages and data)
Token colour		Attributes associate with Tokens
Tokens		Dynamically varying black dots associated with a place
Global Declaration node		Defines all colour sets
<b>CP-net Control Elements</b>		Control functions and definitions
Colour Sets ( $\Sigma$ )	$\Sigma$ finite set of non-empty types	Each token on a place p must have a token colour that belongs to type C(p)
Initialization function (I)	Defined from P into closed expressions such that $\forall p \in P [Type(I(p)) = C(p)_{ms}]$	Initial marking
Arc expression (E)	$\forall a \in A [Type(E(a))C(p(a))_{ms} \wedge Type(Var(E(a))) \subseteq \Sigma]$ where p(a) is the place of N(a)	Maps each arc, a, to an expression of type C(p(a))
Guard function (G)	It is defined from T into expressions such that $\forall t \in T [Type(G(t)) = B \wedge [Type(Var(G(t))) \subseteq \Sigma]$	Additional constraint (Boolean) enabling transition
Node function (N)	Defined from A into $P \times T \cup T \times P$	(v) The node function maps source and destination nodes
Color function ©	Defined from P into $\Sigma$	C maps each place, p, to a colour set C(p)
<b>Fixed Objects</b>		Fixed objects (nodes and links)
Places (P)	P is a finite set of places	State of a resource allocation, or of process (circle)
Port Place		Connections for communication between Objects
Arcs (A)	A is a finite set of arcs such that $P \cap T = P \cap A = T \cap A = \emptyset$	Connects a place with a transition or a transition with a place
Hierarchical structure		Hierarchical structure is developed for the CP-net
Transitions (T)	T is a finite set of transitions	Actions of resource allocation system (rectangle)

### 2.3.2 Discrete Event System Specification (DEVS)

The Discrete Event System Specification (DEVS) (Zeigler, Praehofer, & Kim, 2000) is a formalism that provides a means for describing the components of discrete-event simulation. In Classic DEVS, basic (atomic) models and their elements are described; these elements include input and output ports for receiving and sending information (messages), a set of state variables, internal and external transition functions and a time advance function (Mittal, Zeigler, Risco Martín, Sahin, & Jamshidi, 2008). Classic DEVS is mathematically represented as a tuple of seven elements

$M = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta)$ .  $X$  is an input set,  $S$  is set of states,  $Y$  is set of outputs,  $\delta_{int}$  is internal transition function,  $\delta_{ext}$  is external transition function,  $\lambda$  is the output function, and  $ta$  is the time advance function. (Zeigler, et al., 2000). Table 4 provides a list of the Classic DEVS elements with definitions.

**Table 4 - Classic DEVS Elements**

Code	Definition
$e$	time elapsed since last transition
$ta$	$S \rightarrow R_{0,\infty}^+$ is the set positive reals with 0 and $\infty$
$Q$	$Q = \{(s, e) \mid s \in S, 0 < e < ta(s)\}$ is the total state set
$S$	Set of states
$X$	Set of input values
$Y$	Set of output values
$\delta_{ext}$	$Q \times X \rightarrow S$ is the external transition function
$\delta_{int}$	$S \rightarrow S$ is the internal transition function
$\lambda$	$S \rightarrow Y$ is the output function

The DEVS formalism now includes Classic DEVS, Parallel DEVS and Classic Coupled DEVS, having been enlarged over time from Classic DEVS. Parallel DEVS was introduced by Zeigler fifteen years after the Classic DEVS formalism. It removes constraints that originated with the sequential operation of early computers and hindered the exploitation of parallelism. Parallel DEVS differs from classic DEVS in allowing all imminent components to be activated and to send their output to other components. The receiver is responsible for examining this input and properly interpreting it. Messages, basically lists of port-value pairs, are the basic exchange medium. According to Zeigler (2000), a basic Parallel DEVS is a structure,  $DEVS = (x_m, y_m, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$ . Table 5 lists Parallel DEVS elements and their definitions. In comparison to Classic DEVS, in Parallel DEVS, there is the addition of ports and the confluent transition function for resolution of collisions between external and internal events.

**Table 5 - Parallel DEVS Elements**

Code	Definition
(ta) time advance function	$S \rightarrow R_{0,\infty}^+$ is the set positive reals with 0 and $\infty$
(Q) set of total states	$Q = \{(s,e) \mid s \in S, 0 < e < ta(s)\}$ is the total state set
(S) set of sequential states	set of states
( $X_m$ ) set of input ports and values	set of input values and ports
( $Y_m$ ) set of output ports and values	set of output values and ports
( $\delta_{con}$ ) confluent transition function	decides next state if collision between external and internal event
( $\delta_{ext}$ ) external state transition	$Q \times X \rightarrow S$ is the external transition function
( $\delta_{int}$ ) internal state transition	$S \rightarrow S$ is the internal transition function
( $\lambda$ ) output function	$S \rightarrow Y$ is the output function

Parallel DEVS with a buffer is an elaboration on the Parallel DEVS with the explicit inclusion of a buffer, V, which functions as a queue for holding an arbitrary input set. “A processor that has a buffer is defined in Parallel DEVS as:  $DEVS_{processing\_time} = (X_m, Y_m, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$ ” (Zeigler, et al., 2000). Table 6 shows the elements of Parallel DEVS with a Buffer and their definitions.

**Table 6 - Parallel DEVS Processor with a Buffer**

Code	Definition
( $X_m$ ) set of input ports and values	set of input values and ports
( $Y_m$ ) set of output ports and values	set of output values and ports
(V) Queue	V is a queue that holds an arbitrary set or a bag
(ta) time advance function	$S \rightarrow R_{0,\infty}^+$ is the set positive reals with 0 and $\infty$
(S) set of states	Set of states
( $\lambda$ ) output function	$S \rightarrow Y$ is the output function
( $\delta_{int}$ ) internal state transition	$S \rightarrow S$ is the internal transition function
( $\delta_{ext}$ ) external state transition	$Q \times X \rightarrow S$ is the external transition function
( $\delta_{con}$ ) confluent transition function	Decides next state if collision between external and internal even

In classic Coupled DEVS, the DEVS formalism includes elements for building models from components. Under this construct, atomic models may be coupled together to form coupled models. The specification includes the external interfaces, input and output ports and values, the components (which are DEVS models), and the coupling relations:  $N = \{X, Y, D, \{M_d \mid d \in D\}, \text{EIC}, \text{EOC}, \text{IC}, \text{Select}\}$  (Zeigler, et al., 2000). Table 7 shows the elements that make up Classic Coupled DEVS.

**Table 7 - Classic Coupled DEVS Elements**

Code	Definition
(D) component names	Set of the component names
(IC) internal coupling	Connects component outputs to component inputs
(EOC) external output coupling	Connects component outputs to external outputs
(EIC) external input coupling	Connects external inputs to component inputs
$(X_d)$ set of input ports and values	set of input values and ports
$(Y_d)$ set of output ports and values	set of output values and ports
(Y) output ports and values	Set of output ports and values $Y = \{(p, v) \mid p \in \text{OPorts}, v \in Y_p\}$
(X) input ports and values	Set of input ports and values $X = \{(p, v) \mid p \in \text{IPorts}, v \in X_p\}$
$(M_d)$ DEVS Model	$M_d = (X_d, Y_d, S, \delta_{ext}, \delta_{int}), \lambda, ta)$ is a DEVS
$X_d$	$X_d = \{(p, v) \mid p \in \text{IPorts}_d, v \in X_p\}$
$Y_d$	$Y_d = \{(p, v) \mid p \in \text{OPorts}_d, v \in Y_p\}$
Select	Tie-breaking function (used in Classic DEVS)

This introduction to the four DEVS model types provides a foundation for the remainder of the literature review, for method discussions in Chapter 3, and for data collection and analysis in Chapter 4.

## 2.4 Themes

Figure 3 is a thematic Map that shows the major research areas related to executable architecture, divided into categories. The blue boxes show the topic area with the principal researcher and date. The orange boxes show the focus of the research, and the rose boxes show identified Gaps. The cloud overlay is suggestive of areas that this study has addressed to some degree. These research areas, with their key topic areas and related gaps, will be discussed in this section.

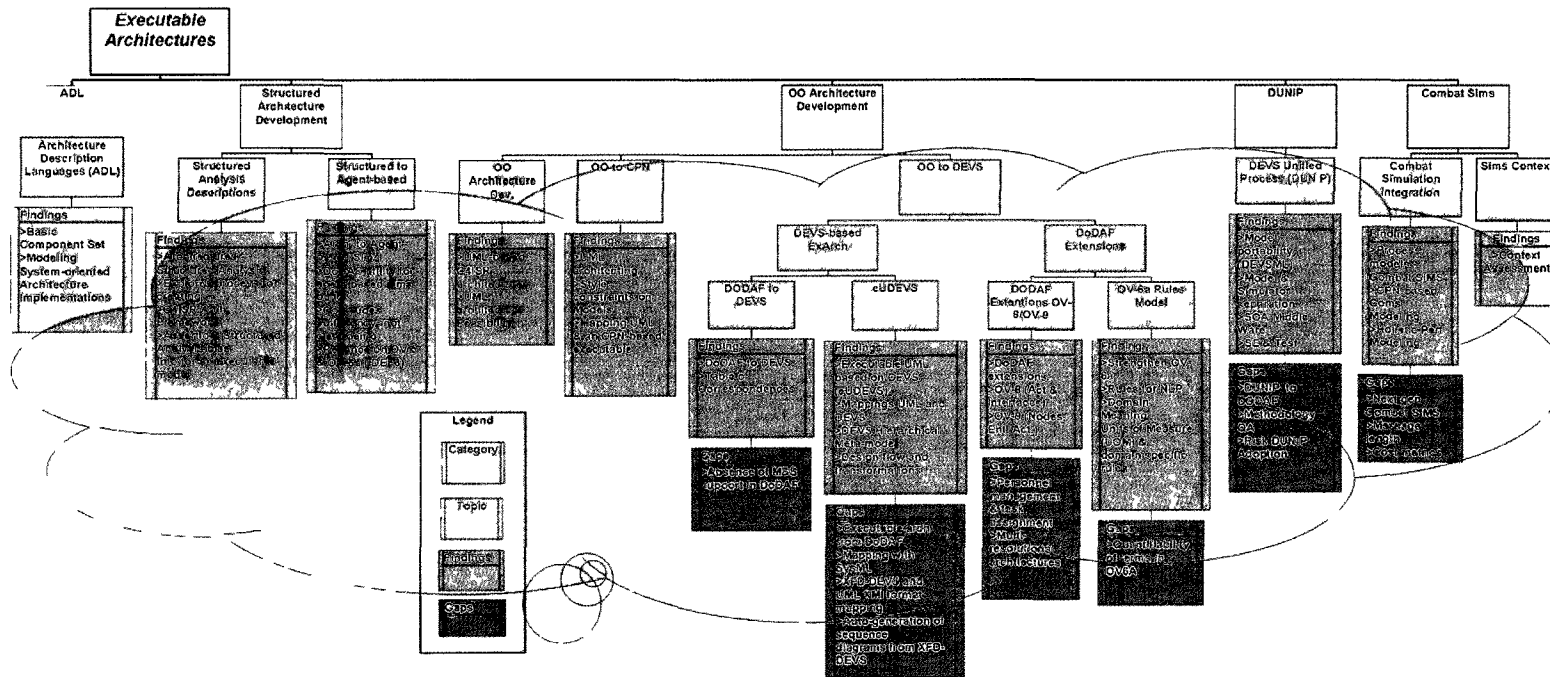


Figure 3 - Executable Architecture Literature Thematic Map

#### **2.4.1 *Architecture Description Language (ADL)***

Petty, McKenzie, and Qingwen (2002) simulated the data flows in a federation using Rapide and ACME, which are proprietary tools that were introduced in the paper. Using Acme, they estimated the number of entities that a federate could support. Both Rapide and Acme are proprietary examples of Architecture Description Languages (ADL). An ADL is a language that represents software designs at the architecture level, in terms of components and interactions (some ADLs support simulation). They cite the assertion (Shaw & Garlan, 1996) that six types of ADL language elemental types form a sufficient vocabulary for expressing any software architecture: Component (performs computation and retains state), Connector (represents relations or interactions between components), Port (a component interaction point), Role (the interaction point of a connector), Representation (a composite object – component or connector), and Binding (mapping between composed object interfaces and external interfaces). The ADL topic category is primarily focused on systems oriented architecture implementations, but it is relevant to this work because the elemental types are similar to the elemental categories described and used in this dissertation, to be discussed in Chapter 4.

#### **2.4.2 *Structured Architecture Development and Executable Architectures***

This section covers two key structured analysis-oriented approaches to the development of executable architectures depicted in Figure 3 as a topic: Structured Analysis to Coloured Petri Nets and Structured Analysis to Agent Simulation.

##### **2.4.2.1 *Structured Analysis & Coloured Petri Nets:***

In the Wagenhals and Levis (2000) paper, “C4ISR architectures. I: Developing a process for C4ISR architecture design,” the authors explored a process for creating the essential and supporting products of the DOD C4ISR Architecture Framework (version 2.0) and asserted that using Structured Analysis it is possible to develop a process that generates the necessary information for derivation of an executable model.

In a related paper, “C4ISR architectures: II. A structured analysis approach for architecture design,” Wagenhals, Shin, Kim, and Levis (2000) provide a detailed explanation of the development of a coherent set of architecture descriptions conforming to the C4ISR Architecture Framework based on the Structured Analysis modeling methods. In the words of the authors, they describe the “necessary and sufficient” sets of

information for creating executable models from the architectures, using a Coloured Petri Net simulation construct. In this study, the executable model was developed using the Activity Model (developed in IDEF0), the Data Model (developed in IDEF1X), the Rule Model and the State Transition Diagram. They describe elemental associations between these four models and a Coloured Petri Net executable implementation. Associations were described as follows: IDEF 0 Activities to CP-net Transitions, IDEF 0 arrows to CP-net Arc-Place-Arc combination, and IDEF 0 arrow to CP-net Color Sets associated with the CP-net Place. IDEF 1X entities are used to derive the names of color sets in the CP-net Global Declaration Node, and each Color Set that is assigned to a place has the same number and type of attributes as shown in the IDEF1X data model. Rules in the Rule Model were used to specify the Arc Inscriptions and Guard Functions. The State Transition Diagram was created by tracing a thread through the IDEF0 model, and the State Transition Diagram is used to verify that the model executes correctly.

#### **2.4.2.2 Structured Analysis to Agent Simulation:**

In his thesis, “The Use of Integrated Architectures to Support Agent Based Simulation An Initial Investigation,” Zinn (2004) investigated the utility of using DODAF architecture products for providing needed data for agent based simulations. Zinn proposed a process of taking information from DODAF architectures and importing it into an agent-based simulation. This was accomplished by means of a case study where architecture data from a proposed Air Operations Center architecture was used in the combat model System Effectiveness Analysis Simulation (SEAS). In his research, he relied heavily on the DODAF Activity Diagram (OV-5) and the Rule diagram (OV-6a), which was developed using IDEF3 (IDEF, 2010). It may be observed that IDEF3 is a very robust modeling language in comparison to the simple DODAF meta-model for a Rules Diagram (OV-6a) (addressed in Chapter 4). In the context of his case study, Zinn made a general assertion that DODAF is sufficient for developing executable architectures, but because there is no clear, elemental traceability in his thesis, the validity of this assertion is more anecdotal than specific.

### **2.4.3 Object Oriented Architecture Development**

Object oriented implementations of both static and executable architecture implementations are discussed in this section.

#### **2.4.3.1 Object-Oriented Architecture Development**

In their study, Bienvenu, Shin, and Levis (2000) investigated object-oriented approaches to developing C4ISR architecture. They provided a UML-based process using object-oriented methods for developing C4ISR architectures, and they provided a table of correspondences between C4ISR views and UML products. This work was foundational in the object-oriented language implementation of DODAF architectures.

#### **2.4.3.2 Object Oriented to Coloured Petri Nets (CP-net)**

This study by Wagenhals, et al. (2002) provides a description of an architecting process based on the object-oriented Unified Modeling Language (UML). It is one of the seminal papers in the area of executable architectures. They describe a mapping between the UML static implementations and an executable model based on Colored Petri Nets (CP-net), and they examine DODAF product sufficiency in terms of the CP-net simulation end state objective. Their model focus was on the UML Sequence Diagram (OV6c), the UML Collaboration Diagram, and the Class Diagram.

Using the Unified Modeling Language (UML) to describe the architecture, the authors provided keen insight into the development of simulations from static, UML-specified DODAF architectures and also showed the correspondence of UML elements to the elements of a Coloured Petri Net (CP-net)-based simulation. The primary justification for the development of executable architectures is validation and verification of static models. The authors provided a step by step methodology for building CP-net from UML, utilizing both structure and behavior UML diagrams. They used the Class Diagram, a structure diagram type, as well as the Activity Diagram, the Sequence Diagram, and the Collaboration Diagram: all behavior diagrams, emphasizing the importance of concordance between diagrams. In their approach the sequence and activity diagrams are used to facilitate the development of the class diagram, hence the importance of diagrammatic concordance. Their method imposes two class implementation style constraints:

- The first constraint requires the partitioning of classes into those that represent fixed structures (represented by non-association classes) and those that represent transient structures (represented by association classes).
- The second style constraint requires that all non-association classes which represent the fixed elements of the architecture be converted into classes that contain either operations or attributes but not both.

As Wagenhals, Haider, and Levis point out (2002), the partitioning of classes into association and non-association is based on the interoperability emphasis in DODAF, in which transient structures (i.e., messages) are passed between fixed structures (i.e., nodes and links). Having these two categories of objects facilitates a mapping between UML and CP-net. Accordingly, non-association classes contain the operations and perform actions that cause a change of state to a token or message, and it is the non-association classes with their operations that form the basis for the CP-net transitions. Non Association classes are structured into parent and aggregation classes. The Class Diagram structure becomes the basis for the hierarchical CP-net structure. Association classes have only attributes, which become the basis for the global declaration node and the message tokens. This stylistic approach supports an unambiguous mapping from UML to a CP-net. Table 8 provides a useful summary of UML to CP-net mappings described by Wagenhals et al. (2002).

**Table 8 - UML to CP-net Mapping**

Step	UML	CP-net
1	Attributes of all classes	Global Declaration Node, Color sets
	Class structure	Hierarchical Structure
2	Each non-association class (parent classes with only operations)	Transition
3	Association Class or Aggregated Class	Place (referred to as “port places”) with Color sets defined from attributes)
4		Arcs (placed between transitions & places)
5	Activity Diagram	Place
6	Associations in Class Diagram	Place (one to one)
7		Sub-page (for each substitution transition)
8		Inputs, Outputs, I/O port places
9	Based on Activity Diagram	Arcs
10	Rules (each operation)	Arc Inscriptions, guard functions, or code segments

Consistent with their initial premise concerning the importance of executable architecture, Wagenhals et al. devote considerable attention to the evaluation of architectures. The authors divide this topic into logical and behavioral evaluation:

- Logical evaluation is based on proper running of the CP-net simulation, e.g., does it run without deadlocks and infinite cycles?
- Behavioral evaluation of architecture focuses on correct sequencing and on stimulus driven behavior. Stimulus based evaluation would assess the model in steps using code stops to evaluate discrete sequences.

In their conclusion, Wagenhals et al. highlight the CP-net-based method as a means for development and subsequent validation of architectures. They further suggest the applicability of the method to future UML-oriented architecture tool implementations.

The discussion of the development of foundational use cases is weak but was not the focus of their study. A table of correspondences between the UML elements and the CP-net would have been useful. The authors did not address resourcing and the effects on the CP-net model. Presumably this would add additional parallel transitions to the CP-net accounting for multiple processing capabilities. Certainly any analysis of system measures of performance (MOPs) would need to account for resourcing.

#### **2.4.4 Object Oriented to DEVS**

This section covers Object Oriented to DEVS implementations. It includes DEVS implementations, DODAF extensions supporting DEVS implementations, and DEVS Unified Process (DUNIP).

##### **2.4.4.1 DEVS-based Executable Architectures**

In their paper entitled, “Enhancing DODAF with a DEVS-based System Lifecycle Development Process,” Zeigler and Mittal (2005) suggested a method for transforming DODAF descriptions of an architecture to a DEVS representation. In this paper the authors provided some justification for the endeavor and also provided an introduction to the “Bifurcated DEVS-to-DODAF Development Process.” In general, the paper is written at a high level of abstraction and is lacking in specifics, but it does provide a table of correspondences between DODAF models (Views) and related DEVS simulation components. This is one of the more useful elements of the paper and has direct relevance to the dissertation objectives. This paper led to Mittal’s dissertation.

Risco-Martin, De La Cruz, Mittal, and Zeigler (2009) in their paper entitled, “eUDEVS: Executable UML with DEVS Theory of Modeling and Simulation,” described the essential mappings between UML and DEVS modeling. Their work focuses on the UML Structure and Behavior models that contribute to the development of a DEVS-based system model. The UML Structure models are the Component, Package, and Class Diagrams. The UML Behavior models are the State Machine, the Sequence Diagram, the Timing Diagram, and Use Case. In this paper the authors propose a design flow and set of transformations to generate a Discrete Event Specification (DEVS) executable simulation model from a UML graphic specification. The authors describe the UML state machine deficiency with respect to the DEVS state machine, in that UML contains no provision for timeouts for each state, which is known as time advance in DEVS. This problem is cited by Mittal (Mittal, 2006) in his paper “Extending DODAF to Allow Integrated DEVS-Based Modeling and Simulation.” In this paper he coined the term eUDEVS which stands for executable UML based on DEVS. His work builds upon the elemental mapping described by Mittal (Mittal, 2006) by providing a detailed implementation. The authors describe a 3 step method:

1. Synthesis of a static structure defined using a UML model;
2. Specification of behavior using an XML-based finite deterministic DEVS state machine;
3. Auto-generation of Platform Specific Models (PSM) from the Platform Independent Models (PIMs), later described under DUNIP.

Additionally, the authors provide a DEVS hierarchical meta-model that is useful in understanding the elemental components that make up DEVS, from a taxonomy point of view. In Chapter 4, a similar approach to DEVS elemental description is taken in the exploration of the relationships between DEVS and the Executable Architecture Specification.

#### **2.4.4.2 DODAF Extensions**

Mittal (2006), in his journal article entitled, “Extending DODAF to allow Integrated DEVS-Based Modeling and Simulation,” addressed the question of extending DODAF to support integrated DEVS-based modeling. His work cited DODAF’s shortcomings, including ill-defined information exchanges, the need for a linking of entities, activities, and nodes, and a need to identify ports associated with activity-to-activity communication (since DEVS is a port-based modeling construct). He defined two new OV products, the OV-8 and the OV-9, as extensions of the DODAF: the OV-8 addresses activities and their logical interface information and the OV-9 maps nodes, entities, and activities. This is similar conceptually to Activities-based methodology (Ring, Nicholson, & S, 2008). Mittal asserted the need for the OV-8 and OV-9 as intermediate precursor products in the development of the DEVS simulation. Mittal used the OV-5 activity model, the OV-6c (Sequence Diagram), and the OV-6a (Rules diagram – IDEF3), as a basis for generating a DEVS-based simulation.

In a second, related paper by Mittal, Mitra, Gupta, and Zeigler (2006) entitled “Strengthening OV-6a Semantics with Rule-Based Meta-models in DEVS/DODAF based Life-cycle Architectures Development,” the authors described a means for semantically strengthening the critical OV-6a Rules Model through application of Units of Measure (UOM), Domain Meaning, and formatting to domain specific rules, thereby removing ambiguity and aiding in translation of static to dynamic architectures.

#### 2.4.4.3 DEVS Unified Process (DUNIP)

The DEVS Unified Process (DUNIP) (Mittal, 2007) is based on the Bifurcated Model Continuity-based Life Cycle Process (Zeigler & Mittal, 2005), referred to hereafter as the Bifurcated Model. In order to understand DUNIP, one must first understand the Bifurcated Model, which is a process model that describes a simulation supported method for developing and testing systems of systems and enterprise level systems (Mittal, et al., 2008). The graph shown in Figure 4 depicts the steps that are described below:

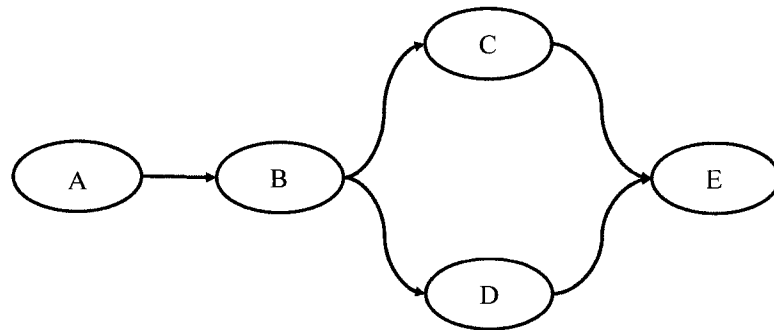


Figure 4 - Bifurcated Model

- A. **Develop behavior and systems requirements specifications:** DODAF descriptions of the operational, systems and technical views are created to describe the system under test. These views are static DODAF models that are mapped to a system simulation implementation (e.g., DEVS).
- B. **Model Structures at higher levels of system specification:** A system simulation is developed using platform independent model (PIM) concepts from Model Driven Architecture (MDA) (OMG, 2003), in which the simulation model is separate from the simulator. The model describes a branching from step (B) to step (C) and step (B) to step (D), hence the term bifurcation.
- C. **Reference Master Model (Simulation Execution):** This is a master simulation model for any implementation of behavior requirements, and it can be run and analyzed to study logical and performance attributes (step C connects to step E).

D. **Semi-automated test suite design:** This is a test suite that provides models or simulation interactions or stimulation behaviors for interaction with the live system under test (step D connects to step E).

E. **Verification and Validation (V&V):** Both steps C and D come together to support V&V, which leads to system optimization and fielding.

Mittal (Mittal, 2007) elaborated on the Bifurcated Model in the development of the DEVS Unified process (DUNIP). The DUNIP process is comprised of the following four components:

1. Automated DEVS model generation from requirement specification formats (e.g., DODAF);
2. Collaborative model development using DEVSMML, which is a platform independent, XML based specification language;
3. Automated generation of the test suite (from the Bifurcated Model);
4. Net-centric execution of the model and test suite over a Service Oriented Architecture (SOA) (W3C, 2004).

#### 2.4.5 *Executable Extensions to Combat Simulations*

A mixed approach utilizing elements of several methods described above was applied by Pawlowski and Ring (2004) in their MITRE Technical Report entitled “Executable Architecture Methodology for Analysis, FY04 Final Report.” They described their method for converting static DODAF-based architecture products into an executable architecture that supports the dynamic analysis of a system in terms of performance and effectiveness and resource utilization. They created a three-fold modeling construct in which executable architectures or process models serve as an extension of combat simulation models. This coupling was further augmented with communications timing data supplied by a supporting communications model. Their approach leveraged the translation of static process models into a dynamic Bonaparte Colored Petri Net executable. This executable process model, in concert with a communications modeling tool and a combat simulation, were combined into an HLA based federation. Essentially the object of the experimentation was to use executable architectures as a vehicle for detailed process study and investigation, in the larger

context of a combat simulation. A significant part of this research focused on integration and alignment of models through the notion of operational model complementarity.

Garcia (2011) extended this work by developing a method for assessing a system's executable architecture in a larger operational or system of systems context (addressing the why and how information interrogatives). His research describes a means to assess the contribution and efficiency of the system before it is built. This research led to the development of a method for synthesizing observations about executable architectures, based on (1) the assessment recommendations provided by the North Atlantic Treaty Organization (NATO) Code of Best Practice for Command and Control (C2) Assessment (CCRP, 2002) and (2) metrics for operational efficiency from the Military Missions and Means Framework (Sheehan, Deitz, Bray, Harris, & Wong, 2003). These two approaches show that the methods can be successfully mixed delivering more functionality as needed for executable architectures. However, both are based on contributions to the extended applicability of executable architectures. As such, they show that all three categories are useful and should be taken into consideration when evaluating executable architectures in support of a common theory.

#### ***2.4.6 Literature Analysis, Synthesis and Conclusions***

Table 9 provides a synopsis of the main literature review topics, findings, and identified research gaps. The research spans a period of about ten years. Table 9 shows the research categorized into five areas as follows:

- Architecture Description Languages,
- Structured Modeling and Transformations,
- Object Oriented (OO) Transformations,
- DUNIP,
- Executable Extensions to Combat Simulations.

Within each category, the primary research topics are shown with the principle author, year, key findings, and research gaps that surface from the research.

In the literature review, it is apparent that Petty, Bienvenu, Garcia, Mittal, Pawlowski, Wagenhals, Zeigler, Zinn, and their respective co-authors have investigated various aspects of the transformation of static DODAF architectures into executable

architectures. Each research effort proposed specific methods and approaches for making these transformations. Petty and McKenzie used proprietary Architecture Description Languages to describe simulation federation communications. Wagenhals and Zinn initially focused on Structured implementations of DODAF and their transformations to executable models. Bienvenu demonstrated the development of architecture models developed in UML and Wagenhals led a team that demonstrated a method for their translation into CP-net. Zeigler, Mittal, and Risco-Martin explored the transformation of UML developed Architectures into DEVS-based executable implementations, and Mittal described augmentations to address some of the deficiencies in the DODAF meta-model, suggesting the addition of two new products to address issues associated with modularity, to align DODAF to the DEVS construct. Mittal developed DUNIP, which was based on the Bifurcated Model Continuity-based Life Cycle Process, which was described earlier by Zeigler and Mittal. The focus of DUNIP was on platform XML-based independent models (which is similar to the platform independent models in Model Driven Architectures (MDA)) and SOA model interoperability. This was a leap forward that focused on model portability and SOA communications. Additionally, it is suggested in the DUNIP literature that the method has been extended to other modeling languages, such as BPMN. The last major category is executable extensions to combat simulations, in which process models are run in conjunction with combat simulations and communications models. This approach calls to mind the Bifurcated Model Continuity-based Life Cycle Model, with its notional capacity to support system subject-of-test, in the context of a test suite. The contextual analysis by Garcia extends this work with its focus on system of system executable architecture integration.

Each of these research efforts starts with some form of static DODAF or DODAF-like model and enlarges the modeling perspective into simulations. Whether through a structured language to a CP-net-based executable or through an object-oriented (UML) language to DEVS, transformation to executable simulations is a common theme. Use of DODAF views was the starting point, and most transformation approaches were manual with the exception of Mittal and Risco-Martin, who proposed a semi-automated implementation through the use of DUNIP. All addressed reasons for the development of executable architectures, with process investigation and model V&V as the key drivers

for all. Similarly, each approached the translation of static architecture views from an elemental level perspective, where DODAF views were described in terms of their constituent elemental components, which were subsequently translated into executable models. In the case of Wagenhals the elemental transformations from OO to CP-net were unambiguous. For the others, there was a spectrum of transparency in their transformation explanations.

The gaps that were identified from the literature review are shown in Table 9 next to associated research topics and topic category. The far right column in Table 9 indicates with a check mark that there is a relationship between one or more of the gaps in the adjacent cell. After the gaps were identified, they were thematically mapped to the Executable Architecture Concept Triangle components: Architecture Elements, Modeling Languages, M&S Formalisms, and Executable Architecture Specifications. This was facilitated using a concept mapping tool. A concept mapping tool is useful for visually identifying thematic relationships, and MindManager 8 (MindManager, 2011) is the tool that was chosen for this task. Figure 5 shows the mapping of gap themes to the components of the Executable Architecture Concept Triangle: blue lines map to the Architecture Elements, the green lines map to Modeling Languages, purple lines map to Modeling and Simulation Formalisms, and red lines map to Executable Architecture Specifications. The legend in Figure 5 identifies the meaning of the shapes: Categories, Topics, Gaps (related), Gaps (not related), and Themes. Themes are interpretations of the meaning of the gaps, and are shown in Figure 5 to the right of the gap. Based on the assessment of the themes conveyed by the gaps, it becomes obvious which gaps are related to the central concepts of the dissertation, and which are not. The shape representing Gaps (not related) is present for those gaps not directly related to the dissertation topic. Again, relationships between the gap themes and the components of Executable Architecture Concept Triangles are shown with the relationships lines. Many of the gaps have more than one theme, which can be shown to relate to more than one concept in the triangle; for example, mapping other Languages (e.g., BPMN) to CP-net suggests Modeling Language and M&S Formalism themes. This method allows for the synthesis of gap themes into a coherent conceptual framework.

To reiterate, the definition of executable architectures was addressed in Chapter 1, and for the purposes of this dissertation, executable architecture refers to executable models or simulations that are based on static models developed in the context of some Architecture Framework (e.g., DODAF or MODAF). These simulations enable both behavioral and performance analysis. They extend static architecture modeling into the domain of executable process modeling.

#### **2.4.7 *Insight: At the Language Level No Common Concept for Executable Architectures***

As described in the literature review, various approaches to the topic of executable architectures have been investigated. Levis and Wagenhals were pioneers in architecture based development of Coloured Petri-Net-based simulation implementations. They explored both structured (IDEF) and UML architecture implementations (Wagenhals, et al., 2002). Mittal explored DODAF from the perspective of DEVS simulation implementations. Mittal's work was based on a UML architecture modeling language implementation, and suggested various extensions to DODAF to accommodate DEVS simulations implementations. Each approach contributed to our overall understanding of the relationships between architecture frameworks and simulation.

In conclusion, executable architectures are both useful and used. However, it is clear from a language implementation perspective that there is no common concept for developing executable architectures. Rather, there are a variety of modeling language implementation approaches that are possible, and similarly, from a simulation end-state perspective, there are a number of possible approaches to simulation definition, to include CP-net and DEVS implementations.

From examination of the literature, it becomes apparent that previous research has produced much valuable information from a specifically focused, deconstructionist perspective; that is, through a process that breaks down one or more particular models into parts, for analysis and alignment of those component parts towards the objective of building executable models. However, it also becomes apparent that a clear, holistic picture for Executable Architecture Specifications has not yet emerged: that is, there is a perceived need to develop Executable Architecture Specifications that include both a static and dynamic perspective, within the context of related components.

**Table 9 - Literature Topics, Findings and Gaps (1)**

Category	Topic	Author	Year	Findings	Gaps	Gaps to Topic
ADL	Federation Performance Analysis	Petty, McKenzie, Xu	2002	>Simulated the data flow in a federation using Rapide & ACME (proprietary) >Estimated the number of entities that a federate could support	\$\$ADL Elements to Executable Architecture Mapping	√
		Petty, McKenzie, Xu	2004	>Application of ADL to federate performance analysis >Predictive analysis (robustness, composability, knowledge transfer, and risk reduction)		
Structured Modeling and Transforms	Structured to CPN	Wagenhals & Levis	2000	>Architectures are described & interpreted in the context of Structured Analysis >Explores process for creating essential products of the DoD C4ISR Arch. Framework >Assertion: Structured Analysis bias in its representation of the products >Show products provide necessary info for the derivation of an executable model		
	Structured to Agent	Zinn	2004	>Case study on Air Force AOC architecture used to build Agent based simulation >Investigated utility of DoDAF architecture for providing basis for agent simulation >Reliance on OV 5 & OV 6a (DEF3) >Conclusion: DoDAF provides the needed information (not clearly demonstrated)	\$\$Sufficiency of DODAF	√
OO Transforms	OO Arch. Dev	Bienvenu, Shin & Levis	2000	>Provides A UML-based process for developing C4ISR architectures >Demonstrates the feasibility of developing C4ISR architecture descriptions using UML		
	OO to CPN	Wagenhals et al.	2002	>General description of an architecting process based on the UML >Rationale for style constraints on Models for building DoD C4ISR architectures >Describes a mapping between the UML & an executable model based on CPN	\$\$Resources and the effects on the CPN model \$\$Mapping other languages (e.g., BPMN) & CPN	√
	OO to DEVS Simulations	Ziegler & Mittal	2005	>Described method for transforming DoDAF architectures to a DEVS representation >Table of correspondences between DoDAF models (Views) and DEVS components	\$\$Absence of integrated modeling and simulation support in DoDAF	√
		Risco-Martin et al.	2009	>Described the essential mappings between UML and DEVS >Propose a design flow and set of transformations to generate DEVS executable simulation from UML	>Auto-generation of sequence diagrams from XFD-DEVS specs \$\$Executable architectures based on DoDAF \$\$SysML to DEVS mappings	√
	DoDAF Extensions & Modifications	Mittal	2006	>Addressed DoDAF extensions to support DEVS based modeling >2 new OVs: OV-8 (activities and interfaces) & OV-9 (nodes, entities & activities mapping)	\$\$Personnel management and task assignment at proper resolution of architectural execution >Evaluation of multi-resolutional architectures	√
		Mittal et al.	2006	>Described a means for semantically strengthening the OV 6a Rules Model >Present the semantic structure for OV 6a to aid the dev of semi-automated models >Application of Domain Meaning, Units of Measure (UOM) & domain specific rules >Describe how OV 6a can be structured in a more generalized meta-model framework such that every rule is reducible to meaningful code	>Quantifiability of terms in OV6A \$\$DoDAF is missing a rule-based structure that would allow different architectures to be used for multiple designs	√
DUNIP	DEVS Unified Process (DUNIP)	Mittal et al.	2007	>DEVS Unified Process (DUNIP), uses DEVS for SE and testing (Bifurcated Model) >XML based DEVS Modeling Language (DEVSMML) (model portability) >Supports distributed models deployment over SOA Middle-ware >Methodology for testing any proposed SOA based integration infrastructure	>DoDAF transformation to DUNIP \$\$QA issues associated with DUNIP >Study of risk associated with adopting DUNIP (cost / perf.)	√
Executable Extensions to Combat Sims	Combat Simulations & Exarch	Pawlowski & Ring	2004	>Process models in context of combat SIMS >DoDAF-based architecture products to CPN based executable model >Dynamic analysis of a system performance, effectiveness & resource utilization	\$\$Develop the next-gen of combat sims raw data from standard DoDAF >Investigation of message length representation associated IERs for passing to the coms net model to determine time for sending >Research integration of cost metrics for both static and exarch	√
	Context Analysis	Garcia	2010	>Context based analysis of executable architectures	\$\$Agent-based process model interaction with combat SIM	√

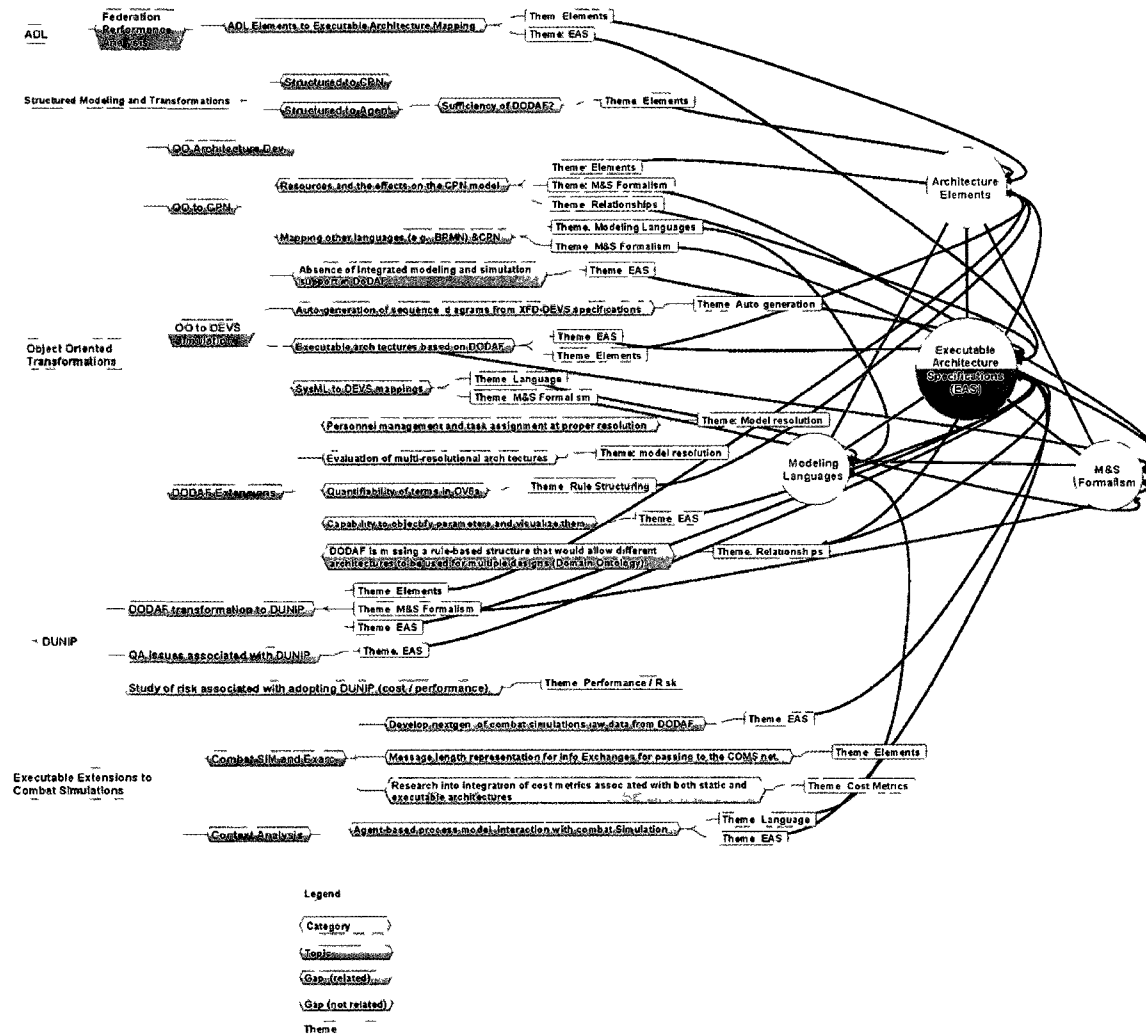


Figure 5 - Literature Themes to Gaps to Architecture Framework Map

## 2.5 Theoretical Framework

Figure 6 is designed to illustrate the theoretical observations that I drew from the literature on Executable Architectures, in which Levis, Mittal, Pawlowski, Wagenhals, Zeigler, and Zinn and others investigated the transformation of static DODAF architectures into dynamic executable architectures.

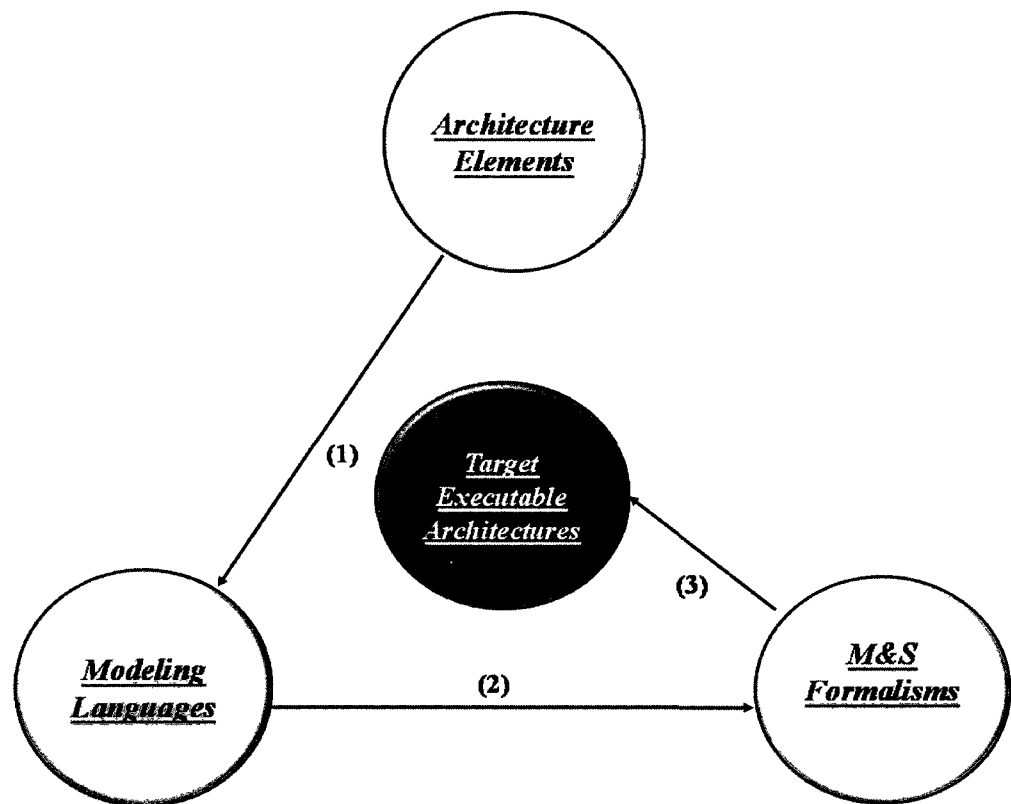


Figure 6 - Building Theory

The figure suggests that specific components used in the Development of Executable Architecture can be generalized into the following conceptual categories: DODAF into Architecture Elements, IDEF, UML, SysML, BPMN generalized into Modeling Languages, and Coloured Petri Nets and DEVS are generalized into Modeling and

Simulation Formalisms. These 3 conceptual categories are the foundational building blocks leading to the next level of theory.

In general, the research followed three steps to come up with use case specific target implementations:

1. Static Models based on DODAF were developed using specific modeling language implementations (UML, IDEF, etc.).
2. These static models were then converted into dynamic implementations based on CP-net or DEVS (M&S Formalisms).
3. This resulted in a target Executable Architecture.

In the context of these four concept categories, the question then arose as to whether there were other relationships.

## **2.6 Executable Architecture Concept Triangle**

Figure 7, the Executable Architecture Concept Triangle (EACT), represents a theoretical framework or conceptual guide for inquiry into the dimensions of executable architectures. A theoretical framework provides a conceptual guide for choosing concepts to be investigated and for suggesting research questions (Corbin & Strauss, 2008). It is “not as common in qualitative research, but in some instances can be useful.... if the researcher is building upon a program of research or wants to develop middle-range theory, a previously identified theoretical framework can provide insight, direction and a useful list of initial concepts” (Corbin & Strauss, 2008).

Initial results of this research were presented in (Shuman, 2010). The research, derived from observations of current approaches (Levis & Wagenhals, 2000; S Mittal, 2006; B. P. Zeigler & Mittal, 2005), hypothesized that three component categories are needed to define a set of potential elements for an executable architecture. These categories are architecture elements, modeling languages and modeling and simulation formalisms. A theory of executable architectures must ensure that the architecture can be described completely and consistently through all three components. All elements captured in the Architecture Elements need to be part of the formalism and should be the subject or object of activities modeled with the Modeling Language.

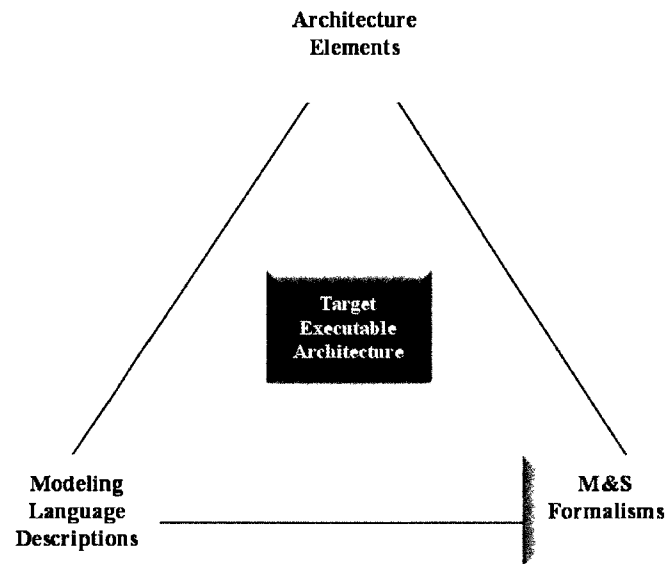


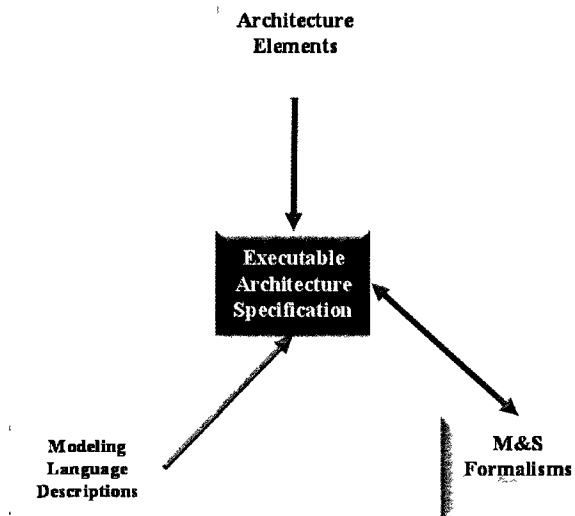
Figure 7 - Simplified Executable Architecture Concept Triangle

These component categories are further described as follows:

- **Architecture Elements:** An architecture framework (AF) defines the architecture elements and their relationships to each other in the context of various models or views (DOD, 2007a). Architecture Elements are the building blocks of architecture, and they define the WHO, WHAT, WHERE, HOW, WHY and WHEN parts of an architecture.
- **Modeling Languages:** Modeling Languages describe the dynamic, relational and conditional aspects of systems. They utilize graphical, symbolic & standard notations, and provide rich descriptions & specificity.
- **Modeling & Simulation Formalisms:** Modeling & Simulation Formalisms provide standard mathematical notations for elements & relationships with respect to Dynamic modeling. They provide high level, abstract descriptions. M&S Formalisms are useful for Validation & Verification (V&V).
- **Target Executable Architecture:** The Target Executable Architecture is the target or resulting specification that is defined through the other three components.

In the process of reviewing the literature, it was observed that in the Architecture Frameworks the interrogative elements *Who*, *What*, and *Where* are sufficient for static modeling; however, *When*, *How*, and *Why* are insufficient for dynamic modeling (i.e., simulation). Sage and Rouse (2009) described these elements in terms of Information and Knowledge Interrogatives. As discussed in Chapter 1, the inclusion of simulation capability in an architecture framework would provide an order of magnitude greater capability in model verification, validation, plausibility analysis, and performance analysis to include timing, resource, and cost constraint analysis. In order to achieve integrated simulation capability in the context of an Architecture Framework, the simulation components must be designed into the static modeling framework in a complementary way – in a way that includes those dynamic elements related to time, process, and rules that are necessary to specify process dynamics.

It became apparent that many deficiencies could be addressed through modeling languages, and one way to address these deficiencies would be through meta-model development such that modeling language elements could be included into a meta-model based on a source Architecture Framework. Such a meta-model could theoretically support *simulation* in the context of an architecture framework. To this effect, the idea for an **Executable Architecture Specification** (EAS) meta-model based on Architecture Elements & Modeling Language Descriptions emerged. Figure 8 illustrates the thought process that led to the idea for the development of the EAS, shown at the center. An additional aspect of the process would be to conduct a plausibility analysis of the EAS by comparing elements and relationships in M&S Formalisms (CP-net & DEVS) to the EAS.



**Figure 8 - Idea for Executable Architecture Specification**

The Executable Architecture Specification is a meta-model. A meta-model is a model that defines the components of a conceptual model, process, or system (Booch, et al., 1999). A meta-model is a special kind of model that specifies the abstract syntax of a modeling language (meta-model, 2011).

The following relationships were explored in the context of the study (Figure 9):

- Architecture Elements form the baseline for the EAS;
- Architecture Elements utilize Modeling Languages;
- Modeling Languages are used to build Architecture models or views;
- Modeling Languages inform Executable Architecture Specifications;
- M&S Formalisms validate Executable Architecture Specification (EAS);
- EAS conforms to M&S Formalisms.

Italics and dashed lines represent potential relationships (these are outside of study scope). These relationships support the development of the Executable Architecture Specifications (EAS).

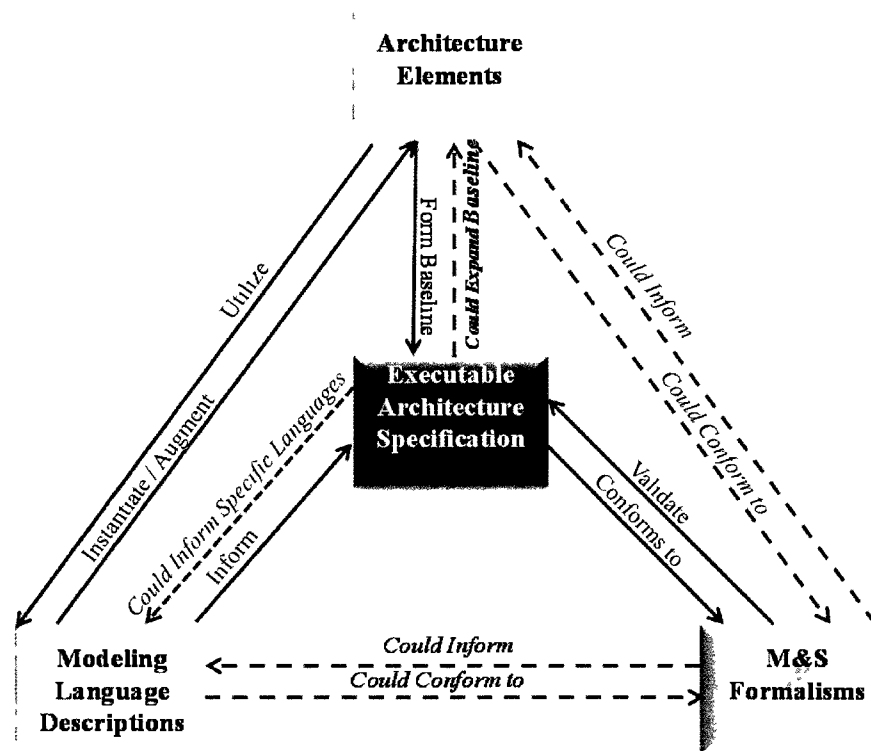


Figure 9 - Relationships Explored

All these components & relationships working together I call the Executable Architecture Concept Triangle (EACT). Figure 10 shows the Executable Architecture Concept Triangle (EACT). It is a UML Class Diagram showing the primary components of Executable Architecture and their relationships. In the center, the EAS is shown with elements categorized according to information interrogatives (semantics), in relationship to each other (Syntax). Both the EACT and the method for developing an EAS were developed, shaped, and refined in the course of the dissertation research.

This dissertation addresses the development of executable architectures in a way that can provide a holistic treatment of the problem space: that can delineate more fully what is missing and what is needed, through examination of the problem space holistically, from the perspective of the key components in the Executable Architecture Concept Triangle: Architecture Elements, Modeling Languages, and Modeling and Simulation Formalisms, and the Executable Architecture Specification.

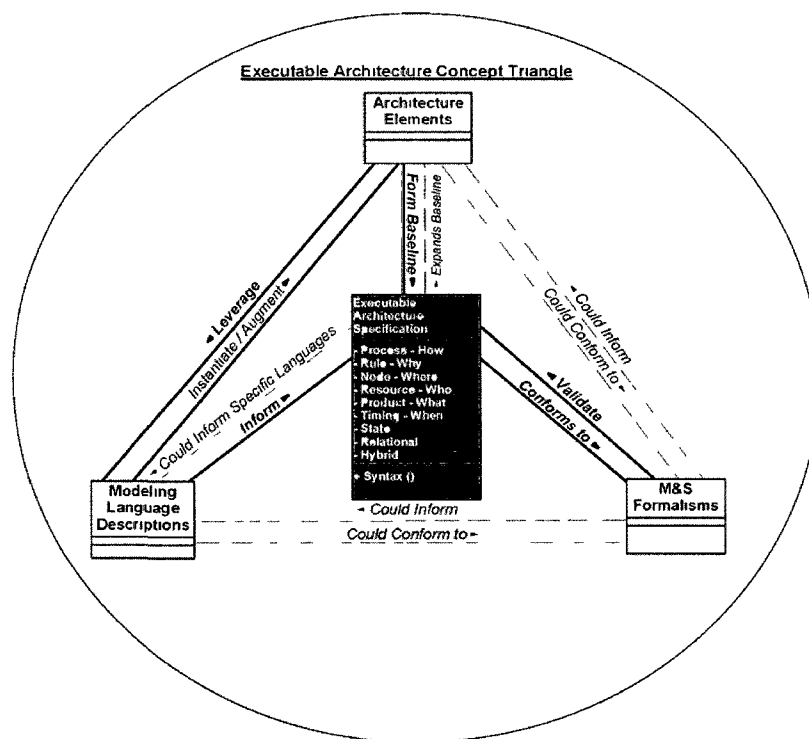


Figure 10 - Executable Architecture Concept Triangle

## 2.7 Transition from Theory to Method

Figure 11 illustrates the transition from theory to method. It shows three examples of the twenty meta-models that were developed in the course of this research through interpretation of source meta-models, one from each of the three EACT component categories. Elements were color coded according to the interrogatives, and parent-child relationships were established. Source models were analyzed according to type and aligned into groups (process, state, timing, node). Then the groups of models were synthesized into group composite models. The four group composite models were then combined into one composite: the EAS, a composite of composites.

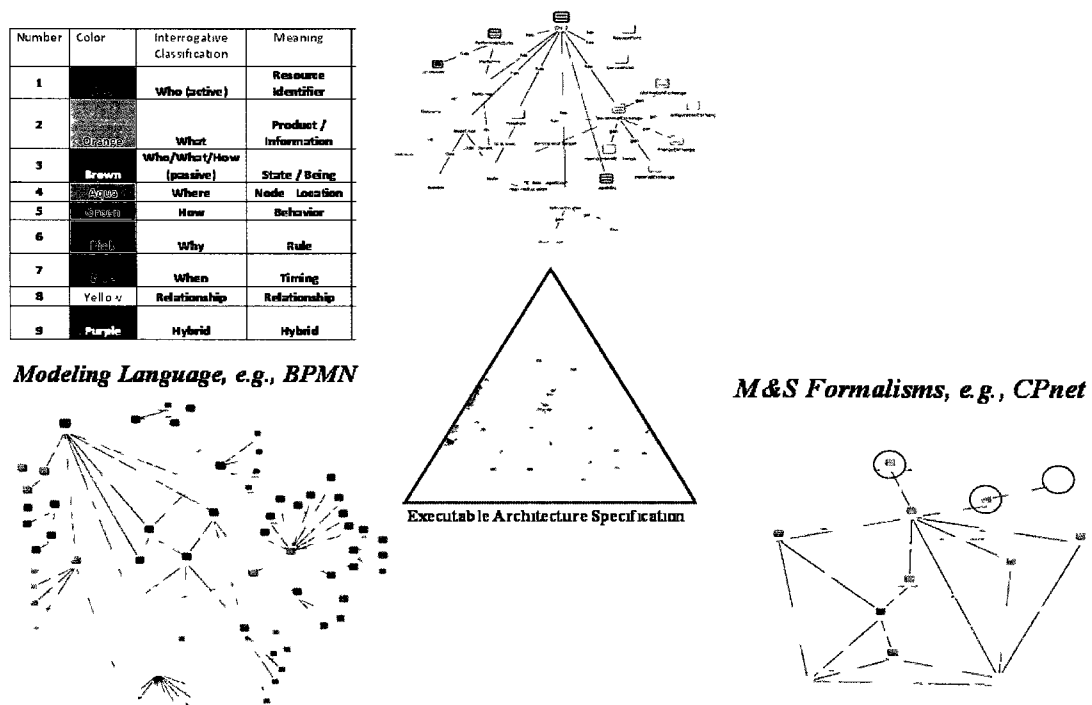


Figure 11 - Transition from Theory to Method

## **CHAPTER 3**

### **RESEARCH METHODS (QUALITATIVE RESEARCH)**

Many researchers believe that all inquiry starts out in a qualitative form (Lauer & Asher, 1988), (Leedy & Ormrod, 2010): “When little information exists on a topic, when variables are unknown, when a relevant theory base is inadequate or missing, a qualitative study can help define what is important” (Leedy & Ormrod, 2010). A qualitative study is useful when a study is exploratory, a concept or phenomenon is under investigation, or a concept is immature due to lack of theory (Creswell, 2009). The characteristics of a qualitative study include:

- Multiple sources of data,
- Emergent design (plan of research cannot be tightly prescribed),
- Inductive data analysis (bottom up),
- Interpretive study,
- Holistic: multiple perspectives, complex picture.

This research study includes all of the above characteristics: multiple sources of data such as source meta-model information from Architecture, Modeling Languages and Modeling and Simulation Formalisms; emergent design, in that the method evolved from conception to implementation; inductive data analysis, in that analysis started at the elemental level and proceeded to higher levels of organization; interpretive study, in that, the organization and categorization of elements was subject to interpretation and some ambiguity, as inherent in ontological organizational schemes; holistic, in that the analytical method sought to explore the problem space from more than one perspective to create a unified, derived result set, which is the Executable Architecture Specification.

#### **3.1 Type of Design and Underlying Assumptions**

The qualitative research design in this dissertation study has been based on data collection and coding techniques associated with elements of Grounded Theory (Glaser & Strauss, 1967). Grounded Theory is rooted in the concept that human dynamics and symbolism are intertwined. To provide a philosophical perspective on Grounded Theory, classically, its domain of inquiry is socio-psychological, which tends to be fairly

subjective, and anti-positivistic. To define: “anti-positivism: knowledge is soft, more subjective, spiritual, or even transcendental – based on experience, insight, and essentially of a personal nature.”(Flood & Carson, 1993); “positivism: knowledge is hard, real and capable of being transmitted in a tangible form” (Flood & Carson, 1993). On a research scale between positivism and anti-positivism, this study leans significantly to the positivist side, yet as a qualitative exploratory study, interpretations must be filtered through the interpretive lens of the author’s domain experience, which is necessarily subjective, or anti-positivist. In this study, the author has leveraged elements of Grounded Theory but has been cognizant of differences. To analyze the potential elements of executable architectures, large volumes of raw data needed to be collected and analyzed in a systematic way for patterns and relationships to emerge; hence, the data collection and coding methods utilized in grounded theory have been very useful. The focus of this study has been modeling language meta-models, which tend to be objective or positivistic yet still vulnerable to the impreciseness of symbolic – linguistic, verbal representation.

### 3.2 Grounded Theory Background

Grounded theory is a qualitative analysis methodology that gets its name from the concept that theory is induced from the data rather than preceding it, an inductive rather than deductive approach (Corbin & Strauss, 2008). It is rooted in **Symbolic Interactionism** (Cutcliffe, 2000). “*Symbolic Interactionists* stress that people construct their realities from the symbols around them through interaction, therefore individuals are active participants in creating meaning in a situation” (Cutcliffe, 2000). Symbolic Interactionism is rooted in Pragmatism, the maxim of which is “Consider what effects, which might conceivably have practical bearings, we conceive the object of our conception to have. Then, our conception of those effects is the whole of our conception of the object” (Peirce, 1998).

Grounded theorists search for patterns and processes to understand how a group of people define, via their social interactions, their reality (Cutcliffe, 2000). There are three primary branches of Grounded Theory, as follows (Cutcliffe, 2000):

- *The Systematic Approach* – (Corbin & Strauss, 2008) prescribes procedures in the form of coding categories and subcategories and development of visual diagrams to present the theory, concluding with explanations of relationships.
- *The Emerging Approach* - (Glaser, 1991) focuses on connecting categories and the identification of emerging theories, and does not force theory into categories.
- *The Constructivist Approach* - (Charmaz, 2000) is more subjective, with the emphasis on feelings, assumptions, and meaning making by study participants.

The approach taken in this research is consistent with the Systematic Approach, in that it is heavily reliant on data coding, category and subcategory allocation of data, and visual methods and mappings, for the development of theory and explanations.

There are a number of points of debate related to grounded theory. These criticisms concern sampling, literature review, creativity and reflexivity, and precision in method (Cutcliffe, 2000).

### 3.2.1 *Sampling (theoretical versus purposeful)*

There is some debate concerning the nature of sampling, whether it should be driven by emerging theory, such that data sources are chosen based on the emerging hypothesis and sample size is based on completeness of findings with respect to given categories of investigation (saturation); or whether the data sampling should be based on purposeful strategies (purposeful sampling). Some advocate for a compromise position in which the initial sampling is purposeful (to delimit), then moving to theoretical sampling as patterns emerge. This last method is closest to what has been used in this study (Cutcliffe, 2000).

### 3.2.2 *Creativity and Reflexivity (Interaction between the researcher and the world being studied)*

Some acknowledge that the experience the researcher brings to the field of inquiry may be enriching to the end result, while others advocate for a more neutral mindset in the approach. In other words, a certain degree of subjectivity on the part of the researcher is unavoidable, and may increase creativity. In the case of this study, the experience of the author in the field has been found to be essential to the navigation of the data sets in question (Cutcliffe, 2000).

### 3.2.3 *Literature Review (beginning or end)*

Some authors advocate for minimizing the literature review at the beginning, to foster the possibility that emergent theory will be grounded in the data. Others argue that literature review should precede data collection and analysis because the literature review can help identify the current gaps in knowledge or help provide a rationale for the proposed research (Cutcliffe, 2000).

### 3.2.4 *Lack of Precision*

One further criticism centers on “method slurring” or mixing of methods, such as mixing with phenomenology, which also uses coding. There is another criticism directed toward deficiencies of method, such as the absence of theoretical coding. Conversely, there are those who advocate for method evolution, suggesting advantages such as a more thorough, multi-dimensional analysis of phenomena. Cutcliffe (2000) cites Stern (1994)), who advocates for clear, purposeful intent with respect to method mixing. In other words, regardless of the methods chosen, there should be a clear and conscious recognition and articulation of the nature of the methodology, whether mixed or classical.

### 3.2.5 *Conclusions with respect to Grounded Theory Criticisms*

In this study, sampling has been generally purposeful but has responded to theoretical sampling concerns as patterns emerged. Sample size has been based on completeness of findings with respect to given categories of investigation (saturation). Again, the experience of the author in the field has been crucial to the navigation of the data sets in question, and the literature reviews have preceded data collection. This has been the basis for the determination by the author that there is a need for a common theoretical framework and method, for development of that theoretical framework and method, and has been the basis for the rationale for this research. The method chosen relied on Grounded Theory coding methods for traceability; but the method departed from Grounded Theory in that it was not focused heavily on emergent symbolic meaning. Furthermore, the object of this study is well defined, finite, and structurally known to the author, setting the stage for the way Grounded Theory is used to populate the tool of choice.

### 3.3 Data Collection, Coding and Analysis, and Theory Development

Data collection and analysis was facilitated using data coding techniques described in grounded theory coding, which is a qualitative analysis methodology, developed by Corbin and Strauss (2008). Inductive knowledge was produced by applying grounded theory to the elements of the components of the EACT (i.e., Architecture Frameworks, Modeling Languages, and Modeling and Simulation Formalisms), which was then synthesized resulting in the final EAS model that comprises all elements and relationships. It should be noted that systematic data collection and analysis have been critical to this study for elemental traceability from authoritative data source through each derived use in model synthesis.

A meta-model describes the constituent elements of a model and the relationships between these elements in terms of semantics and syntax. The components of the concept triangle are well described through authoritative meta-model descriptions. This study has used the UPDM meta-model for architecture models (OMG, 2009a). Language meta-models for UML and SysML and BPMN are available from OMG (OMG, 2006, 2009, 2009). DEVS (Zeigler, et al., 2000) and CP-net (Murata, 1989) are well documented through formal descriptions. The objective of data collection has been to organize elements and to learn as much as possible about them, finding any disconfirming evidence that may suggest revisions in the categories identified or in interrelationships among them. This study will leverage a constant comparative method, moving back and forth between data collection and data analysis, with data analysis driving later data collection. Theory development has been based on exploring data categories and relationships. Data collection and analysis proceeds through the following steps, as illustrated in Figure 12:

- 1) Collect Data.
- 2) Scrutinize data & search for patterns.
- 3) **Code:**
  - a. **Open:** Develop Categories or Themes. (**Categories, Properties, Attributes**)
  - b. **Axial:** Place data into categories or themes. (**Binning**)
  - c. **Selective:** Observe relationships revealed and how they combine to form a story line to describe phenomenon. (**Reduction**)
- 4) **Compare:** Repeat steps 1, 2 and 3 as additional data are collected.

- 5) **Develop theory:** Combine storylines to develop a theory -- in the form of a verbal statement, visual model, or series of hypotheses -- to explain the phenomenon in question (Corbin & Strauss, 2008).

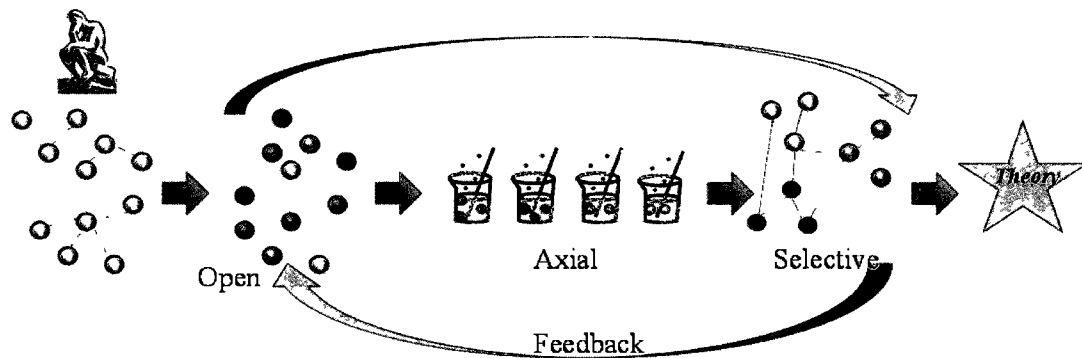


Figure 12 - Data Collection and Analysis

Figure 12 shows a stylized depiction of data collection and analysis. It starts on the left side with loosely organized data; proceeds through Open Coding, which is categorization of the data; to Axial Coding, which entails organization of coded data; to Selective Coding, in which relationships are established and duplications are eliminated. The method involves constant comparisons, repeating steps 1, 2 and 3 as additional data are collected. Theory is developed in the form of a verbal statement, visual model, or series of hypotheses -- to explain the phenomenon in question (Corbin & Strauss, 2008). The result is theory development, in which there is an emerging picture of categories, meaning and relationships.

### 3.4 Data Collection and Analysis: Sources & Tools

Data was collected for each of the three main components of the EACT: Architecture Elements, Modeling Language Descriptions, and M&S Formalisms. In this research the data consists of elements (semantics) & their relationships (syntax) in meta-models and formalisms. For Architecture Elements, data was collected from Process Modeling Operational Views (OV) from Unified Profile for DODAF and MODAF (UPDM). The source was the Object Management Group (OMG). For Modeling Language Models, data was collected from process and structure models from IDEF,

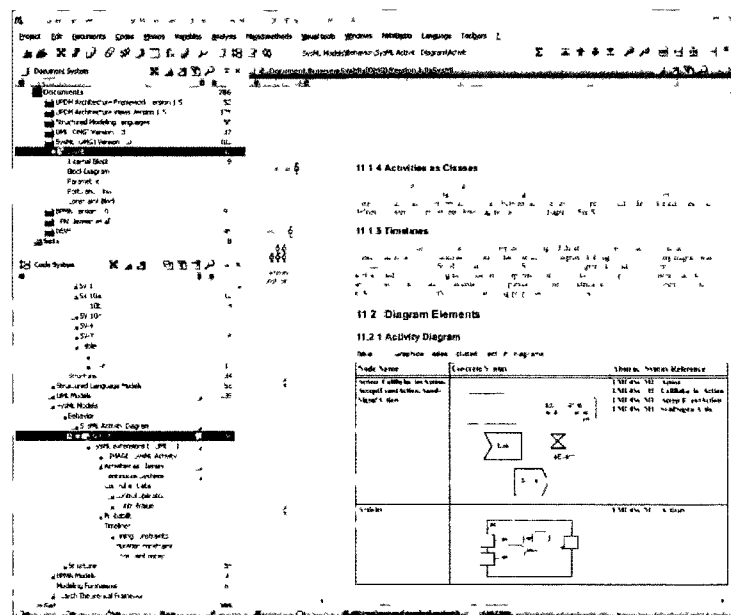
UML, SysML and BPMN. The source was OMG and Integrated Definition Methods (IDEF). For Formalisms, data was collected from DEVS and CP-net specifications from Zeigler, Jensen. The M&S formalism focus here is Discrete Event Simulation, not *\* not Differential Equation System Specifications (DESS)*.

Data source selection was purposeful. Data elements and relationships were collected from the following meta-model sources:

- **Architecture views**
  - **Focus:** UPDM meta-models, Operational View (OV) Process Models.
  - **Source:** Unified Profile for DODAF and MODAF (OMG, 2009a).
  - **Reasons chosen:** representative sample, based on DODAF and MODAF & similar to DNDAF and NAF; DODAF is used extensively across DOD.
- **Modeling language models**
  - **Focus:** IDEF, UML, SysML and BPMN.
  - **Source:** OMG (OMG, 2006, 2009, 2009) , & IDEF (IDEF, 2010) (DeMarco, 1979) descriptions.
  - **Reason chosen:** Broad usage in modeling community, referenced extensively in literature.
- **M&S Formalisms**
  - **Focus:** DEVS (Zeigler, et al., 2000) and CP-net.
  - **Source:** “Theory of Modeling and Simulation” (Zeigler, et al., 2000) and “Coloured Petri nets basic concepts, analysis methods, and practical use” (Jensen, 1992).
  - **Reason chosen:** Broad usage, broadly representative.

In order to conduct Grounded Theory-based coding, several necessary principles became apparent: element traceability from source, identification and building of element relationships (i.e., generalization, composition, and association relationships), and visualization of elements. In order to conduct Grounded Theory-based coding on the large volume of data elements that comprise the EACT, it became apparent that a tool would be needed that could also provide an integrated capability, enabling reproducibility of results, and facilitating ease and speed of coding.

Figure 13 is a snapshot from MAXQDA that shows the 3 data collection windows: the **Document Browser** window (right side), the **Code System** window (lower left), and the **Document System** window (top left).



**Figure 13 - MAXQDA Data Collection Windows**

The Document System window provides a means to organize imported documents. It is a catalogue of source material that is subdivided into Text Groups. A Text Group is a container or folder for grouping text information relevant to that group. The Text Group is populated by files relevant to that Text Group. MAXQDA accommodates .pdf, .rtf and .doc files.

The Document Browser provides a way to review documents and import key text and pictures into the Code System through in-vivo data coding.

The Code System window is populated through in-vivo coding. Codes may then be organized using hierarchical arrangements to support composition and generalization relationships. Code memos can be associated with each code, which is useful for providing amplifying information (e.g., definitions and snapshots of meta-models).

MAXQDA provides visual tools, one of which is called MAXMAPS which supports insertion and traceability of elements (from the Code System to MAXMAPS), insertion of sub-codes, depiction of code colors (for visual categorization), synchronization between code objects in the MAXMAPS window and the Code System (to include traceability back to the supporting Document in the Document Browser), and the development of visual links between MAXMAPS objects.

Figure 14 shows a sample MAXMAPS window. It has three principle panes. The left pane shows the names of visual maps in the system. The center frame shows the map itself, in the case of Figure 14, the OV-5 meta-model. The right pane shows diagram layers that can be associated with particular objects in the map. Layering provides the ability to selectively view objects associated with different layers. This feature is particularly useful in a complicated model, where simplification may be necessary as part of model analysis. Each map is comprised of objects and links.



- Constraint-oriented Modeling,
- Spatial modeling.

According to Fishwick (1995), conceptual models embody entities and relationships where entities have not been clearly identified in terms of state, event, and function. A declarative model is comprised of states and events. This type of modeling is good for modeling a system that has discrete states or events or where there are phases of a process. Functional models are graphs that contain two key components: functions and variables. Fishwick recommended the functional approach if the modeling problem suggests description of the system in terms of objects with functions. Functional or procedural modeling relies on functional elements as the building blocks for the development of a dynamic model.

This research has been limited to executable process modeling and to the model classifications of conceptual, declarative and functional categories. Constraint-oriented and spatial modeling are outside of the scope of this investigation. This delimitation reduces the scope of this study and is consistent with observations of the literature with respect to Executable Architectures. Previous efforts have focused their studies on these modeling areas but not explicitly by reference to Fishwick's taxonomy (Mittal, 2006; Mittal, et al., 2006; Pawlowski III, et al., 2004; Risco-Martin, et al., 2009; Wagenhals, et al., 2002; Zeigler & Mittal, 2005).

### **3.6 EACT Process Flow Chart**

Figure 15, the EACT Process Flow Chart, shows the general pattern that was followed for data collection and analysis. The EACT Process Flow Chart is based on the EACT, which is shown as an insert, in the upper right of the figure. Data were collected and analyzed for each EACT component, using MAXQDA. Meta-models were coded using Open, Axial, and Selective Coding. First meta-models were coded for Architecture Elements, then for Modeling Languages, then for Modeling and Simulation. Each of the steps within the larger rectangles represents a stage of coding and analysis. The large flow chart boxes are numbered showing the sequence of steps in data collection and coding to build the EAS:

- I. Architecture Elements,
- II. Modeling Language Descriptions,
- III. M&S Formalisms,
- IV. Executable Architecture Specification (EAS).

Steps I-III contributed elements that were later selectively coded to build the EAS.

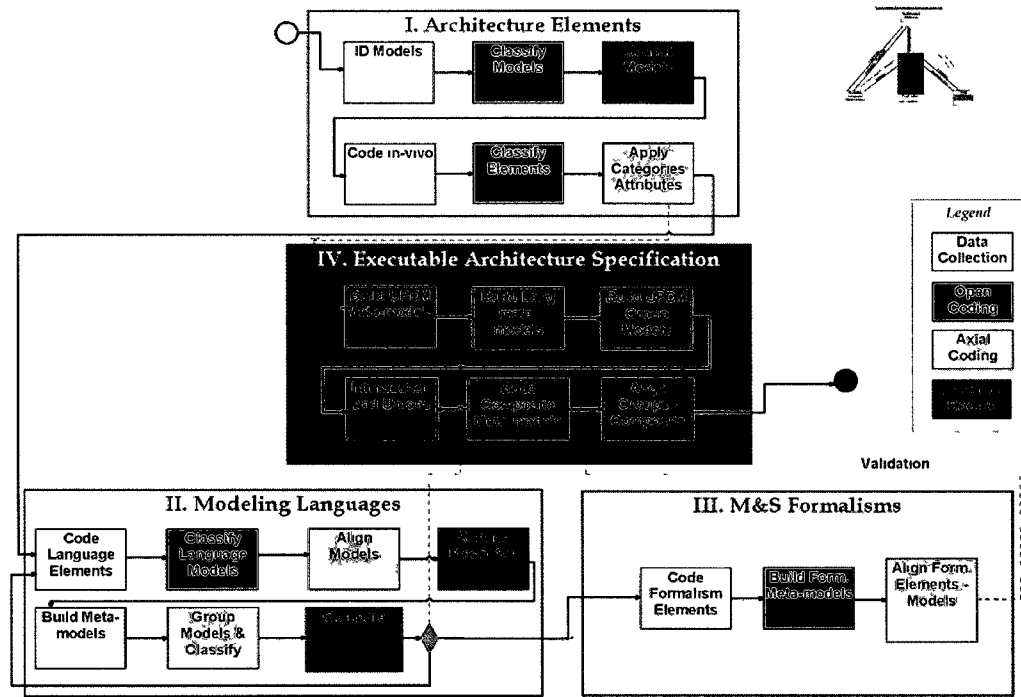


Figure 15 - EACT Process Flow Chart

### 3.7 Data Collection and Analysis of Architecture Elements

Figure 16 provides a more detailed view of the coding process with MAXQDA for Architecture Elements (light blue Architecture Elements box from Figure 15). A table illustrating the first 3 steps of UPDM model identification and selection is shown on the upper right. The last three steps appear along the lower half of Figure 16.

**Step 1:** Identify the target architecture framework set (i.e., UPDM) (**Collect Data**).

**Step 2:** Classify the Architecture Framework models according to types (**Open Coding**).

**Step 3:** Delimit the target architecture set into relevant process models (**Selective Coding**).

**Step 4:** Collect data using in-vivo coding in MAXQDA (**Collect Data**).

**Step 5:** Identify the element categories in MAXQDA (i.e., interrogatives, generalization and composition relationships, etc.) (**Open Coding**).

**Step 6:** Apply categories and attributes to the model elemental set (**Axial Coding**) and establish relationships (**Selective Coding**), using MAXQDA.

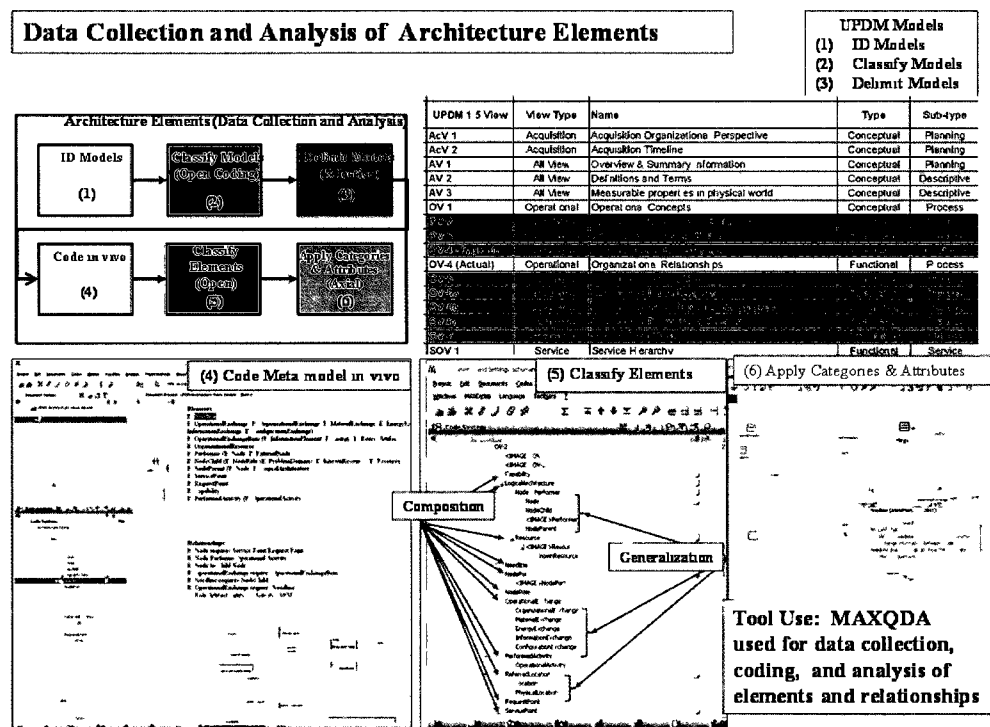


Figure 16 - Data Collection and Analysis of Architecture Elements

### 3.8 Validity

The research project addressed both internal and external validity concerns.

Internal validity means that there are sufficient controls to ensure that the conclusions drawn are warranted (Leedy & Ormrod, 2010). External validity touches on our ability to make generalizations about the world beyond the specifics of this study (Leedy & Ormrod, 2010).

To ensure internal validity in this study, it has been the intent of the author to take all precautions to ensure quality of process and result. The following validation enhancing and mitigating strategies were pursued:

- **Data Triangulation supports internal validity** – Collection of related data from multiple sources should lead to data convergence, thereby substantiating the conceptual framework and the data focus themselves (Leedy & Ormrod, 2010). In this study, data was collected in accordance with the Executable Architecture Concept Triangle, from the UPDM Architecture Framework, from a variety of different Modeling Languages and from two representative and broadly used Modeling and Simulation formalisms, in order to drive a convergence from multiple sources towards the Executable Architecture Specification.
- **Thick description supports internal validity** – The concept suggests an approach where the situation is described in sufficiently rich detail that the readers are able to form their own assessment of the data presented (Leedy & Ormrod, 2010). The detail provided in the data collection and analysis should provide enough detail for the informed readers to form their own opinions.
- **Feedback from others supports internal validity** – Here, the researcher has sought the opinion of dissertation committee and other domain experts. (Leedy & Ormrod, 2010). These persons have long standing expertise in modeling and simulation, and are themselves published authors in the field of modeling and simulation, to include specific expertise in DODAF, UPDM, UML, SysML and BPMN.
- **Representative Sample supports external validity** – The choice of UPDM, which is an offshoot of DODAF and MODAF, is suggestive of the generalizability to other Architecture Frameworks. The choice of a variety of

modeling languages, from UML to BPMN, suggests that the method is generalizable to other models, and the choice of DEVS and CP-net, each with a slightly different perspective on modeling, yet representative of discrete event simulation, suggests generalizability to other M&S Formalisms.

In summary, the method articulated in this chapter is qualitative and exploratory. The research design in this dissertation study is based on data collection and coding techniques associated with elements of Grounded Theory. The method will step through data collection, coding, analysis and theory development leveraging MAXQDA, which is a tool that conforms to the coding and visual representation needs of this dissertation. The method will leverage the Executable Architecture Concept Triangle (EACT), and each of the source components of the EACT: Architecture Elements, Modeling Languages and Modeling and Simulation Formalisms to develop theory related to Executable Architecture Specification development.

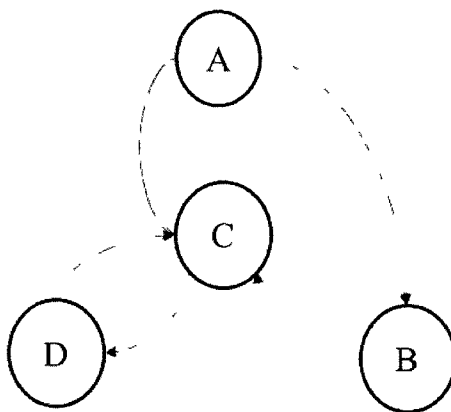
## CHAPTER 4

### DATA COLLECTION AND ANALYSIS

The data collection and analysis process will be described at two levels: first, at the higher level, which provides an overview of the entire process; secondly, at the lower level, affording a more detailed discussion of the various parts of the process, and how they link together to form the whole. The higher level can be described as more abstract; the lower level as more concrete.

#### 4.1 Data Analysis and Findings High Level

Figure 17 is a graph that depicts the major steps in the project associated with the data collection and analysis of executable architecture elements. This section describes at a high level the method used for investigation of both the semantics and syntax of executable architectures.



**Figure 17 - Data Collection and Analysis – High level**

**Step A: Selection of Baseline Models and Data Sets:** In step A, the baseline models and target data sets were selected. The starting point was selection of a bounding and scoping architecture framework, as a point of departure. Unified Profile for DODAF and MODAF (UPDM) is a hybrid architecture framework that provides excellent meta-

models for data collection and analysis, specifically UPDM 1.5 (OMG, 2009a). This investigation leveraged and explored a focused set of UPDM operational process related views (e.g., OV-2, OV-5, etc.), related modeling languages (i.e., IDEF 0, UML, SysML and BPMN) and specific process-oriented model subsets (i.e., SysML Activity Diagram, BPMN Process Model, etc.) within those languages. The motivation for selection of these models is both extensive documented use in the literature and, in accordance with the experience of the author, broad use in the modeling and architecture community. Selected views from UPDM and modeling languages were analyzed in terms of both their elemental meaning, and their relationships to other elementals. Lastly, two well established and representative modeling and simulation formalisms (CP-net and DEVS) were chosen as a basis for comparison and validation purposes. Each of these formalisms is discussed in the literature review. Both are broadly discussed in the literature and have broad acceptance and usage in the modeling and simulation community. Each of these formalisms was explored through their respective descriptive meta-models.

**Step B Open Coding:** In Step B Open Coding was utilized, which was the identification of systems descriptive attributes. Sage and Rouse introduced six interrogatives into information and knowledge management, distinguishing between those that relate to information and those that relate to knowledge: who, what, where, and when refer to information while how and why deal with knowledge (Sage & Rouse, 2009). The six interrogatives are fundamental to defining knowledge management attributes, and in this project were useful in the element comparison phase (described later in Chapter 4). However, the interrogative set was subsequently expanded to 9 categories to accommodate some additional elemental types that did not fit nicely into the other categories. The specifics and motivation for this expansion are explained later in Chapter 4.

The following descriptive attributes were established:

- **Interrogative:** (i.e., who, what, where, when, why, how, etc.);
- **Color:** in parallel to interrogative attribute for visual reference;
- **Model Origin:** for tracking model source;
- **Operational or System Element:** to distinguish between elements coming from UPDM Operational or Systems models;

- **Model Group:** to distinguish between behavioral or structural models;
- **Parent Code:** to track parent child relationships for ontology building.

**Step C Axial Coding:** In step C, Axial Coding was utilized, which is essentially placing data into categories by assigning attributes. In this step elements were identified from specific models (e.g., UPDM OV-5, OV-2, UML Activity Diagram, Sequence Diagram, etc.), and tagged with the attributes identified in Step B. MAXQDA supports in-vivo coding, category development, object color coding and ontological relationships, and code mapping. For this reason it was chosen to support the process.

**Step D Selective Coding:** In Step D Selective Coding was utilized, which is the observation of relationships and how they combine to form a story line to describe phenomena, described simply as alignment and reduction. In Step D elements were organized, compared and aggregated through the use of visual maps of the elements, organizational data views, and queries of the elements based on attributes. A detailed data roadmap was then developed for guiding element organization, aggregation and comparison to facilitate analysis of the data elements. This step supported categorization by identification of identical elements, elements of the same equivalence class and identification of individual elements and their extensions. Elements were then analyzed in terms of interrogative attributes - first by model of origin, then with respect to other interrogative attributes. Elements were next placed into group meta-model visual maps, which eventually results in developing increasingly holistic composite UPDM-Language meta-model maps. Redundant or duplicative elements were then eliminated through visual inspection and through comparative queries of the elements, based on attributes. This led to development of a composite UPDM-Language meta-model along with a UPDM-composite meta-model, the comparison of which, revealed both the elements that are shared in common, as well as those elements from the language meta-model that are augments. As a result, the governing concepts of the Executable Architecture Specification, which are the executable architecture elemental meanings (semantics) and relationships (syntax), were derived and identified.

Figure 18 below was presented in Chapter 3 and is provided here again to reinforce the explanation of steps A-D above.

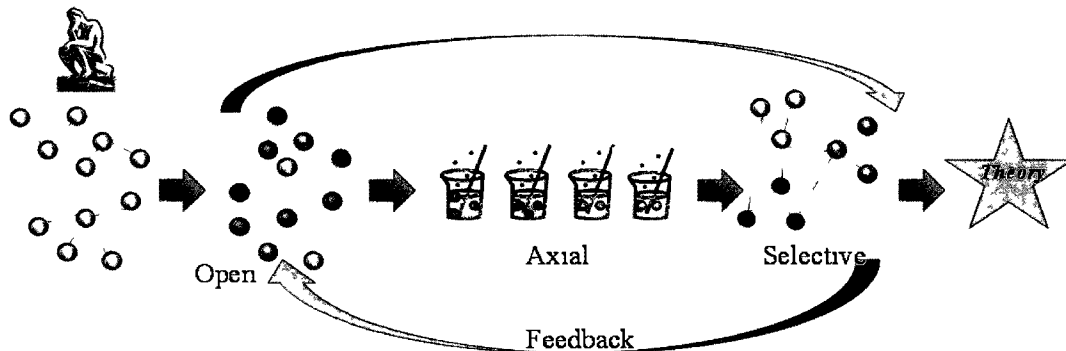


Figure 18 - Data Collection and Analysis

Figure 18 above shows a stylized depiction of data collection and analysis. It starts on the left side with the loosely organized data or elements (from source documents); proceeds through Open Coding, which is category or attribute development; to Axial Coding, which entails organization of elements into categories through application of attributes; to Selective Coding, which is alignment and reduction of elements. The result is theory development, in which there is an emerging picture of categories, meaning and relationships. The arrows indicate that data collection can, and often does drive further data collection and analysis. In other words, once the pattern emerges, the Selectively Coded data can then be re-analyzed through the same three steps, Open, Axial and Selective Coding, leading to further refinements of the data. Alternatively more data can be brought into the model to be analyzed through the same process, confirming the pattern.

## 4.2 Data Analysis and Findings: Detail Level

The preceding section provided a high level view of findings. The following section provides a low level, close-up view: a more detailed explanation of the data collection and analysis process and the findings.

### 4.2.1 Identification of Descriptive Categories (Open Coding)

Architecture Elements are the building blocks of architecture, and they define the who, what, where, how, why and when parts of an architecture. The Information Interrogatives are as follows: What (i.e., entities), When (i.e., time), Where (i.e., location) and Who (i.e., people). The knowledge interrogatives are as follows: How

(i.e., behavior), Why (i.e., purpose, motivation, or rule) (Sage & Rouse, 2009). Additionally, Garcia (2011) showed in his dissertation that the How and the Why belong to the context. In general, the Who, What, and Where address the static, structural elements of architecture. The How, Why and When are process oriented, and tend to be the dynamic elements in architecture. These six categories make a good starting place for investigating the elementals needed in the development of executable architectures because they address most of the key ontological perspectives. The data collection and analysis was started with the six aforementioned interrogatives as the basis for element classification; however, this list was almost immediately expanded because it became apparent that three additional categories were needed: Who / What / How (Passive) (i.e., State, or condition), Relationship (i.e., linking objects), and Hybrid (i.e., objects that have multiple category characteristics). The Who / What / How (Passive), hereafter simply referred to as State, is a way of expressing State in terms of interrogatives; it is framed in this way because a person or resource, a thing or product, and an activity can all have State. The relationships category was added to account for linking objects such as the IDEF0 Input, Control, Output, Mechanism (ICOM) arrow. IDEF0 is a key Modeling Language process model. Similarly, the Activity Edge and Control Flow are linking elements in the UML Activity Diagram, with is an Object Oriented process model.

The need for a relationship category became apparent when the color coded elements were placed in an ontological arrangement in MAXQDA. State was understood up front, but it did not fit nicely into the other ontological categories. Lastly there were objects that did not fit well into any of the above; these were the hybrid objects which have multiple interrogative characteristics. For example, the Capability element is suggestive of behavior (how), function (how), time (when), Rule (why), and Node (where).

**Finding:** The data collection and analysis was started with the six interrogatives as the basis for element classification, however, this list was almost immediately expanded because it became apparent that three additional categories were needed: State, Relationship, and Hybrid.

Table 10 provides a list of all 9 interrogative categories with descriptions. Each of the 9 interrogatives was associated with a color (as shown in Table 10) to support the

grouping of objects based on visual observation of element types. The terms “interrogative” and the associated color codes have been used interchangeably in this document. There are, of course, other interrogatives, such as How Many, How Much (COST), but it is arguable that these are attributes rather than fundamental categories. For this reason they are not used in this study.

From a theoretical point of view, what was needed was an open tool that supports in-vivo coding, category development, object color coding, relationship building, and visual mapping; for this reason MAXQDA was chosen to facilitate data collection and analysis.

**Table 10 - Color and Interrogative Classifications**

Number	Color	Interrogative Classification	Meaning	Description
1	Red	Who (active)	Resource Identifier	Person / or acting agent
2	Orange	What	Product / Information	Thing produced by or resulting from a process (e.g., information)
3	Brown	Who/What/How (passive)	State / Being	Condition
4	Aqua	Where	Node - Location	Operational Node
5	Green	How	Behavior	Process or Activity
6	Pink	Why	Rule	Modifier to Activity (e.g., context, rule, etc.)
7	Blue	When	Timing	Time descriptive or control element
8	Yellow	Relationship	Relationship	Linking or relational Element
9	Purple	Hybrid	Hybrid	Grouping of interrogative classifications

#### **4.2.2 Selection of Baseline Architecture Framework**

This section addresses the selection of an Architecture Framework for data analysis. DODAF was described in detail in the literature review. DODAF 2.0 (DOD, 2009) is the most recent version of DODAF. The main difference between DODAF 2.0



#### 4.2.3 *UPDM Target Set*

After selection of the Architecture Framework, the first task was to select the target architecture views for the study from the larger set. Architecture Frameworks provide standardized modeling constructs, bringing under one umbrella many different kinds of models. Different model views offer unique perspectives into a given system problem space, but not all views within an Architecture Framework are directly relevant to process focused executable architectures. UPDM, based on DODAF and MODAF, describes 45 views, divided into 7 view categories (All Views, Acquisition Views, Strategic Views, Operational Views, Standards Views, System Views, and Service Views).

As introduced earlier, Fishwick (1995) provides a taxonomy for models that classifies them as **conceptual**, **declarative**, **functional**, **constraint-oriented** and **spatial** models. Conceptual models emphasize entities and relationships; declarative modeling is focused on state and state change perspective. Functional modeling depends on functional elements as constituent elements, useful for the development of a dynamic model. This perspective is interesting but not very helpful here because all UPDM Architecture models fall into declarative, functional and, to a lesser extent, conceptual categories; constraint-oriented and spatial categories are out of scope. In the literature on executable architectures, we see that Wagenhals et al. (Wagenhals, et al., 2002), and Risco-Martin et al. (Risco-Martin, et al., 2009), and Levis (Levis & Wagenhals, 2000), all focus on process models of the of the Declarative, Functional and Conceptual Types, in development of Executable Architectures.

The focus of this study is Operational Process modeling. This eliminates system function views, planning views, capability views and technical views, and descriptive views, all shown as sub-types, in Table 11 (Planning, Descriptive, Process, Structural, Function, Capability and Technical).

The remaining operational views are either Process, or structural by subtype. Within the Operational views, the OV-1 was eliminated because it does not add any elements to the other OVs. The OV-4 (Actual) was eliminated as a duplicate of the OV-4. This left the OV-2, OV-3, OV-4, OV-5, OV-6a, OV-6b, and OV-6c and OV-7, all shown in Table 12.

Table 11 - UPDM Views

	A	B	C	D	F	G	H
	#	UPDM 15 View	View Type	Name	Functional or Declarative or Conceptual	Sub-type	Assessment
1							
2	1	AcV-1	Acquisition	Acquisition Organizational Perspective	Conceptual	Planning	NA
3	2	AcV-2	Acquisition	Acquisition Timeline	Conceptual	Planning	NA
4	3	AV-1	All View	Overview & Summary Information	Conceptual	Planning	NA
5	4	AV-2	All View	Definitions and Terms	Conceptual	Descriptive	NA
6	5	AV-3	All View	Measurable properties in physical world	Conceptual	Descriptive	NA
7	6	OV-1	Operational	Operational Concepts	Conceptual	Process	yes
8	7	OV-2	Operational	Operational Node Connectivity Diagram	Functional	Structural	yes
9	8	OV-3	Operational	Information Exchange Matrix	Functional	Process	yes
10	9	OV-4 (Typical)	Operational	Organizational Relationships	Conceptual	Structural	yes
11	10	OV-4 (Actual)	Operational	Organizational Relationships	Functional	Structural	no
12	11	OV-5	Operational	Operational Activity Model	Functional	Process	yes
13	12	OV-6a	Operational	Operational Rules Diagram	Functional	Process	yes
14	13	OV-6b	Operational	Operational State Transition Description	Declarative	Process	yes
15	14	OV-6c	Operational	Operational Event Trace Diagram	Functional	Process	yes
16	15	OV-7	Operational	Logical Data Model	Conceptual	Process	yes
17	16	SOV-1	Service	Service Hierarchy	Functional	System Function	no
18	17	SOV-2	Service	Service Interface Specification	Functional	System Function	no
19	18	SOV-3	Service	Capability to Service Mapping	Functional	System Function	no
20	19	SOV-4a	Service	Service Constraints	Functional	System Function	no
21	20	SOV-4B	Service	Service State Model	Functional	System Function	no
22	21	SOV-5	Service	Service Functionality View	Functional	System Function	no
23	22	StV-1	Strategic	Strategic Vision	Functional	System Function	no
24	23	StV-2	Strategic	Capabilities Hierarchy	Functional	Capability	no
25	24	StV-3	Strategic	Capabilities Planning Timeline	Functional	Capability	no
26	25	StV-4	Strategic	Capabilities Dependencies	Functional	Capability	no
27	26	StV-5	Strategic	Capabilities to Organizational Mapping	Functional	Capability	no
28	27	StV-6	Strategic	Capability to Operational Mapping	Functional	Capability	no
29	28	SV-1	System	System to System Node Connectivity Diagram	Functional	Structural	no
30	29	SV-2	System	Systems Communications Description	Conceptual	System Function	no
31	30	SV-3	System	Resource Interaction Matrix	Functional	System Function	no
32	31	SV-4	System	Functionality Description (Data Flow Diagram)	Functional	System Function	no
33	32	SV-5a	System	Operational Activity to Systems Function Matrix	Functional	System Function	no
34	33	SV-5b	System	Operational Activity to Systems Services Matrix	Functional	System Function	no
35	34	SV-6	System	System Data Exchange Matrix	Functional	System Function	no
36	35	SV-7	System	Resource Performance Parameters Matrix	Functional	System Function	no
37	36	SV-8	System	Capability Configuration Change	Conceptual	Capability	no
38	37	SV-9	System	Technology & Skills Forecast	Conceptual	Planning	no
39	38	SV-10a	System	System Rules Model	Functional	System Function	no
40	39	SV-10b	System	Systems and Services State Transition Description	Declarative	System Function	no
41	40	SV-10c	System	Systems and Services Event-Trace Description	Functional	System Function	no
42	41	SV-11	System	Physical Data Model	Declarative	System Function	no
43	42	SV-12	System	Service Provision View	Functional	System Function	no
44	43	TV-1	Technical	Technical Standards Profile	Conceptual	Technical	no
45	44	TV-2	Technical	Technical Standards Forecast	Conceptual	Technical	no
46	45	TV-3	Technical	Standards Policy	Conceptual	Technical	no
47	46						Legend
48	47						Target Model Set
49	48						Excluded Models

Table 12 - UPDM Target Set

UPDM Baseline Process Views		
Classification	Model	Name
Node	OV-2	Operational Node Connectivity
	OV-4	Operational Relationships Diagram
Product	OV-7	Logical Data Model
Process	OV-3	Information Exchange Matrix
	OV-5	Activity Diagram
	OV-6a	Operational Rules Diagram
State	OV-6b	Operational State Transition
Timing	OV-6c	Operational Event Trace Diagram

#### 4.2.4 Modeling Languages

Table 13 shows four prominent modeling languages aligned to the target UPDM views; this alignment indicates similar characteristics. The Modeling Languages are Structured (IDEF), UML, SysML and BPMN. Process Models from the four Modeling Languages have been coded for analysis as part of this study. The motivation for this choice is that these process modeling Languages are widely used in the literature and, based on the experience of the author, are broadly used in practice.

An earlier peer reviewed publication (Shuman, 2010) described the alignment of these modeling languages to DODAF views. Table 13 shows models from DODAF and the four Modeling Languages categorized according to the where, how, who (passive) when and categories. This means that these model types predominately address the interrogative in question; for example, the UML Activity Diagram is a process model that is predominately oriented towards addressing process or behavior. For this reason it is aligned to the How interrogative group. This alignment to the how interrogative type is not to suggest that there are not elements of other interrogative types within this model, as will be demonstrated later as the description of data collection proceeds.

Table 13 - UPDM Views and Modeling Language Alignment

Interrogative	Color Classification	UPDM Baseline Process Views		Structured Modeling	UML	SysML	BPMN
		Model Number	Name				
Where	Node	OV-2	Operational Node Connectivity Diagram		Class Composite Structure	Block	
		OV-4	Operational Relationships Diagram		Class Composite Structure	Block	
How	Process	OV-3	Information Exchange Matrix	IDEF 0	Activity Diagram	Activity Diagram	Process
		OV-5	Activity Diagram	IDEF 0	Activity Diagram	Activity Diagram	Process
		OV-6a	Operational Rules Diagram	IDEF 0	Activity Diagram	Activity Diagram	Process
Who (passive)	State	OV-6b	Operational State Transition Description		State Machine		
When	Timing	OV-6c	Operational Event Trace Diagram		Sequence Diagram Timing Communications Diagram		
What	Product	OV-7	Logical Data Model	IDEF 1X	Class Composite Structure	Class Block	

### 4.3 Code Organization

A way was needed to organize code elements in term of categories, composition and generalization associations. MAXQDA supports this kind of information management scheme. The top information categories were set up in accordance with the vertex components of the Executable Architecture Concept Triangle. Architecture Elements, Modeling Language Descriptions, and Executable Architecture Formalisms. Within each category, composition relationships were established for the sub-categories, i.e., models types, models and Modeling and Simulation Formalisms. Within each model category composition and generalization relationships were established. This information organization construct provided a way to bin the elements. Figure 20 shows the 1<sup>st</sup> tier information layers (in MAXQDA) and their relationships to the Executable Architecture Concept Triangle (Figure 10), which served as a framing guide.

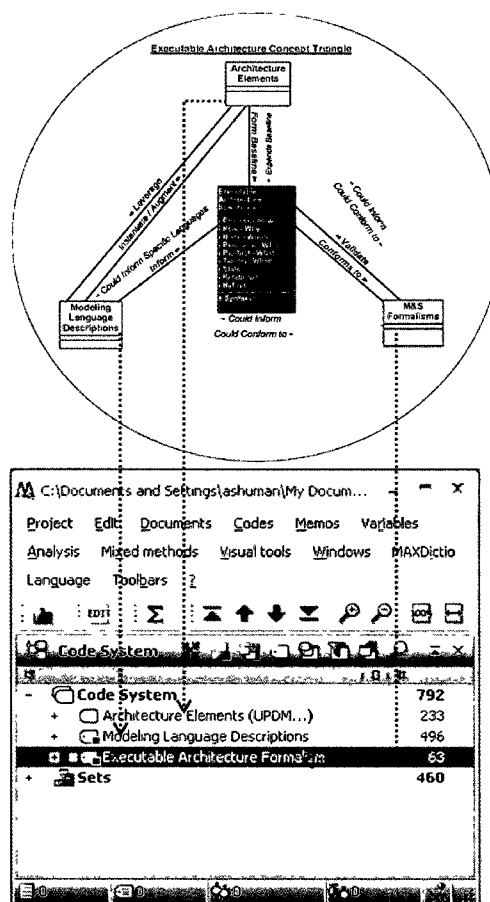


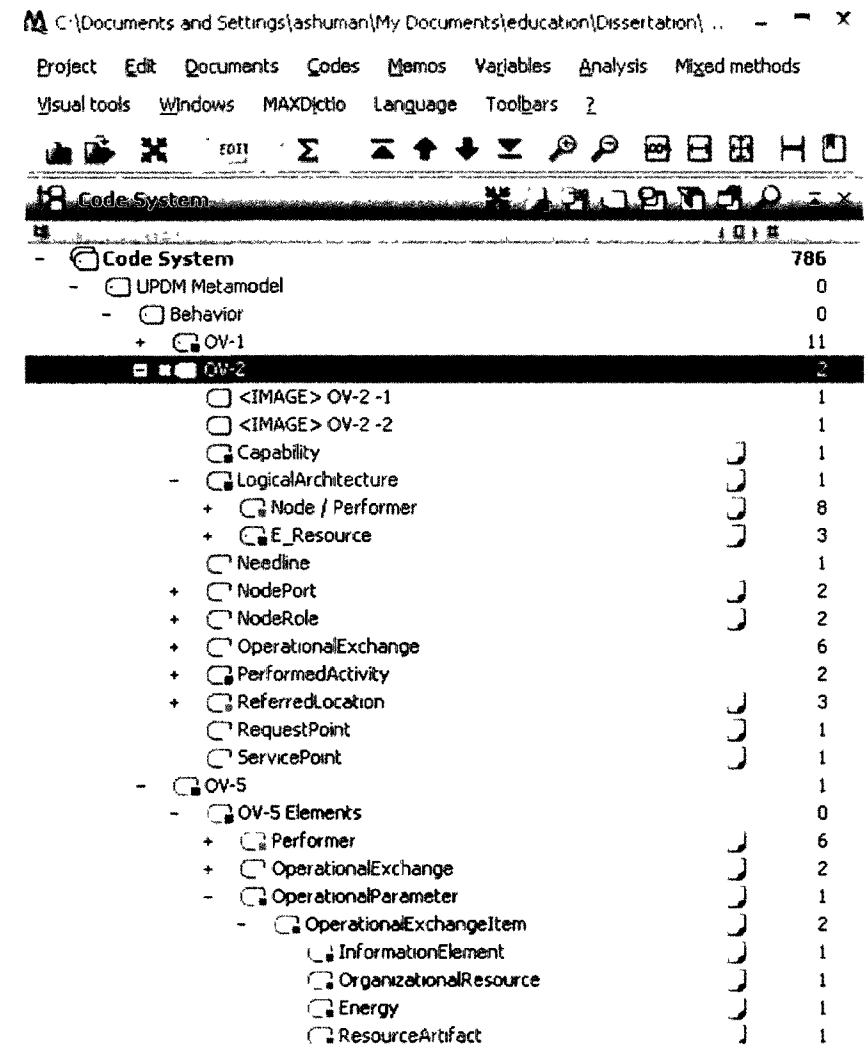
Figure 20 - Top Level of Code Organization in MAXQDA

The complete organizational structure was set up as follows, from 1<sup>st</sup> tier through 4<sup>th</sup> tier:

- a) **Architecture Elements (1<sup>st</sup> tier)**
  - a. **Architecture Framework (UPDM) (2<sup>nd</sup> tier)**
    - i. **Behavior category (3<sup>rd</sup> tier):**
      - 1. **Models (4<sup>th</sup> tier):** OV-1, OV-2, OV-5, OV-6a, OV-6b, OV-6c, SV-1, SV-4, SV-10a, SV-10b, SV-10c
    - ii. **Structure category (3<sup>rd</sup> tier)**
      - 1. **Models (4<sup>th</sup> tier)** OV-4, OV-7
    - iii. **Tables category (3<sup>rd</sup> tier)**
      - 1. **Tables (4<sup>th</sup> tier):** OV-3, SV-6, SV-7
  - b) **Modeling Language Descriptions (1<sup>st</sup> tier)**
    - a. **Structured Language (2<sup>nd</sup> tier)**
      - i. **Behavior category (3<sup>rd</sup> tier)**
        - 1. **Models (4<sup>th</sup> tier):** IDEF 0, DFD
      - ii. **Structure category (3<sup>rd</sup> tier)**
        - 1. **Models (4<sup>th</sup> tier):** IDEF 1X
    - b. **UML (2<sup>nd</sup> tier)**
      - i. **Behavior category (3<sup>rd</sup> tier)**
        - 1. **Models (4<sup>th</sup> tier):** Activity, Common Behaviors, Communications, Interaction, Sequence, State, Timing, Use Case
      - ii. **Structure category (3<sup>rd</sup> tier)**
        - 1. **Models (4<sup>th</sup> tier):** Component, Composite Structure, Package, Object, Class
    - c. **SysML (2<sup>nd</sup> tier)**
      - i. **Behavior category (3<sup>rd</sup> tier)**
        - 1. **Models (4<sup>th</sup> tier):** SysML Activity
      - ii. **Structure category (3<sup>rd</sup> tier)**
        - 1. **Models (4<sup>th</sup> tier):** Block Definition, Internal Block, Parametric
    - d. **BPMN (2<sup>nd</sup> tier)**
      - i. **Behavior category (3<sup>rd</sup> tier)**
        - 1. **Models (4<sup>th</sup> tier)** Process, Choreography, Collaboration, Conversation
  - c) **M&S Formalisms (1<sup>st</sup> tier)**
    - a. **CP-net (2<sup>nd</sup> tier)**
    - b. **DEVS (3<sup>rd</sup> tier)**

The behavior and structure categories shown above support the same pattern of model organization used in UML (OMG, 2009), with a division between structure and behavior models. In addition to this hierarchical organizational, structure code attributes as described in section 4.3 were applied to the code elements. Figure 21 provides a

snapshot from MAXQDA that shows all four layers: (1) component, (2) Architecture Framework or Modeling Language, (3) Type (behavior or structure), and (4) Model Designation.



Code Category	Count
Code System	786
UPDM Metamodel	0
Behavior	0
+ OV-1	11
OV-2	2
<IMAGE> OV-2 -1	1
<IMAGE> OV-2 -2	1
Capability	1
- LogicalArchitecture	1
+ Node / Performer	8
+ E_Resource	3
Needline	1
+ NodePort	2
+ NodeRole	2
+ OperationalExchange	6
+ PerformedActivity	2
+ ReferredLocation	3
RequestPoint	1
ServicePoint	1
- OV-5	1
- OV-5 Elements	0
+ Performer	6
+ OperationalExchange	2
- OperationalParameter	1
- OperationalExchangeItem	2
InformationElement	1
OrganizationalResource	1
Energy	1
ResourceArtifact	1

Figure 21 - Code Categories

Next, the elements were arranged according to composition (i.e., “has-a”) and generalization (i.e., “is-a”) to support model association and ontological categorization, respectively. Both kinds of relationships were important in elemental analysis. Figure 22 shows a snapshot of the Code Window in MAXQDA, with elements for the UPDM OV-2 organized into Composition and Generalization Relationships. Figure 22 is annotated to show those distinctions.

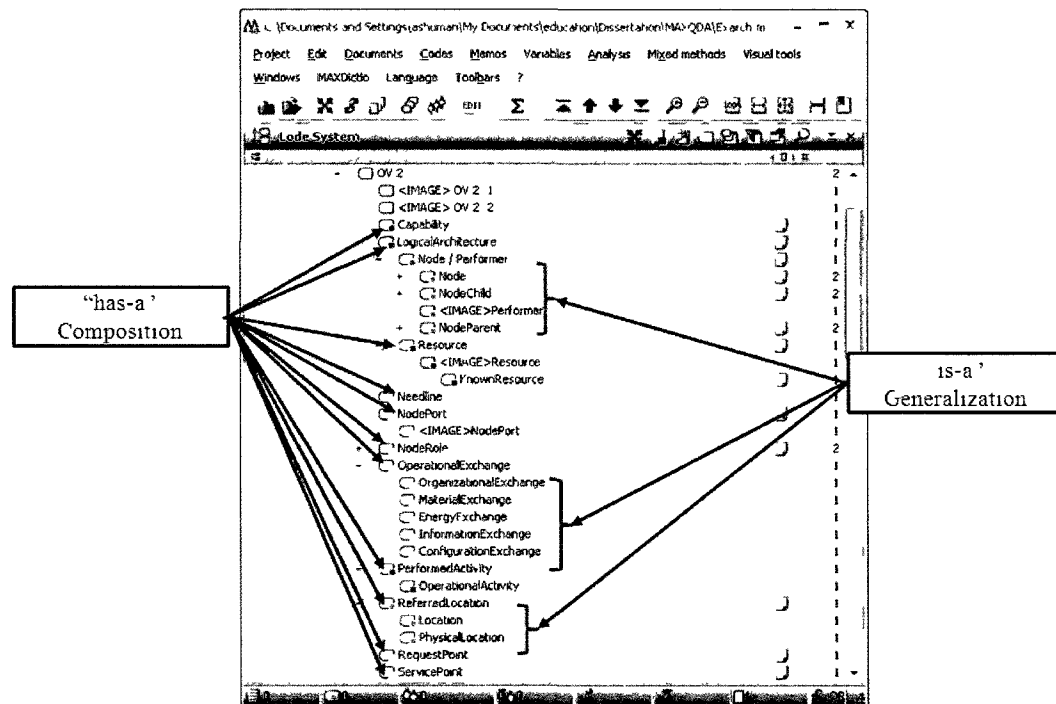


Figure 22 - Elemental Composition and Generalization Relationships (in MAXQDA Code System)

#### 4.3.1 Population of Individual Data Structures (Ontologies and Compositions)

Table 14 contains OV-2 elements pulled from MAXQDA. MAXQDA interacts with MS Excel to support easy export and import of data. The “Code” column contains the names of the codes. The “Interrogatives +” column contains the color and associated interrogative category classifications of each OV-2 element. The “Model Origin” column lists the model source. The “Parent Code” Category contains the hierarchical

organizational code structure in MAXQDA and reflects the aggregations and generalization relationships. For example, the Element “InformationExchange” is in a generalization relationship to the parent element “OperationalExchange”, and “Needline” is in an aggregation relationship to the OV-2 model element. The Model Group column is for classifying each element as Behavior or Structure, and the last column is Ops or Sys representing an Operational or Systems Classification. These codes were used for code grouping, querying and set building (i.e., generation of a group of elements based on specified attribute sets). Each element was color coded to visually reflect an interrogative category consistent with Table 10.

Element color coding was based on interpretation of element definitions as defined in the source documentation. As analysis progressed, elemental color coding was refined to reflect generalization changes. This analysis usually came about in the context of visual inspection of the code through MAXMAPS.

**Table 14 - Sample Coding of OV-2 Elementals**

Code	Interrogative	Model Origin	Parent code	Model Group	Ops or Sys
Capability		OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2	Behavior	Ops
LogicalArchitecture		OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2	Behavior	Ops
Needline	Relationship yellow	OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2	Behavior	Ops
NodePort	Relationship yellow	OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2	Behavior	Ops
NodeRole	Relationship yellow	OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2	Behavior	Ops
OperationalExchange		OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2	Behavior	Ops
OrganizationalExchange	Relationship yellow	OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2\OperationalExchange	Behavior	Ops
MaterialExchange	Relationship yellow	OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2\OperationalExchange	Behavior	Ops
EnergyExchange	Relationship yellow	OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2\OperationalExchange	Behavior	Ops
InformationExchange	Relationship yellow	OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2\OperationalExchange	Behavior	Ops
ConfigurationExchange	Relationship yellow	OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2\OperationalExchange	Behavior	Ops
PerformedActivity		OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2	Behavior	Ops
ReferredLocation		OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2	Behavior	Ops
RequestPoint	Relationship yellow	OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2	Behavior	Ops
ServicePoint	Relationship yellow	OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2	Behavior	Ops
Node Performer		OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2\LogicalArchitecture	Behavior	Ops
Resource		OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2\LogicalArchitecture	Behavior	Ops
Node		OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2\LogicalArchitecture\Node Performer	Behavior	Ops
NodeChild		OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2\LogicalArchitecture\Node Performer	Behavior	Ops
NodeParent		OV 2	Architecture Elements (UPDM )\UPDM Architecture Framework\Behavior\OV 2\LogicalArchitecture\Node Performer	Behavior	Ops

Figure 23 is a sample MAXMAPS OV-2 visual model. As coding progressed, some additional categories were added because there were elements that did not fit well into the original six interrogatives. These additional categories included: relationships (yellow), hybrids (purple), and a category for Who / What / How (Passive), for state (explained in section 4.2.1).

In Table 14, Logical Architecture is classified as Hybrid (Purple) because it has children elements (generalization relationships) that fall into more than one main category: having a node child classified as Where (Aqua) and a Resource child classified as Who (Red). This method, based on ontologies and attribute coding reveals an ambiguity that reflects the source UPDM meta-model relationships, and may be a case where the source meta-model is incorrect or questionable.

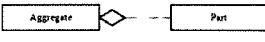


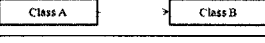

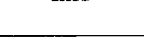





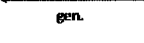
Figure 23 shows an OV-2 drawn in MAXQDA MAX MAPS. MAXMAPS supports the depiction of objects, links and annotations. Each Object is linked or synchronized with a code in the MAXQDA database. Each code in the database was defined based on authoritative source material definitions, using the code memo feature. By touching the object on the map, the definition from the associated memo is displayed on the map. This was useful in sorting out relationships.

Links show relationships between objects. The links in Figure 23 show aggregation and generalization relationships annotated as “has” and “gen.” on the relationship lines. This was the starting point for all elemental depictions. Other relationships such as association relationships were added to complete the model. As an example of the “has-a” and “is-a” relationship depiction, it may be seen in Figure 23 that Resource “is a” Performer (generalization), while the OV-2 “has a” Performer element (aggregation).

**Finding:** This method, based on ontologies and attribute coding reveals an ambiguity in UPDM that reflects the source meta-model relationships, and may be a case where the source meta-model is incorrect or questionable.



Table 15 - UML Relationships and MAXMAPS Links Equivalences

Name	Definition	UML Depiction	MAXQDA MAXMAP Depiction
Aggregation	An aggregation relationship depicts a classifier as a part of or as subordinate to, another classifier	 Aggregation	
Association	An association is a structural relationship that describes a set of links a link is a connection among objects	 Association  Association	 
Composition	A composition relationship represents a whole-part relationship and is a type of aggregation	 Composition	
Dependency	A dependency is a semantic relationship between two things in which a change to one thing (the independent thing) may affect the semantics of the other thing (the dependent thing)	 Dependency	
Generalization	A generalization relationship indicates that a specialized (child) model element is based on a general (parent) model element. Although the parent model element can have one or more children, and any child model element can have one or more parents typically a single parent has multiple children	 Generalization Specialization	

#### 4.3.2.1 Data Element Analysis Roadmap

Because of the number and variety of models and associated elements, it became apparent not too far into the coding that a roadmap would be required to help guide the data analysis. This kind of method evolution is typical of grounded theory investigations (Corbin & Strauss, 2008).

#### 4.3.2.2 Roadmap

With over 750 data elements in the database, a way was needed to organize the data for analysis. A roadmap (Figure 24) based on the EACT was constructed to provide a way to address this complexity. The purpose of the roadmap is to provide element by element comparison for elimination of duplicates and redundancies, in order to build a composite or merged meta-model. It aids in model identification, and it provides a framework for comparative analysis, i.e., model alignment and grouping (based on Process, State, Timing and Node).

#### Data Element Analysis Roadmap Steps 1-10:

1. Develop six UPDM meta-models (e.g., OV-2, OV-4, OV-5, etc.).
2. Build four Group UPDM meta-models, based on four types: Process, Timing, State, and Node.

3. Build Composite UPDM meta-model (by merging four group meta-models above into one).
4. Build eleven Modeling Lang. meta-models (e.g., SysML Activity Diagram, BPMN, etc.).
5. Build four Group UPDM-Language meta-models), by aligning Modeling Language meta-models in step 4 to groups meta-models built in step two.
6. Build Composite UPDM-Language m-m (the foundational EAS) by merging the four group m-m from step five.
7. Compare Composite UPDM from step three & EAS from step six.
8. Code & build m-m for M&S Formalisms.
9. Compare M&S Formalism meta-models to EAS.
10. Build EAS Ontologies; conduct element analysis and refine EAS. This step will be described in detail later.

Table 16 lists the 10 roadmap steps depicted in Figure 24, the object of each model step, and coding types. The coding principles were described in Chapter 3. To recall, the coding principles are as follows:

- a) **Open Coding.** Develop Categories or Themes (Categories, Properties, and Attributes).
- b) **Axial Coding.** Place data into categories or themes (**Binning**).
- c) **Selective Coding.** Observe relationships revealed and how they combine to form a story line to describe phenomenon (Corbin & Strauss, 2008).

**Table 16 - Data Element Analysis Roadmap Steps**

Roadmap Step	Roadmap Step Description	Object	Coding and Theory Building
1	Build 6+ UPDM m-m (e g , OV-2, OV-4, OV-5, etc )	UPDM Views	Axial Coding
2	Build 4 Group UPDM m-m	UPDM Groups	Axial / Selective Coding
3	Build Composite UPDM m-m (composed of 4 groups)	UPDM	Axial / Selective Coding
4	Build 11 Modeling Lang m-m (e g , SysML Activity Diagram, BPMN, etc )	Modeling Language Models	Axial / Selective Coding
5	Build 4 Group UPDM-Lang m-m (merge 3&4)	UPDM - Language	Axial / Selective Coding
6	Build Composite UPDM-Language m-m (the EAS), composed of 4 group m-m step 5	UPDM - Language	Selective Coding / Theory Building
7	Compare Composite UPDM (step 3) & EAS	UPDM & UPDM - Language	Selective Coding
8	Code & build m-m for M&S Formalisms	CP-net, DEVS	Axial Coding
9	Comp Formalisms m-m to EAS	CP-net, DEVS & UPDM-Langu	Selective Coding
10	Build EAS Ontologies (element analysis)	UPDM-Language	Theory Building

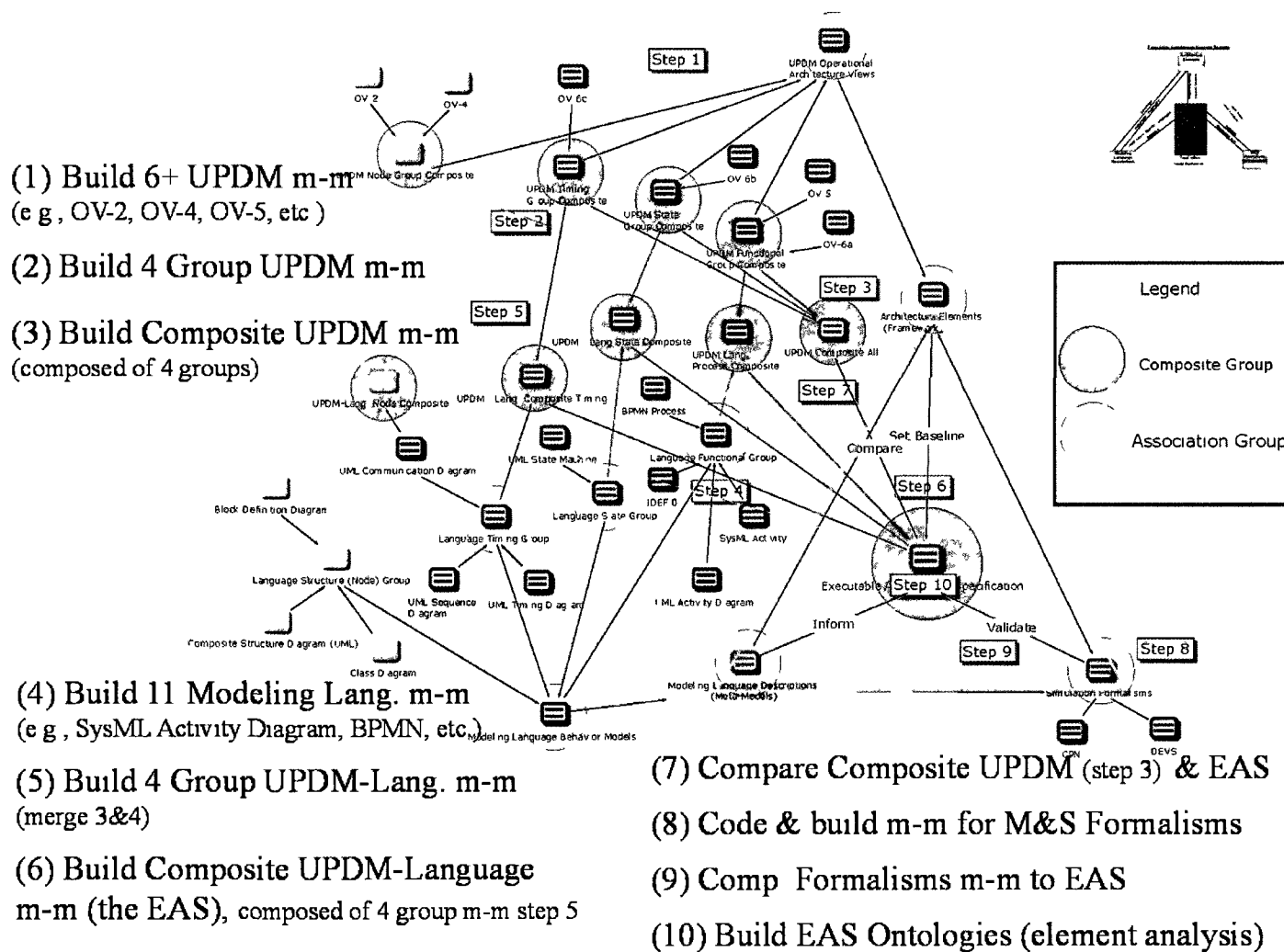


Figure 24 - Data Element Analysis Roadmap: across Similar Meta-model (m-m) Types

#### 4.4 Data Element Analysis Roadmap Execution

This section provides a detailed explanation of the ten steps of the Data Element Analysis Roadmap.

##### 4.4.1 Step 1: Code, Classify & Build UPDM Meta-models

Figure 25 shows the Executable Architecture Concept Triangle (EACT) with the Architecture Elements component highlighted at the top, included here as a guidepost to which Steps 1, 2 and 3 of the roadmap align. In other words, this section will focus on the components within EACT that relate to Architecture Elements. To this end, Modeling language model elementals were coded, categorized and aligned to UPDM model groups. Again, the coloring scheme shown in Table 10 reflects interrogative categories discussed in the following sections.

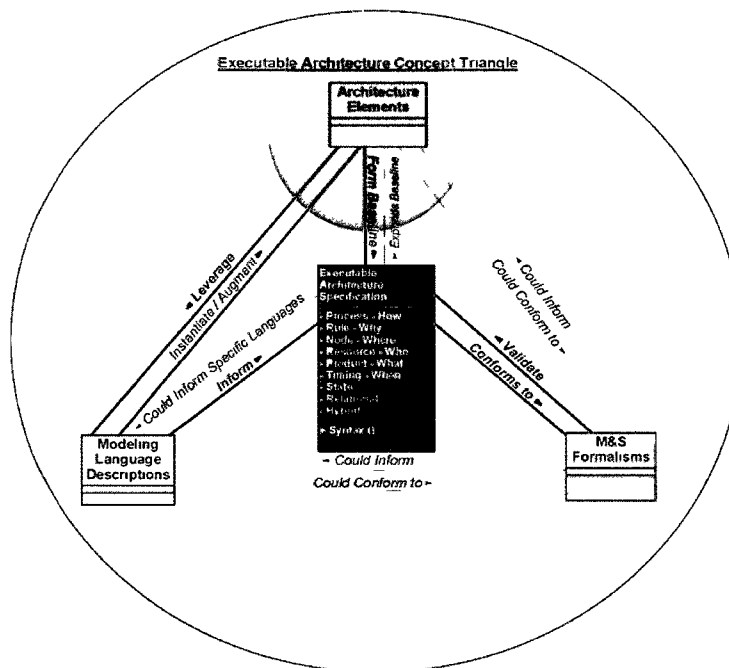
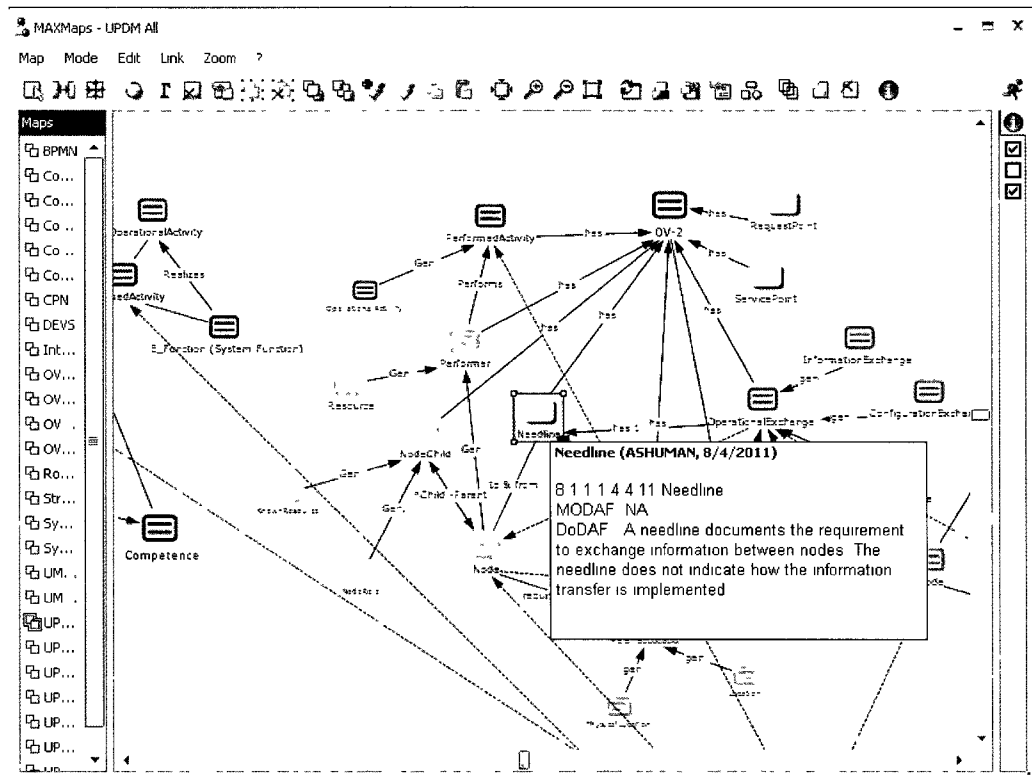


Figure 25 - Executable Architecture Triangle (Architecture Elements Guidepost)

Step 1 of the Roadmap is the development of Architecture meta-models for the targets set from UPDM, and it begins with Open Coding. Although in most cases, Open

### Figure 26 - Meta-Models

Figure 27 shows an example. Each visual model object (element) was classified and color coded to reflect the interrogatives categories discussed in paragraph 4.6. The purple lines are suggestive of cross-model common elements. This will be addressed at length later.



**Figure 27 - MAXMAPS with Mouse-Over Memo Display (Needline Definition)**

#### **4.4.2 Step 2: Build Group UPDM Meta-model Maps & Adjust coding**

Roadmap Step 2 is the specification of four groups based on the interrogatives and the development of group composite UPDM meta-models aligned to those groups. The alignment of target UPDM models to the groups is shown in Table 17; this same alignment may be seen in the Roadmap. This approach provides a manageable way to break the problem down into workable pieces. Models were grouped together according to four interrogative focus areas: How, State, When, and Where. It is evident that all

models contain elements of more than one type (as may be seen from the many colored objects in Figure 26), but they can be classified according to principle interrogative focus area. For example, the UPDM OV-5 was placed into the behavior group.

**Table 17 - Composite Groups**

<b>Composite Group</b>	<b>UPDM Models</b>	<b>Color</b>
<b>Behavior</b>	OV-5, OV-6a	Green
<b>State</b>	OV-6b	Brown
<b>Timing</b>	OV-6c	Blue
<b>Node</b>	OV-2, OV-4	Aqua

The research shows that models that are of the same interrogative type can be compared in order to produce composite models. Nothing is lost by over-generalization because each composite that is produced for the model group is compared against all other groups in the further refining step 3.

Figure 28 is the UPDM function group composite (OV-5 and OV-6a). This is a simple composite model that fuses the element “OperationalConstraint” and the element “SubjectofOperationalConstraint” into the Operational Activity Model (OV-5). This UPDM functional group composite is the target for the next two composite fusions: the first to the other UPDM groups, and the second to the modeling language composite.

Building composite group meta-models was an intermediate step in building a foundational model set around which other UPDM elemental additions and language model elements were added. Figure 28 shows the UPDM group composite for the UPDM function group, with the source OV-6a and OV-5 in the top left and right corners respectively. It is a very straightforward grouping in that it simply shows the “OperationalConstraint”, “SubjectofOperationalConstraint” and Mission elements from the OV-6a added to the OV-5. This addition, in turn, requires the addition of generalization lines linking SubjectofOperationalConstraint to Node, “OperationalActivity”, “OperationalExchangeItem”, OperationalActivity and “PerformedActivity” (a suggestion which is not part of the original meta-model).

#### 4.4.3 Step 3: Build Common UPDM Meta-model map & Adjust coding

Roadmap Step 3 is the building of a UPDM composite meta-model. Figure 29 is a progression that is based on the previous composite group functional UPDM model. It is considerably more complex because it combines all elements from the original seven UPDM operational meta-models into one model. Key parent nodes have been annotated with yellow circles to highlight them as central parent nodes. The observer can easily see that while all interrogative categories are present in the Composite UPDM Behavior meta-model, the time attribute is remarkably lacking because the only explicit time element that is seen in the composite UPDM model is the sequence element. The element “ActualMeasurementSet”, categorized as hybrid or purple, is associated in the parent OV-3 meta-model with the “OperationalExchange” element. The “ActualMeasurementSet” does contain “Measures” that have, among other attributes, two

time attributes: periodicity and timeliness. However, this is not a very robust set of time related attributes or elements. This is not particularly surprising given that DODAF and MODAF were not designed as *simulation* modeling frameworks. Any simulation modeling tool is necessarily going to have to address timing considerations much more explicitly and broadly.

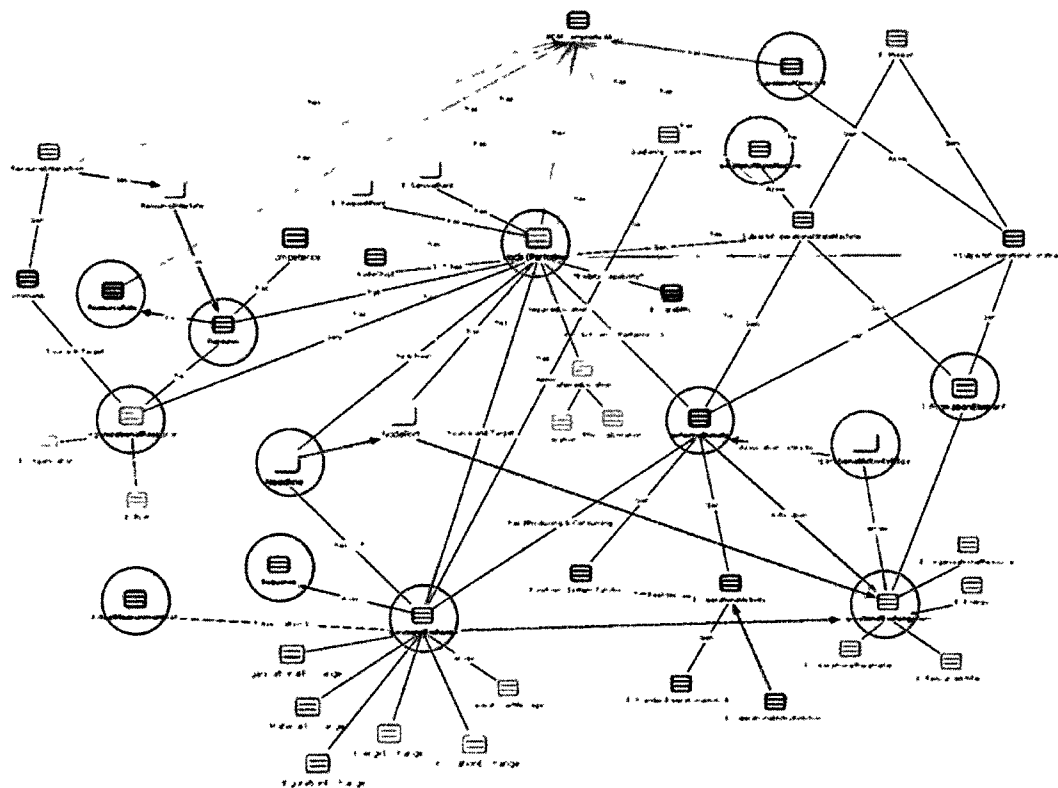


Figure 29 - UPDM Composite OV-5 & OV-6a & OV-6b & OV-2 & OV-6c & OV-4

**Finding:** It is of note that all interrogative categories are present in the Composite UPDM Behavior meta-model; however, the time attribute appears to be remarkably lacking. The only explicit time element that is seen in the composite UPDM model is the sequence element.

#### 4.4.4 Step 4: Code, Classify & Build Language Meta-model Maps

Figure 30 shows the Executable Architecture Triangle with the *Modeling Language Descriptions* component on the left vertex highlighted. It is included here as a guidepost to which Steps 4 and 5 align; in other words, the focus of discussion regarding interaction and relationship within EACT is now shifted to the Modeling Languages. Roadmap Step 4 illustrates how modeling language meta-models are developed for each of the modeling languages associated with the four analysis grouping: Behavior, State, Timing, and Node. Meta-models were developed in step 4 for each of the Models shown in Table 18, which are then aligned to the analysis groups. This alignment is also shown in the Roadmap, Figure 24, where the UPDM Group Composites are color coded as shown in Table 18.

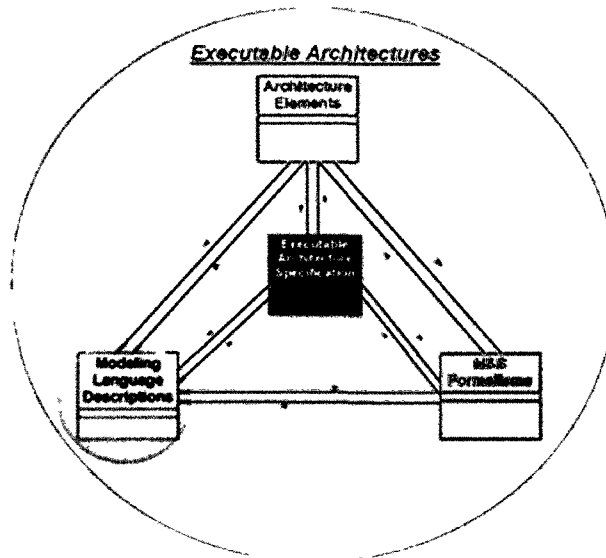


Figure 30 - Executable Architecture Triangle (Modeling Language Guidepost)

**Table 18 - Group - Language Meta-model Alignment**

Group	Model	Color
Behavior	IDEF0, SysML Activity Diagram, UML Activity Diagram, BPMN Process Diagram	Green
State	UML State	Brown
Timing	UML Sequence, UML Communication, UML Timing	Blue
Node	SysML Block, UML Class, UML Composite	Aqua

Figure 31 depicts the meta-model for the SysML Activity Diagram; it is representative of what was done for the other Language models shown in Table 18.

The SysML Activity Diagram is similar to the UML Activity Diagram, except for the additions shown highlighted with aqua circles. As Dori (2002) pointed out, UML, and by extension SysML, are both encumbered with implementation detail. This is a drawback from a purely modeling language point of view. The large gray circles in Figure 31 are examples of implementation detail that does not contribute to conceptual, functional, or declarative modeling (See Table 17 Definition Column). Upon reflection, it becomes apparent that in comparison to the UPDM OV-5, there are a number of elements that are part of the SysML Activity Diagram (Figure 32) that could augment the OV-5.

**Finding:** Comparison of the SysML Activity Diagram to the UPDM OV-5 reveals that there are a number of SysML Activity Diagram elements that could augment the UPDM OV-5 (e.g., time constraints, duration constraints and rate, and probability rules.)

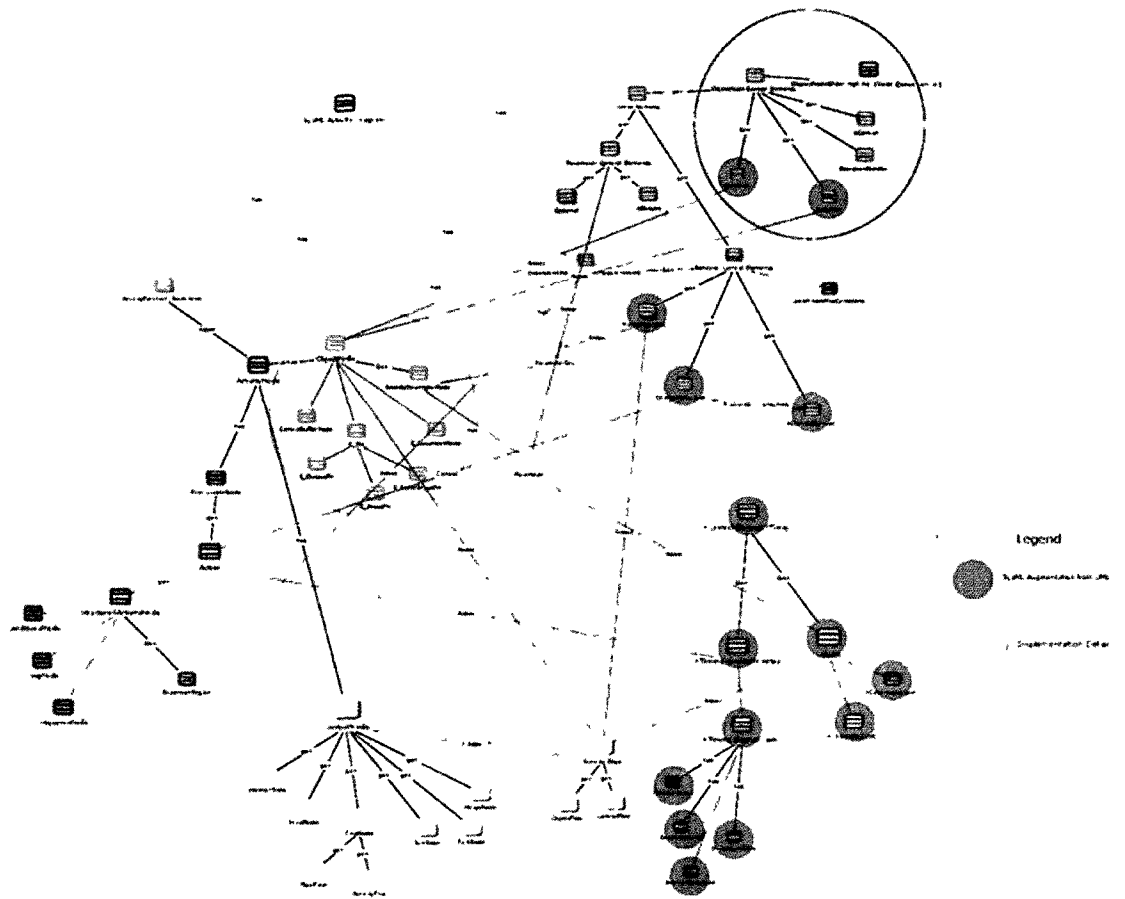


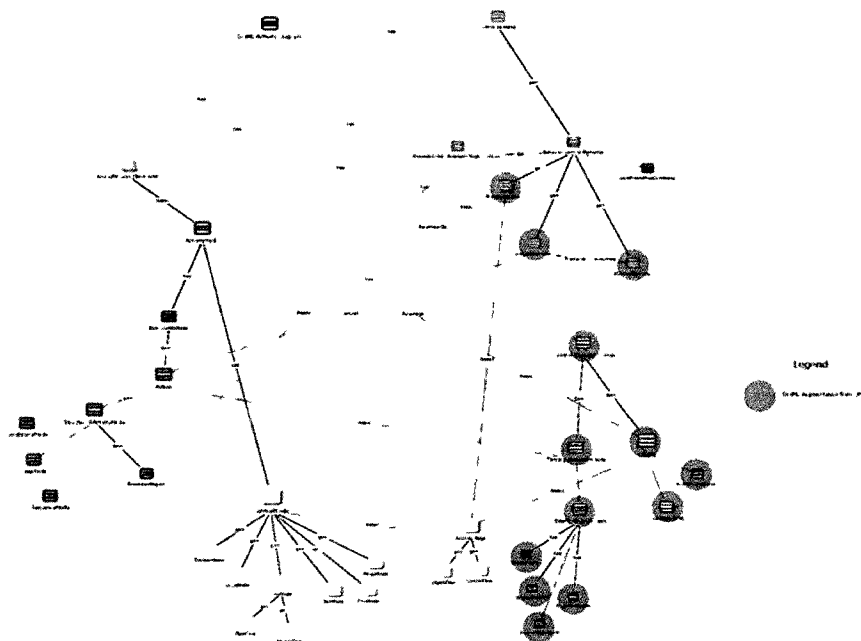
Figure 31 - SysML Activity Diagram

The ability to conduct queries against the data set based on attributes that have been assigned to the data (through Axial Coding) is important because it can help to sort through questions related to the data; for example, Table 19 contains elements from the SysML Activity Diagram that have been marked as having Implementation Detail. Table 19 was used to verify that these elements could reasonably be classified as implementation-level, enabling their exclusion from the process meta-model. Detailed elemental inspection of SysML/UML confirms Dori's (2002) assertion that from a modeling perspective it is unwieldy, or heavily weighted with implementation-level detail, thereby reducing efficiency for purposes of process modeling.

**Finding:** Detailed elemental inspection of SysML/UML shows that from a modeling perspective it is laden with implementation-level detail.

Table 19 - SysML Activity Diagram Implementation Detail Elements

Code	Interrogative	Model	Definition
ObjectNode	Where (location) aqua	SysML Activity	An object node is an abstract activity node that is part of defining object flow in an activity
CentralBufferNode	Where (location) aqua	SysML Activity	A central buffer node is an object node for managing flows from multiple sources and destinations
DataStore	Where (location) aqua	SysML Activity	A data store node is a central buffer node for non-transient information
ExpansionNode	Where (location) aqua	SysML Activity	An expansion node is an object node used to indicate a flow across the boundary of an expansion region
Activity Diagram Parameter Control Elements (Logical)	Why (Rule) pink	SysML Activity	Grouping of Parameter Control Elements
+Optional (Parameter control)	Why (Rule) pink	SysML Activity	When the «optional» stereotype is applied to parameters, the lower multiplicity must be equal to zero. This means the parameter is not required to have a value for the activity or any behavior to begin execution. Otherwise, the lower multiplicity must be greater than zero, which is called "required." The absence of this stereotype indicates a constraint, see below
isStream (Parameter control)	Why (Rule) pink	SysML Activity	Parameters are extended in complete activities to add support for streaming, exceptions, and parameter sets
ObjectNode Control Elements	Why (Rule) pink	SysML Activity	Grouping of ObjectNode Control Elements
+OverWrite (Object Node control)	Why (Rule) pink	SysML Activity	When the «overwrite» stereotype is applied to object nodes, a token arriving at a full object node replaces the ones already there (a full object node has as many tokens as allowed by its upper bound)
ExceptionHandler(Object Node control)	Why (Rule) pink	SysML Activity	An exception handler is an element that specifies a body to execute in case the specified exception occurs during the execution of the protected node
+NoBuffer (ObjectNode control)	Why (Rule) pink	SysML Activity	When the «nobuffer» stereotype is applied to object nodes, tokens arriving at the node are discarded if they are refused by outgoing edges, or refused by actions for object nodes that are input pins



**Figure 32 - SysML Activity (-Implementation)**

SysML is a system engineering extension of UML. That is, the Activity Diagram in SysML contains elemental extensions beyond the Activity Diagram in UML. Table 20 shows the SysML Activity Diagram element augmentations to the UML Activity Diagram. The augmentation elements fall into two interrogative categories: rule (pink) and timing (blue). The timing elements include Time Constraint, Duration Constraint and Rate as key elements, and the following elements from the timing diagram: x, y, z. Timing diagram elements were included because the SysML Activity Diagram has a loosely worded provision for the inclusion of timing diagram constraints, through annotation. The Rule elements deal with the probability of an occurrence and the use of data as control. The timing and rule classified elements are candidate augmentations for a future UPDM (and by extension DODAF, since UPDM is based on DODAF), as well as for an Executable Architecture Specification based on UPDM.

**Finding:** The timing and rule classified elements are candidate augmentations for a future UPDM (and by extension DODAF, since UPDM).

Table 20 - SysML Non-Implementation Detail Element Augmentations (over UML)

Code	Interrogatives	Model Origin	Model Group	Ops or Sys	Augment	Implementation	Definition
	When (Event- timing) blue	SysML Activity	Behavior	Both	1	0	
	When (Event- timing) blue	SysML Activity	Behavior	Both	1	0	When the «rate» stereotype is applied to an activity edge, it specifies the expected value of the number of objects and values that traverse the edge per time interval, that is, the expected value rate at which they leave the source node and arrive at the target node.
	When (Event- timing) blue	SysML Activity	Behavior	Both	1	0	Discrete rate is a special case of rate of flow (see Rate) where the increment of time between items is non-zero. Examples include the production of assemblies in a factory and signals set at periodic time intervals.
	When (Event- timing) blue	SysML Activity	Behavior	Both	1	0	Continuous rate is a special case of rate of flow (see Rate) where the increment of time between items approaches zero.
	When (Event- timing) blue	SysML Activity	Behavior	Both	1	0	The simple time model in UML can be used to represent timing and duration constraints on actions in an activity model. These constraints can be notated as constraint notes in an activity diagram. Although the UML 2 timing diagram was not included in this version of SysML, it can complement SysML behavior diagrams to notate this information.
	When (Event- timing) blue	SysML Activity	Behavior	Both	1	0	Timing Diagram Timing Diagrams are used to show interactions when a primary purpose of the diagram is to reason about time. Timing diagrams focus on conditions changing within and among Lifelines along a linear time axis. Timing diagrams describe behavior of both individual classifiers and interactions of classifiers, focusing attention on time of occurrence of events causing changes in the modeled conditions of the Lifelines.
	When (Event- timing) blue	UML Timing	Behavior	Both	1	0	A DurationConstraint defines a Constraint that refers to a DurationInterval.
	When (Event- timing) blue	UML Timing	Behavior	Both	1	0	A TimeConstraint defines a Constraint that refers to a TimeInterval.
	When (Event- timing) blue	UML Timing	Behavior	Both	1	0	A DestructionEvent models the destruction of an object.
	Why (Rule) pink	SysML Activity	Behavior	Both	1	0	A control operator is a behavior that is intended to represent an arbitrarily complex logical operator that can be used to enable and disable other actions. When the «controlOperator» stereotype is applied to behaviors, the behavior takes control values as inputs or provides them as outputs, that is, it treats control as data.
	Why (Rule) pink	SysML Activity	Behavior	Both	1	0	When the «probability» stereotype is applied to edges coming out of decision nodes and object nodes, it provides an expression for the probability that the edge will be traversed.
	Why (Rule) pink	SysML Activity	Behavior	Both	1	0	A control operator is a behavior that is intended to represent an arbitrarily complex logical operator that can be used to enable and disable other actions. When the «controlOperator» stereotype is applied to behaviors, the behavior takes control values as inputs or provides them as outputs, that is, it treats control as data.

#### 4.4.5 Step 5: Building Group Meta-model Maps

Step 5 is the development of group UPDM-Language Composites. In this step four group composites are constructed as shown in Table 21, and as indicated in Step 5 of the Roadmap, Figure 24.

**Table 21 - UPDM-Language Model Group Composites**

Composite Group	Group Member Models	Roadmap Color
<b>Behavior</b>	OV-5, OV-6a, IDEF0, UML Activity Diagram, SysML Activity Diagram, BPMN Process Model, UML Timing	Green
<b>State</b>	OV-6b, UML State	Brown
<b>Timing</b>	OV-6c, UML Sequence, UML Communications, UML Timing	Blue
<b>Node</b>	OV-2, OV-4, Class, Block, Composite	Aqua

##### 4.4.5.1 Set building with attribute queries

Figure 33 is representative of this step; it is a composite Behavior meta-model that is composed of the group member models shown for the Behavior composite group, shown above in Table 21. The elements for this meta-model were produced by running a series of code queries against the code database, which resulted in data sets, each of which was used to compare and analyze elements within that set. Four data sets were created to support the Behavior group composite meta-model development; these are data sets 1-4 shown in Table 22. The code queries were based upon the model source attribute, which had been previously coded for each data element in the database. Data sets 5-7 in Table 22 were used to support the development of the other model composite groups (i.e., state, timing, and node). The result of each query was a data set that was used for the assessment and comparison of elements. The descriptive attributes (Interrogative, Color, Model Origin, Operational or System Element, Model Group, Parent Code) described in Section 4.1 were used to create and populate selective data

sets. Traceability from data set to database to authoritative source is supported in the tool, which was important to the data management of hundreds of objects.

**Table 22 - Code Query Sets**

<b>Data Set</b>	<b>Group</b>	<b>Query Source</b>
1	Behavior	OV5, OV-6a & IDEF0
2	Behavior	OV5, OV-6a & UML Activity Diagram
3	Behavior	OV5, OV-6a & SysML Activity Diagram
4	Behavior	OV5, OV-6a & BPMN
5	State	UML State, OV-6b
6	Timing	UML Sequence, UML Timing, UML Simple Time, OV-6c
7	Node	UML Block, BPMN Process, UML Communications, UML Seq, SysML Act.

It was frequently necessary to look up the element definition in order to trace the element back to the authoritative source, particularly where there was some ambiguity concerning its meaning or its relationships to other elements.

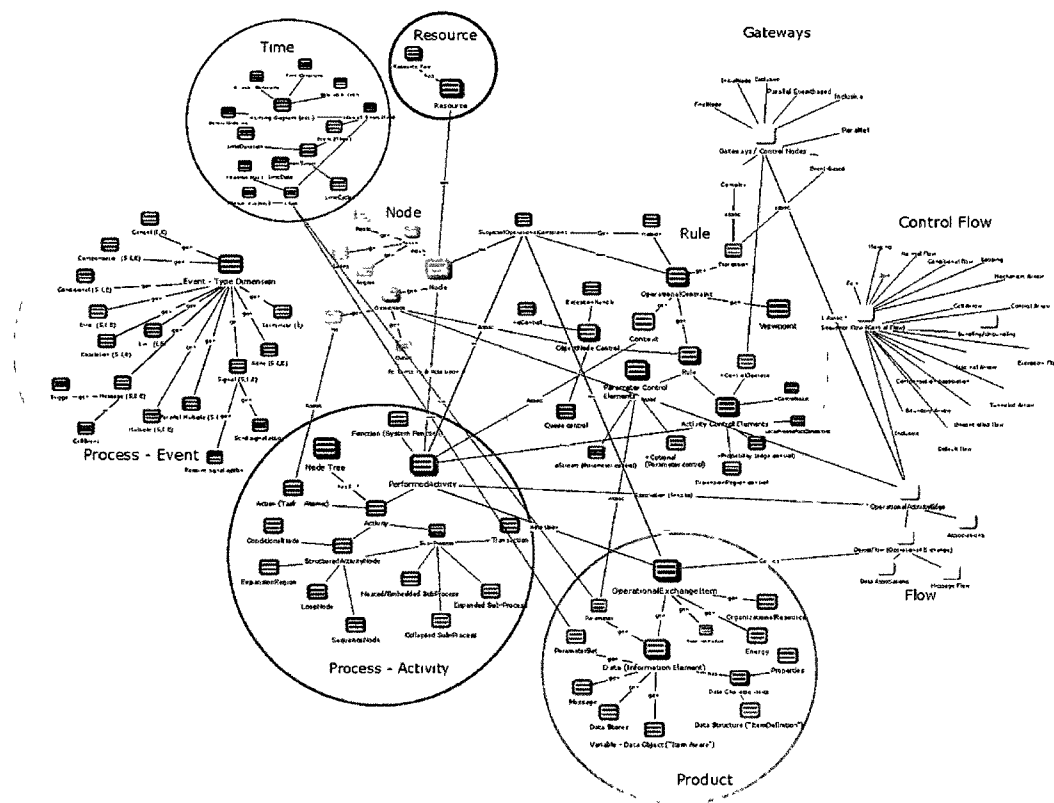


Figure 33 - Composite Functional Group UPDM-Language

#### 4.4.5.2 Element Comparisons

Within each analysis group, Behavior, State, Timing and Node, elements were compared to each other. To do this, each of the analysis groups was developed incrementally by querying for model elements associated with those groups and cross-comparing the findings. The basic principle observed is that similar elements have to be compared to determine whether they are individual element, duplicate, equivalent or an extension. The comparison was inclusive, meaning the bias was for inclusion rather than elimination of elements, such that only duplicative elements were excluded. Elements were classified according to one of four comparison classifications, as shown and defined in Table 23. Table 24 shows the result of a query for BPMN elements; in this table, element organization reflects the ontologies (composition and generalization

relationships) that were created in the database, which resulted in groups (parent elements) that have child or specialization elements associated with them.

The parent-child relationship is useful because it results in element groups that can be compared to similar groups of elements from other model queries. For example, in Table 24 there are a number of elements that fall under the Event element (i.e., Cancel, Compensation, Conditional, etc.); these are child elements of the parent Event element for BPMN. Next, the Event element is cross compared to other Event or Event-like elements in Table 25, for other modeling languages.

**Table 23 - Comparison Classifications**

<b>Comparison Classification</b>	<b>Definition</b>	<b>Included (yes/no)</b>
Individual Element	Unique	yes
Duplicate	Same as another element	no
Equivalent	Similar to another element	no
Extension	Extension of another element	yes

Table 25 and Table 26 show comparison tables for the Event and Activities element groups. Each code was assessed using the comparison classifications listed in Table 23. If a code was a duplicate or the same equivalent class to another, it was not added to the composite meta-model: if it was identical or an individual extension it was retained. For example, Table 26 lists the activity elements from the languages associated with the analysis group: BPMN, UML Activity Diagram, SysML Activity Diagram, IDEF0, OV-5, and OV-6a in the table rows. The columns list the languages, and an x in the intersection of row and column indicates that the element is found in the source model.

Each element was analyzed within a comparison classification. The result was a series of analyzed lists of elements (Process-Event, Process-Activity, Rule, Control Node Flow and Gateway, Time, Product, and Nodes). The comparison tables and meta-models

are included in the dissertation appendix. This step represents Selective Coding, in which relationships are established and redundancies are removed. The “Behavior” group composite meta-model (Figure 33) was then developed in MAXQDA MAXMAPS. Figure 33 includes circles that annotate the comparison categories discussed above. The method was very useful for making comparisons, but the weakness of it is that it is subject to human interpretation. The results achieved from comparisons through tabular methods and visual mapping of elements and was mutually reinforcing from a validation point of view.

Table 24 - BPMN Elements

	Code	Interrogatives	Model Origin
x	Activities	How (Functional) green	Function
x	Task (Atomic)	How (Functional) green	Function
x	Human Interaction	How (Functional) green	Function
x	Sub Process	How (Functional) green	Function
x	Nested/Embedded SubProcess	How (Functional) green	Function
x	Expanded Sub-Process	How (Functional) green	Function
x	Collapsed Sub Process	How (Functional) green	Function
x	Transaction	How (Functional) green	Function
x	Event - Type Dimension	How (Functional) green	Event
x	Cancel (I,E)	How (Functional) green	Event
x	Compensation (S,I,E)	How (Functional) green	Event
x	Conditional (S,I,E)	How (Functional) green	Event
x	Error (S,I,E)	How (Functional) green	Event
x	Escalation (S,I,E)	How (Functional) green	Event
x	Link (I,E)	How (Functional) green	Event
x	Message (S,I,E)	How (Functional) green	Event
x	Multiple (S,I,E)	How (Functional) green	Event
x	None (S,I,E)	How (Functional) green	Event
x	Parallel Multiple (S,I)	How (Functional) green	Event
x	Signal (S,I,E)	How (Functional) green	Event
x	Terminate (E)	How (Functional) green	Event
	Flow Element (Objects)	Hybrid purple	None
x	Gateways	Relationship yellow	Gateway
x	Complex	Relationship yellow	Gateway
x	Event-Based	Relationship yellow	Gateway
x	Exclusive	Relationship yellow	Gateway
x	Inclusive	Relationship yellow	Gateway
x	Parallel	Relationship yellow	Gateway
x	Parallel Eventbased	Relationship yellow	Gateway
x	Connecting Objects	Relationship yellow	Control Node or flow
x	Sequence Flow (Control Flow)	Relationship yellow	Control Node or flow
x	Merging	Relationship yellow	Control Node or flow
x	Looping	Relationship yellow	Control Node or flow
x	Fork	Relationship yellow	Control Node or flow
x	Join	Relationship yellow	Control Node or flow
x	Normal Flow	Relationship yellow	Control Node or flow
x	Conditional flow	Relationship yellow	Control Node or flow
x	Default flow	Relationship yellow	Control Node or flow
x	Exception Flow	Relationship yellow	Control Node or flow
x	Compensation Association	Relationship yellow	Control Node or flow
x	Uncontrolled flow	Relationship yellow	Control Node or flow
x	Data Flow	Relationship yellow	Control Node or flow
x	Message Flow	Relationship yellow	Control Node or flow
x	Data Associations	Relationship yellow	Control Node or flow
x	Associations	Relationship yellow	
x	Message Flow Associations	Relationship yellow	
x	Correlations	Relationship yellow	
x	ParticipantAssociation	Relationship yellow	
x	Data Characteristics	What (Product) orange	Characteristics
x	Data Structure (ItemDefinition)	What (Product) orange	Characteristics
x	DataState	What (Product) orange	Characteristics
x	Data	What (Product) orange	Data
x	Data Objects	What (Product) orange	Data
x	Data Object References	What (Product) orange	Data
x	Data Stores	What (Product) orange	Data
x	Message	What (Product) orange	Data
x	Event Timer	When (Event-timing) blue	Time
x	timeDate	When (Event-timing) blue	Time
x	timeCycle	When (Event-timing) blue	Time
x	timeDuration	When (Event-timing) blue	Time
	Resource	Who (Resource) green	
	Resource Role	Who (Resource) green	
	Performer	Who (Resource) green	
	Participants	Who (Resource) green	
	PartnerRole	Who (Resource) green	
	PartnerEntity	Who (Resource) green	
x	Rule	Why (Rule) pink	None
x	Scopes	Why (Rule) pink	None
x	Expressions	Why (Rule) pink	None
	Properties	What (State) brown	
x	Swimlanes	Where (Node + location) aqua	
x	Pools	Where (Node + location) aqua	
x	Lanes	Where (Node + location) aqua	
	Artifacts	annotation	
	Interaction Node	annotation	
	Participant Multiplicity	annotation	
	Text Annotation	annotation	
	Group	annotation	

Table 25 - Functional Group Elemental Comparisons (Events)

Group: Function (F) Subgroup: Event: ((E))	BPMN	UML Act	SysML Act	IDEF0	OV-5	count	Comparison Classification	Comment
Events (f3)	x	x	x			0	IE	
Cancel (I,E)	x					1	IE	
Compensation (S,I,E)	x					2	IE	
Conditional (S,I,E)	x					3	IE	
Error (S,I,E)	x					4	IE	
Escalation (S,I,E)	x					5	IE	
Link (I,E)	x					6	IE	
Message (S,I,E)	x	x	x			7	IE	
Multiple (S,I,E)	x					8	IE	
None (S,I,E)	x					9	IE	
Parallel Multiple (S,I)	x					10	IE	
Signal (S,I,E)	x	x	x			11	IE	
Terminate (E)	x					12	IE	
Event		x	x				I (event)	
ChangeEvent		x	x				SEC (Conditional)	Implementation level
MessageEvent		x	x				I (Message)	
Trigger		x	x			13	IX (Message)	
CallEvent		x	x			14	IX (Message)	
SignalEvent		x	x				I (Signal)	
Send signal action		x	x			15	IX (Signal)	
Receive signal action		x	x			16	IX (Signal)	
						Elemental Comparative Classification		
						x	Individual Elements (IE)	
						x	Identical (I)	
						x	Same Equivalent Class (SEC)	
						x	Individual Extension (IX)	

Table 26 - Process Group Comparisons (Activities)

Group: Function (F) Subgroup: Performed/Activity ((1))	BPMN	UML Act	SysML Act	IDEF0	OV-5	OV-6a	Count	Comparison Classification	Comment
<b>BPMN Process &amp; Collab</b>									
Activities	x						1	IE	
Task (Atomic)	x						2	IE	
Human Interaction	x							SEC (Activity)	
Sub-Process	x						3	IE	
Nested/Embedded SubProcess	x						4	IE	
Expanded Sub Process	x						5	IE	
Collapsed Sub Process	x						6	IE	
Transaction	x						7	IE	
<b>IDEF 0</b>									
Function				x				I (Activity)	
<b>OV-5</b>									
PerformedActivity					x		8	IE	Parent to OperationalActivity and
OperationalActivityAction					x			SEC (OperationalActivity)	
OperationalActivity					x			I (BPMN Activity)	
StandardOperationalActivity					x			SEC (OperationalActivity)	
<b>OV-6a</b>									
OperationalActivity						x		I (OperationalActivity)	
<b>SysML &amp; UML Activity</b>									
Action		x	x					I (Task)	
StructuredActivityNode		x	x				9	IE	
ConditionalNode		x	x				10	IE	
ExpansionRegion		x	x				11	IE	
LoopNode		x	x				12	IE	
SequenceNode		x	x				13	IE	
<b>OV-4</b>									added here because of UPDM ref to function here
Function		x	x				14	IE	
						Elemental Comparative Classification			
						x	Individual Elements (IE)		
						x	Identical (I)		
						x	Same Equivalent Class (SEC)		
						x	Individual Extension (IX)		

Three other composite meta-models were developed for the state, timing, and node composite categories. The most complex of the four composite groups is the process group. Relationships shown in the meta-model were derived from contributing models. It may be observed that the relationships between the node, the process, the information exchange, and the data exchange (specified in the source OV-5 Activity diagram) are preserved. Similarly, relationships between gateways in BPMN and actions are maintained. This method preserves relationships from component models in addition to building new ones to reflect the new juxtaposition of elements in the composite meta-model. Ontological relationships (composition and generalization relationships) initially came directly from the data structure in MAXQDA, but were expanded to include similar elements from other models. Aggregation relationships come from MAXQDA “has-a” relationships. The result is Table 40 and Table 41, which contains the ontologically organized elements (discussed later in this chapter). Building the group meta-model required the allocation of related children elements to a common parent. An example would be the allocation of control flow from different model sources to a common parent. Association relationships are captured using MAXMAPS and are preserved across model types through manual inspection and traceability from component to group composite model.

In addition to these four main analysis groups, a validity check using data triangulation principles was conducted (i.e., looking at the same data set from different perspectives), whereby three additional queries were run using the interrogative attribute, for What (i.e., product), Why (i.e., rule) and Relational. The result was a set of composite group meta-models that were merged in step 6 of the roadmap, described below.

#### **4.4.6 Step 6: Build Common Meta-model Map & Adjusting codes**

Step 6 is the development of a unified composite UPDM Language Composite (i.e., the Executable Architecture Specification (EAS)). It was created by taking the four group composite meta-models described in step 5, above, and merging them manually. Each of the group meta-models was printed out and manually transferred to a whiteboard, through which cross model elemental relationships became apparent when the models were in juxtaposition because there were elements that were in common. Because the

process group was the most complex, it was used as the model core; the others were arrayed around it. The same thing was done in MAXQDA in order to produce a merged model. After the model was initially merged, other relationships, such as parent child became more apparent through iterative inspection.

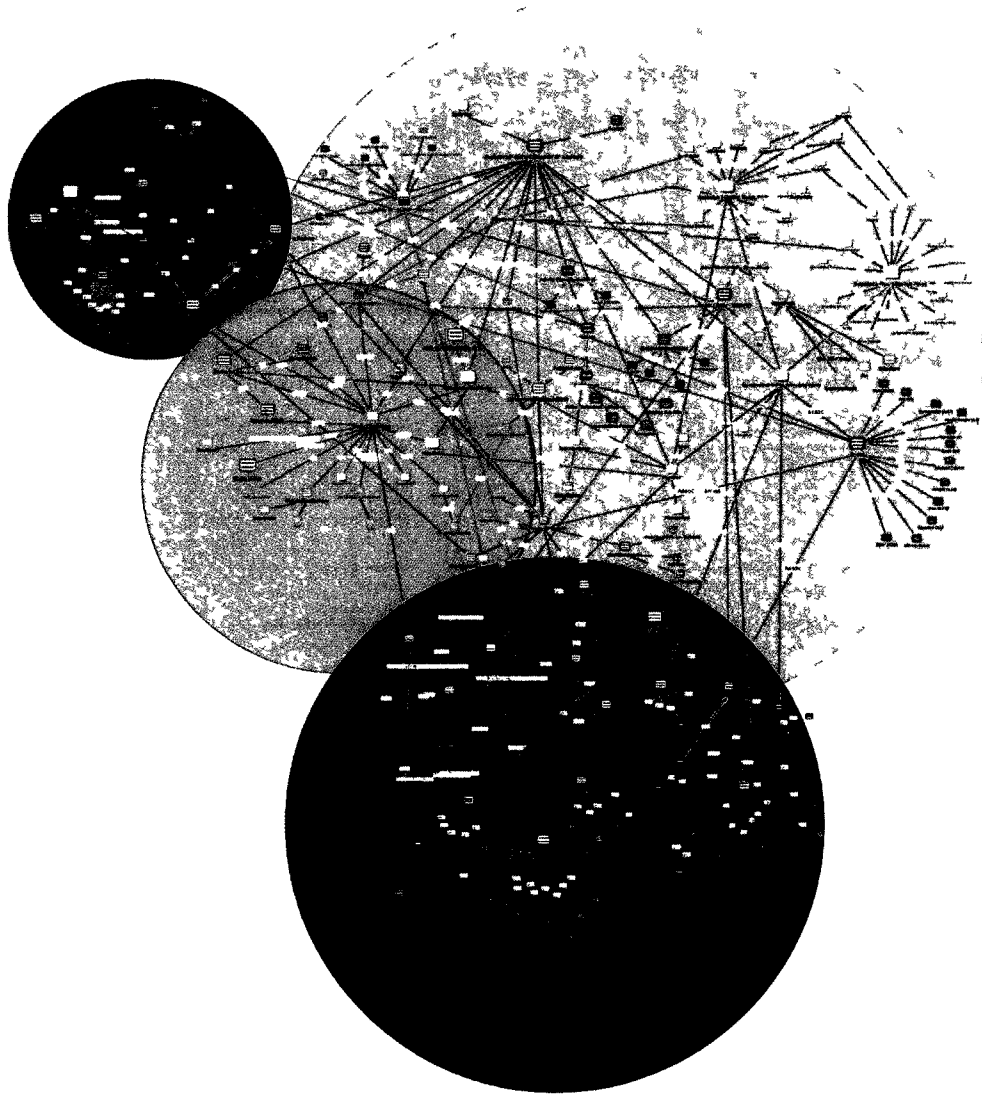
The result is Figure 34, the EAS, showing the four functional groups together. The diagram emphasizes the four functional groups: Process, State, Node and Timing. It combines all models shown in Table 27, under column Member Models, into one composite model.

**Table 27 - Composite UPDM-Language Member Models**

<b>Composite Group</b>	<b>Member Models</b>	<b>Roadmap Color</b>
<b>Behavior</b>	OV-5, OV-6a, IDEF0, UML Activity Diagram, SysML Activity Diagram, BPMN Process Model, UML Timing	Green
<b>State</b>	OV-6b, UML State	Brown
<b>Timing</b>	OV-6c, UML Sequence, UML Communications, UML Timing	Blue
<b>Node</b>	OV-2, OV-4, Class, Block, Composite	Aqua

The constituent groups are highlighted as four large color-coded circles in Figure 34 to show the elements that are associated with the functional groups. The largest functional group is the process group, followed by the timing group. There is overlap between groups, but this is to be expected since some elements are shared between the groups. This is indicative of cross-model integration, which is a desirable trait. For example both the Event Timer and Control Elements (time) belong to both Time and Process functional groups. For this reason, the large color-coded circles are shown overlapping. Figure 34 also shows elements highlighted with small yellow, orange and red circles, for element characterization. The yellow circles indicate generalizations (foundational elements). These elements are higher level generalizations in the data organizational structure, ontologically. The small orange circles represent first tier specializations. They are specializations of the generalizations. The red circles indicate candidate elemental augmentations to the UPDM data set. Table 10 - Color and

Interrogative Classifications shows a synopsis of color coding for Figure 34. While Figure 34 is very complicated; the visual depiction of it can be simplified for analysis purposes because the objects on the map were constructed in layers (supported by the tool), which supports hiding of any unwanted detail, as necessary.



**Figure 34 - EAS Meta-model**

#### 4.4.7 Step 7: Compare Composite Maps (UPDM & Language)

Step 7 is a comparison step, in which the UPDM Composite and the EAS are compared for differences. The purpose of this comparison is to determine candidate element augmentation to an Executable Architecture Specification, based on detailed inspection of the meta-models. By comparing the UPDM composite (Figure 29) to the UPDM-Language composite (Figure 34), it is possible to determine those elements that represent the difference set or the deltas. Figure 35 shows Figure 29 and Figure 34 side by side. The deltas are highlighted with red circles in the right graphic, the UPDM-Language Composite model (Figure 34).

Table 28 and Table 29 provide comparisons between the UPDM composite and the UPDM-Language composite models, where elements are organized by color category. The leftmost column entitled “Element” is the generalization or parent element. The column entitled “Specification” contains subordinate elements. The columns “Composite UPDM” and “Composite All” are marked to show a side by side comparison of elements. *The Elements in the “Composite All” column that are highlighted in yellow are the candidate additions to the Executable Architecture Specification, augmented by adding language.* The comments column has a synopsis of each of the augmented elements.

Table 30 is a synopsis of the candidate element augments, by element generalization, with descriptive comments and category classifications expressed in terms of primary and secondary (where applicable) interrogatives. Table 30 also contains the number of augmentations per element generalization. The majority of the elemental augmentations fall into the functional category. The pink or rule category, which is related to the functional, is second in terms of numbers, with time (blue) third.

A closer look at the kind of elements in Table 30 reveals some interesting features. The Event Element, in row 1, addresses Logical Events stimulation or response. This elemental category was derived from the BPMN process model, and theoretically, it is similar to the concept of token flow control in Colored Petri-Nets. It is, in essence, the token factory, and is an enabler for data flow stimulation, response, and flow control in the context of state transition. The Event object is critical to dynamic process modeling,

because it provides a source of model stimulations, resulting in model subsequent state change and activity response.

The Event Timer, listed in row 3, is similar in that it addresses data flow and flow control from a time control perspective, providing a time-based mechanism for stimulating the model through token generation. Activity Control Elements, in row 9, were derived from SysML. Two features are of note: *random occurrence probabilities* and the *use of data as control*. Random occurrence, i.e., stochastic behavior, is important to dynamic process modeling because process modeling must support more than just deterministic behavior modeling. Real world systems that are being modeled often exhibit non-deterministic behavior, and as such the tools that are brought to bear to mimic or simulate those non-deterministic processes must support these kinds of patterns. The use of data as control is important because it allows for processes to control other processes, through intermediate data that is generated by the process. This enables the processes to generate change in the simulation model, as a result of both deterministic and stochastic triggers in the model. The result is a model that can change and adapt in response to random changes in the internal behavior of the model, or in response to external stimuli.

Control Elements Time, in row 4, addresses the ability to provide detailed, time-based control over the model, which could be as simple as control of a one-time event, in terms of occurrence and duration, or as complex as the control of a schedule of events. In addition, the control features provided in the UML sequence diagram offer iterative control of time-based behavior. Most of the other added elements are related to fine-grained logical and temporal control.

In summation, addition of Logical Events, Time Events, Occurrence Probabilities and fine-grained timing controls to the Executable Architecture Specification will significantly improve the ability of UPDM to support simulation.

**Finding:** Addition of Logical Events, Time Events, Occurrence Probabilities and fine-grained timing controls to the Executable Architecture Specification will significantly improve the ability of UPDM to support simulation.

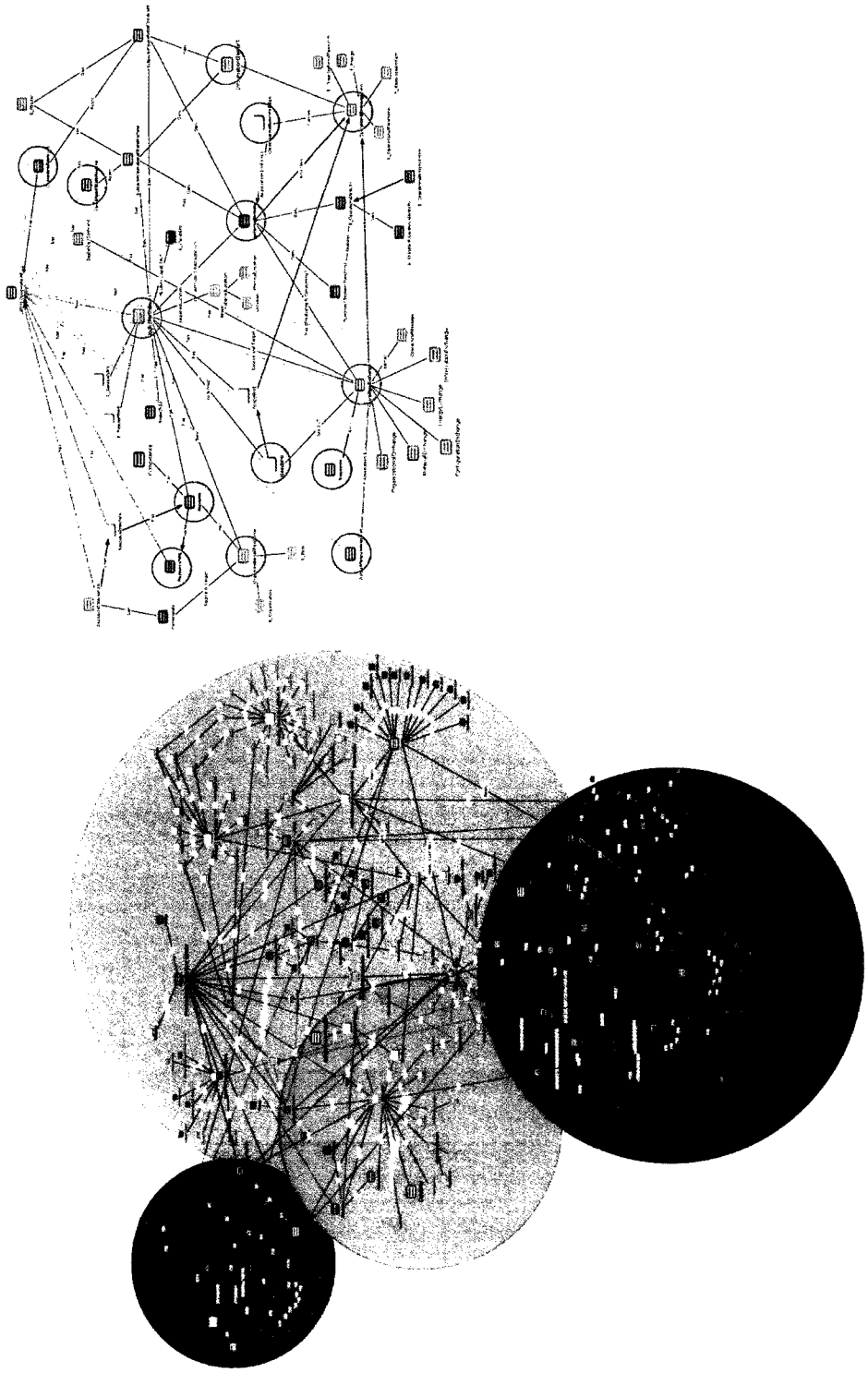


Figure 35 - EAS and UPDM Comparison

Table 28 - Comparisons 1

Element	Color	Specification	Composite UPDM	Composite All	Comments
Performed Activity			X	X	
Event (Logical)					Process flow control or determinant (logical). Control of token, data, message and signal generation
		None		X	
		Signal		X	
		Message		X	
		Multiple		X	
		Conditional		X	
		Cancel		X	
		Terminate		X	
		Escalation		X	
		Parallel Multiple		X	
		Compensation		X	
		Link		X	
Structured ActivityNode			p		Logical and ordering control of subordinate nodes or activity groups. Nesting is not new to UPDM, but control is not specified in detail
		Loop Node		X	Structured activity node that represents a loop
		Conditional Node		X	Structured activity node that represents an exclusive choice among alternatives
		SequenceNode		X	Order specification of actions
		ExpansionRegion		X	Nested region with explicit inputs and outputs
Node			X	X	
OrganizationalResource			X	X	
Resource			X	X	
ResourceRole			X	X	
Competence			X	X	
Operational Exchange			X	X	
OperationalExchangeItem			X	X	
InformationElement			X	X	
State					
Data Characteristics					Descriptive information about data
		Properties / attributes (e.g., cost, size, priority, etc.)	X	X	Detailed description of data and characteristics supporting model analysis
		Structure (Entity item, attributes, relationships)	X	X	Data Structure, semantics & syntax
		Data State	X	X	State of the data or information element
Sequence			X	X	Time ordered events and messaging in a sequence diagram
Event Timer	3				Time based control of process flow
		TimeDate		X	TimeDate Time trigger
		TimeCycle		X	TimeCycle Time trigger
		TimeDuration		X	TimeDuration Time trigger
Control Elements (Time)	5				Time based control of elements (activities, processes), rate, duration, time constraints, general ordering and termination and creation event
		Rate		X	Rate of object flow across activity edge or rate or into or out of parameter
		Time Constraint		X	Behavior or activity occurrence at certain time interval or time
		Duration Constraint		X	Duration of action
		Destruction Event		X	End of event or action
		GeneralOrdering		X	Sequencing of activities

Table 29 - Comparisons 2

Element	Color	Specification	Composite UPDM	Composite All	Comments
OperationalConstraint			P		Detailed specification of Operational Context (viewpoint, mission, scope) or Rule sets
		Viewpoint		X	Partial, more detailed elaborations
		Mission		X	Operational Context
		Rule (expression)		X	Fine grained logical control
		Scope		X	Activity context
		Context		X	Functional Environment, context
Activity Control Elements					Behavior control Stochastic behavior specification (monte carlo, probabilities, non-determinism); execution specifications, execution control
		Probability (edge)		X	Stochastic behavior / Monte Carlo Simulations
		ControlValue		X	Allows control values to be treated as data for enabling and disabling behavior (actions)
		Constraint block		X	Delimiting property, similar to Rule or Expression of Operational constraint
		LocalPreandPostConditions		X	Pre and post condition global constraints that apply to activity
		BehavioralFeature		X	Specification of aspect of behavior
Communications Diagram Control Annotations					Controls behavior of multiple nested activity regions (Expansion Regions)
					Fined grained control of messaging in Communications Diagram: Logical and time-based control of communications diagrams
		Sequence Expression		X	Procedural nesting
		Iteration		X	Sequence of messages at given nestin depth
		Guard		X	Message execution dependent on truth of some condition clause
		Condition Clause		X	Boolean predicate

Table 30 - Augmentation Synopsis and Categorization

Row Number	Element Generalization	Nature of Elemental Augmentation	Primary Category	Secondary Category	Number of Element:
1	Events	Process flow control or determinant (logical). Control of token, data, message and signal generation			11
2	Structured Activity Node Control	Logical and ordering control of subordinate nodes or activity groups. Nesting is not new to UPDM, but control is not specified in detail			4
3	Event Timer	Time based control of process flow			3
4	Control Elements (Time)	Time based control of elements (activities, processes), rate, duration, time constraints, general ordering and termination and creation event			5
5	Sequence or Event Trace Control	Event Trace behavior description / control			6
6	Gateways	Detailed logical control of process flows			12
7	Sequence Flow Control	Detailed logical control of process flows and flow ordering			16
8	Operational Constraints	Detailed specification of Operational Context (viewpoint, mission, scope) or Rule sets			5
9	Activity Control Elements	Behavior control: Stochastic behavior specification (monte carlo, probabilities, non-determinism); execution specifications, execution control			6
10	Communication Diagram or messaging control	Fined grained control of messaging in Communications Diagram: Logical and time-based control of communications diagrams			4
11	State Transition Control	Logical control of state transitions			10

#### 4.4.8 Step 8: Coding Model Simulation Formalisms

Step 8 is the coding of the M&S formalism (i.e., CP-net and DEVS). Figure 36 shows the Executable Architecture Concept Triangle with the Modeling Formalisms component highlighted on the right vertex. The focus of this section will be on how Modeling and Simulation Formalisms can be leveraged to provide a plausibility check for the composite meta-model.

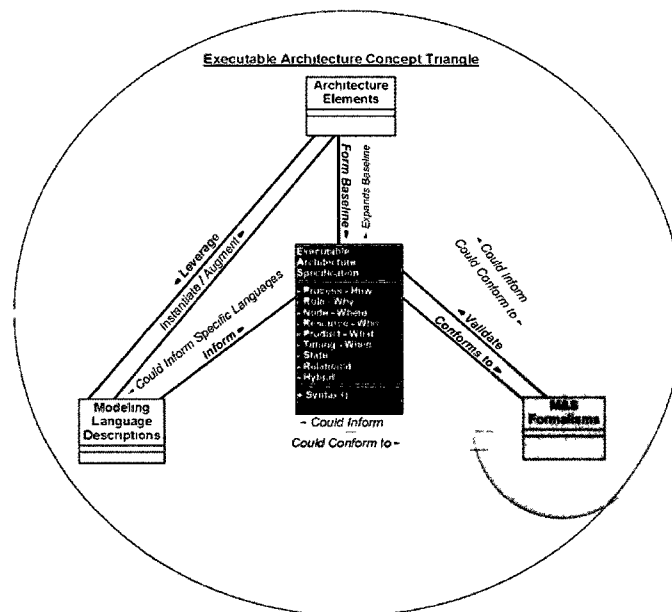


Figure 36 - Executable Architecture Triangle (Modeling Formalisms Guidepost)

After modeling language analysis, the third major component of this investigation was a validating step, during which composite meta-model findings were compared to modeling formalisms that describe behavior modeling. This is a validation step that includes both elemental and relational comparisons. Elemental comparison entails one-to-one or one-to-many comparisons. The relational comparisons were done by comparing relationships of elements in the formalism to relationships in the composite UPDM-Language meta-model. The elements of the Executable Architecture Specification were examined in the context of two prominent, well-established modeling

formalisms: Coloured Petri Nets (CP-net) and the Discrete Event System Specification (DEVS). As with language meta-models, both CP-net and DEVS were coded using in-vivo coding in MAXQDA.

A modeling formalism for executable architectures should holistically describe the elements of an executable architecture using a standard mathematical notation (Tolk, et al., 2010). Comparisons of model formalisms and composite UPDM-Language elements can provide a basis for determining the degree to which the composite UPDM – Language meta-model supports simulation. From the opposite perspective, such comparisons can provide a basis for determining whether there are any obvious gaps in coverage. Two seminal references were used as the basis for formalism coding: “Coloured Petri Nets Basic Concepts, Analysis Methods, and Practical Use” (Jensen, 1992) for CP-net coding, and “Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems” (Zeigler, et al., 2000) for DEVS.

#### **4.4.8.1 Coloured Petri Nets**

The objective here is to provide a holistic, formalism-based comparison to the derived meta-model, by showing traceability between the elements of CP-net and the composite meta-model, thereby suggesting holism or well roundedness of the meta-model construct. The purpose of identifying the elements in the CP-net was to ensure that all CP-net elements were accounted for in the composite meta-model, and thereby to ultimately ensure representation in the elements of executable architecture.

Table 31 is an elaboration on Table 3, presented in the literature review, in that it provides an additional column for elemental interrogative interpretations.

Table 31 - CP-net Elements

Code	Formal Definition	Interpretation	Interrogatives +
<b>Transitory Objects</b>		Ephemeral objects (messages and data)	What (Product) orange
Token colour		Attributes associate with Tokens	What (Product) orange
Tokens		Dynamically varying black dots associated with a place	What (Product) orange
Global Declaration node		Defines all colour sets	What (Product) orange
<b>CP-net Control Elements</b>		Control functions and definitions	Why (Rule) pink
Colour Sets ( $\Sigma$ )	$\Sigma$ finite set of non-empty types	Each token on a place p must have a token colour that belongs to type C(p)	Why (Rule) pink
Initialization function (I)	Defined from P into closed expressions such that $\forall p \in P [Type(I(p)) = C(p)_{ms}]$	Initial marking	Why (Rule) pink
Arc expression (E)	$\forall a \in A [Type(E(a))C(p(a))_{ms} \wedge Type(Var(E(a))) \subseteq \Sigma]$ where p(a) is the place of N(a)	Maps each arc, a, to an expression of type C(p(a))	Why (Rule) pink
Guard function (G)	It is defined from T into expressions such that $\forall t \in T [Type(G(t)) = B \wedge [Type(Var(G(t))) \subseteq \Sigma]$	Additional constraint (Boolean) enabling transition	Why (Rule) pink
Node function (N)	Defined from A into $P \times T \cup T \times P$	(v) The node function maps source and destination nodes	Relationship yellow
Color function ©	Defined from P into $\Sigma$	C maps each place, p, to a colour set C(p)	Why (Rule) pink
<b>Fixed Objects</b>		Fixed objects (nodes and links)	Hybrid purple
Places (P)	P is a finite set of places	State of a resource allocation, or of process (circle)	Where (location) aqua
Port Place		Connections for communication between Objects	Relationship yellow
Arcs (A)	A is a finite set of arcs such that $P \cap T = P \cap A = T \cap A = \emptyset$	Connects a place with a transition or a transition with a place	Relationship yellow
Hierarchical structure		Hierarchical structure is developed for the CP-net	Relationship yellow
Transitions (T)	T is a finite set of transitions	Actions of resource allocation system (rectangle)	How (Functional) green

Figure 37 shows the same Colored Petri net elements depicted from a meta-model, relational perspective, in which graphical relationships were derived from formal definitions and through the verbal descriptions of these elements (Jensen, 1992)

The interrogative color attributes are useful in helping to make visual comparisons between elements. This comparison will be accomplished by using Figure 38, which shows a hierarchical, top-down depiction of the CP-net elements along with Figure 34. The intermediate elements in Figure 38: Fixed, Transitory, and Control Elements are categories suggested by Wagenhals, Haider and Levis (2002). The tags extending from the leaf elements in Figure 38 show alignment of similar elemental, derived from comparison of the CP-net top-down model and the composite UPDM-Language meta-model. The elemental alignment described in the research of Wagenhals et al. (2002) was leveraged for validation purposes. Both Figure 37 and Figure 38 are used in Step 9.

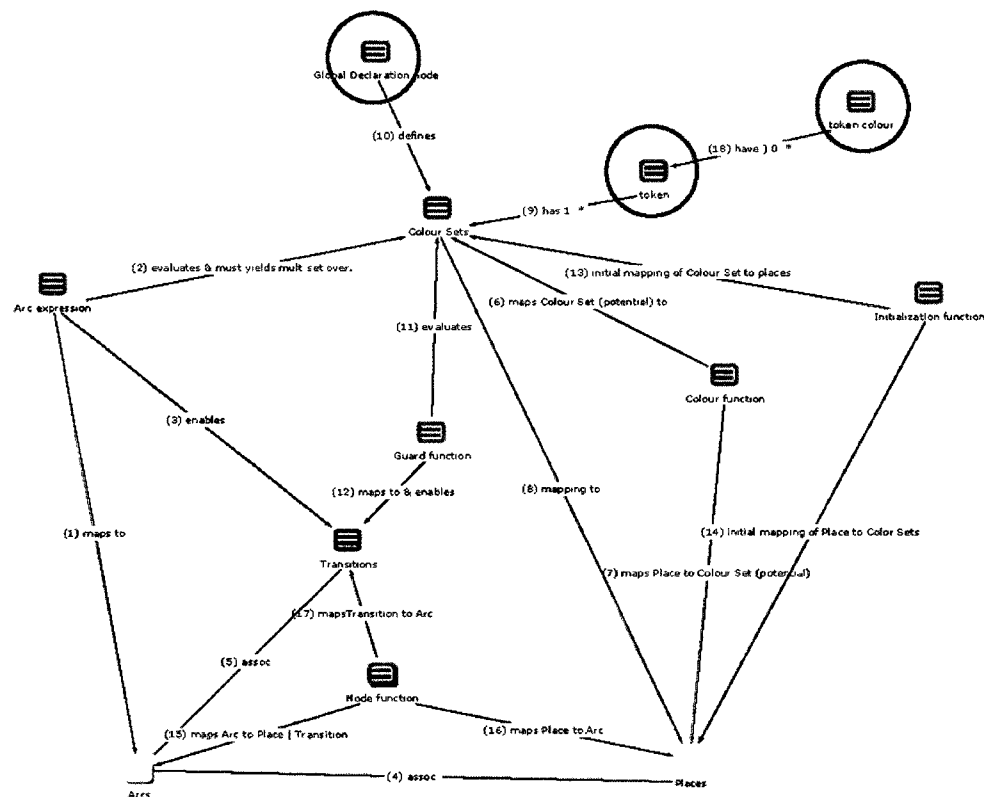
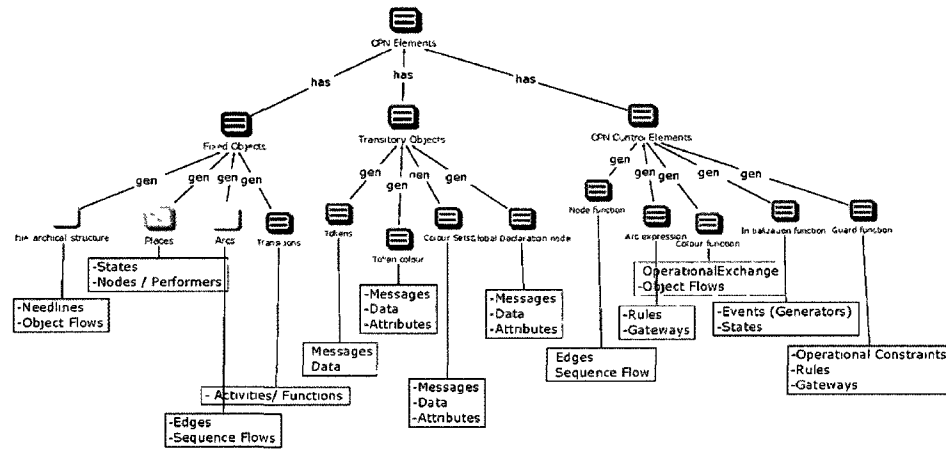


Figure 37 - Non-hierarchical CP-net



**Figure 38 - CP-net Hierarchical Elements and Similar Composite Elements**

#### 4.4.9 Step 9: Compare Simulation Formalism Elements to Composite Meta-model

Step 9 is a comparison step of the formalism elements with the composite UPDM-Language meta-model. Figure 38 provides a flattened out model of CP-net elements. The flattened version was useful in tracing between CP-net and the composite. Comparison between the two is not entirely straight forward, because it depends on how the CP-net model is conceptualized. Tokens can represent resources; they can also represent information flow, as (Wagenhals, et al., 2002) documented in their elegant description of CP-net-based modeling of executable architectures. Figure 38 provides comparisons between CP-net elements and composite UPDM-Language composite model elements. Additionally, resultant alignment comparisons are shown in Table 32. All CP-net elements are addressed by one or more elements within the composite model, lending credibility to the holism of the composite model. Referring back to Table 10, all interrogative classifications (function, node, rule, relationship, product, state, resource) are addressed by CP-net except time (reflected in Table 32). Apart from general ordering, CP-net does not address timing. Candidates 1, 2 and 3 in Table 32 are similar elements from the UPDM-Language Composite that are similar to the CP-net Elements.

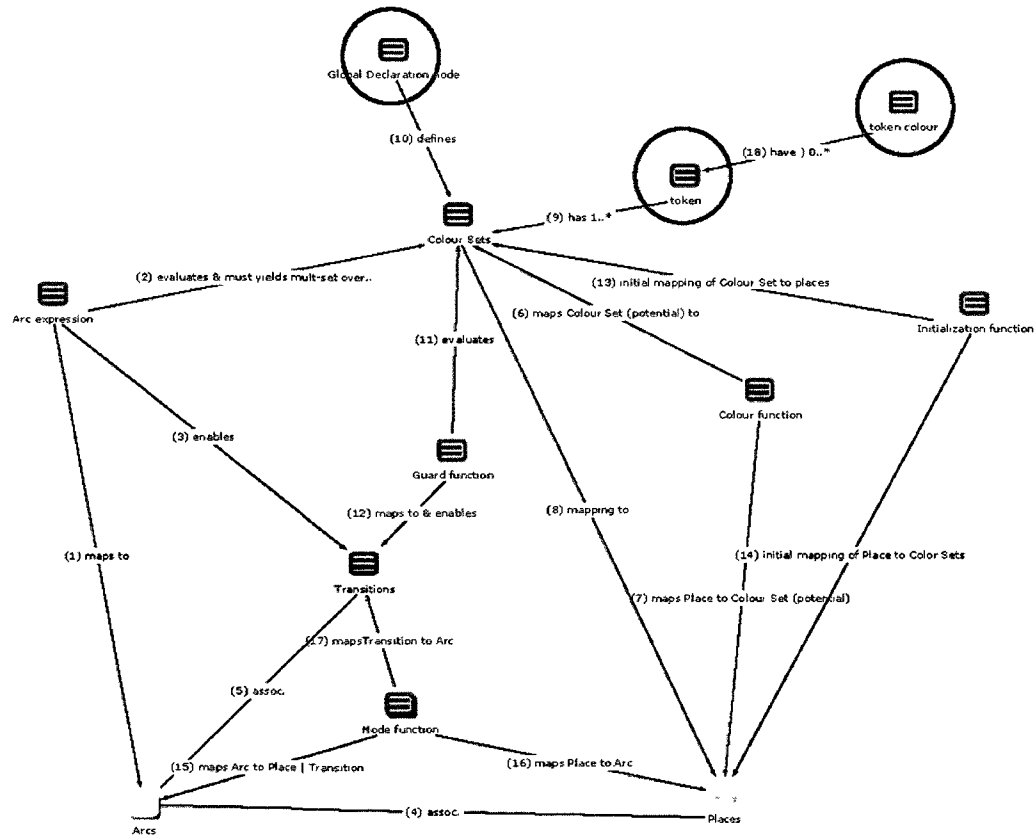
Table 32 - CP-net Cross Model Comparison

CP-net Elements	EAS Model Elements		
	Candidate 1	Candidate 2	Candidate 3
Initialization function	Events (Generators)	State	
Colour function	OperationalExchange	Object Flow	
Arc expression	Rule	Gateway	
Guard function	Rule	Gateway	
Node function	Edges	Sequence Flow	
<b>Transitory Objects</b>			
Tokens	Message	Data	Resource
Token colour	Message	Data	Attribute
Colour Sets	OperationalExchange	Data	Attribute
Global Declaration node	Message	Data	Attribute
<b>Fixed Objects</b>			
Hierarchical structure	Operational Exchange		
Places	Node / Performer	State	
Arcs	Edge	Sequence Flow	
Transitions	Activity / Function		

Figure 39 is a meta-model for CP-net. The relationships from this meta-model are shown in Table 33; in addition, this table shows a comparison of relationships between CP-net and the EAS meta-model. The basic elements in Table 33 were derived from Table 32, but it also includes the relationships between the elements. The table shows the relationship between the element (from) and element (to) for both CP-net and the EAS meta-model.

This table serves two purposes. First, it looks at corresponding relationships between CP-net and the Composite to see if the relationships from the CP-net meta-model exist in the Composite meta-model. CP-net relationships were compared to the corresponding EAS meta-model relationships, and it was determined that they were roughly equivalent. The comparison of some relationships is straight forward. For example, the *Arch Expression* **enables** the *Transition element* in CP-net is equivalent to a Rule **association** to an Activity/Function. Other comparisons become understandable in context. For example, the *Colour Function* and the *Node Function* in CP-net are mathematical formalism functions or rules that map other elements together, and they do not have direct equivalents in the EAS meta-model; however, there are equivalents to the

results of the elemental mappings afforded by these rules. For example, the Node function maps an Arc to a Place or an Arc to a Transition, and there are equivalents to these mappings in the EAS meta-model. The equivalent relationships (shown in Table 33) in the EAS are associations between Node/Performer and Operational Exchange, and Activity/Function and OperationalActivityEdge. Similarly, the result of the action of the Color Function is the equivalent of mapping an association between OperationalExchange and Node/Performer in EAS.



**Figure 39 - CP-net Relationships**

In regard to the Global Declaration Node in CP-net, in the closest analogous element in EAS is all Information Elements. The Global Declaration Node (Jensen, 1992) is described in CP-net but is not part of the classic nine tuple. It is a definition

found in CP-net formalism implementations that is used to describe a declaration of *Colour Sets*, whereas the EAS meta-model operates at a process-modeling level of abstraction that has no need for such definitions, per se. In other words, the Global Declaration Node is used to describe variables found in code level implementations of CP-net. The Composite meta-model describes processes at a higher level of abstraction.

Comparison of CP-net relationships to Composite relationships for validation can be useful (as evidenced by the majority of relationships that do have equivalents), but because of the markedly different levels of abstraction, this comparison does not always produce results in every category. The utility in this approach is revealed by the non-availability of disconfirming evidence. If there were obvious relational gaps in the composite meta-model in comparison to the formalism this would provide evidence of holes in the composite meta-model.

Table 33 - CP-net to Composite Relationship Comparisons

#	Cpnet Element (from)	Relationship	Cpnet Element (to)	EAS Element (from)	Secondary Characteristic	Relationship	EAS Element (to)	Equivalent	Similar to #
1	Arcs	maps to	Arcs	Gateway or Control Node		OperationalActivityEdge	OperationalActivityEdge	yes	
2	Arcs	evaluates & must yields mult set over	Color Sets	Gateway or Control Node		InformationElements	InformationElements	yes	
3	Arcs	enables	Places	Gateway or Control Node		OperationalActivityEdge	OperationalActivityEdge	yes	
4	Arcs	assoc	Places	OperationalExchange		OperationalExchange	OperationalExchange	yes	
5	Arcs	assoc	Places	OperationalExchange		OperationalExchange	OperationalExchange	yes	
6	Color Sets	maps Colour Set (potential) to Place	Color Sets	OperationalExchange		OperationalExchange	OperationalExchange	yes	#8
7	Color Sets	maps Place to Colour Set (potential)	Places	OperationalExchange		OperationalExchange	OperationalExchange	yes	#8
8	Color Sets	mapping to	Places	OperationalExchange		OperationalExchange	OperationalExchange	yes	
9	Color Sets	has	Tokens	InformationElements		InformationElements	InformationElements		
10	Global Data Element Node	defines	Color Sets, all tokens with attributes	InformationElements		InformationElements	InformationElements		
11	Global Data Element	evaluates	Color Sets, tokens with attributes	Gateway or Control Node		InformationElements	InformationElements	yes	
12	Global Data Element	maps to and enables	Places	OperationalExchange		OperationalExchange	OperationalExchange	yes	
13	Initialisation Function	initial map of Colour Sets to Places	Color Sets	OperationalExchange		OperationalExchange	OperationalExchange	yes	#8
14	Initialisation Function	initial map of Places to Color Sets	Places	OperationalExchange		OperationalExchange	OperationalExchange	yes	#8
15	Node Function	maps Arc to Place   Transition	Arcs	OperationalExchange		OperationalExchange	OperationalExchange	yes	#5
16	Node Function	maps Place to Arc	Places	OperationalExchange		OperationalExchange	OperationalExchange	yes	#5
17	Node Function	maps Transition to Arc	Transitions	OperationalActivityEdge		OperationalActivityEdge	OperationalActivityEdge	yes	#5
18	Tokens	has	Tokens	InformationElements		InformationElements	InformationElements	yes	

#### 4.4.10 Discrete Event System Specification (DEVS)

Four basic types of DEVS models were described in the literature review. For each type of DEVS model, a table of elements was developed (Tables 3-6). In this section, a composite table was constructed based on tables 4-7, to reflect the largest possible set of DEVS element configurations. In this section the DEVS variants were represented with a brief description and a tabular synopsis of elements with interrogative elemental descriptions. This set is used as a plausibility check against the composite meta-model, similar to the process completed for CP-net.

##### 4.4.10.1 Classic DEVS

A discrete event system specification (DEVS) is a tuple of seven elements:  $M = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta)$ . Table 34 provides a list of the Classic DEVS elements with definitions and interrogative or color classifications.

Table 34 - Classic DEVS Elements

Code	Definition	Interrogatives +
c	time elapsed since last transition	When (Event- timing) blue
ta	$S \rightarrow R_{0,\infty}^+$ is the set positive reals with 0 and $\infty$	When (Event- timing) blue
Q	$Q = \{(s,e) \mid s \in S, 0 < e < ta(s)\}$ is the total state set	What (State) brown
S	Set of states	What (State) brown
X	Set of input values	What (Product) orange
Y	Set of output values	What (Product) orange
$\delta_{ext}$	$Q \times X \rightarrow S$ is the external transition function	How (Functional) green
$\delta_{int}$	$S \rightarrow S$ is the internal transition function	How (Functional) green
$\lambda$	$S \rightarrow Y$ is the output function	How (Functional) green

##### 4.4.10.2 Parallel DEVS

Parallel DEVS was introduced by Zeigler fifteen years after the Classic DEVS formalism. It removes constraints originating with the sequential operation of early computers that hindered the exploitation of parallelism. A basic Parallel DEVS is described mathematically in the following way:  $DEVS = (x_m, y_m, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$ .

(Zeigler, et al., 2000). Table 35 lists the elements, their definitions and color classifications. Through comparison of color classified elements between Table 34 and

Table 35, it is evident that relationship elements now come into play with the addition of ports. Another key difference is the addition of the Confluent Transition Function, for resolution of collisions between external and internal events. [It may be observed that the Confluent Transition Function is an implementation detail, that probably will not come into play at the process modeling level of abstraction.]

**Table 35 - Parallel DEVS Elements**

Code	Definition	Interrogatives +
(ta) time advance function	$S \rightarrow R_{0,\infty}^+$ is the set positive reals with 0 and $\infty$	When (Event-timing) blue
(Q) set of total states	$Q = \{(s,e) \mid s \in S, 0 < e < ta(s)\}$ is the total state set	What (State) brown
(S) set of sequential states	set of states	What (State) brown
( $X_m$ ) set of input ports and values	set of input values and ports	Relationship yellow
( $Y_m$ ) set of output ports and values	set of output values and ports	Relationship yellow
( $\delta_{con}$ ) confluent transition function	decides next state if collision between external and internal even	How (Functional) green
( $\delta_{ext}$ ) external state transition	$Q \times X \rightarrow S$ is the external transition function	How (Functional) green
( $\delta_{int}$ ) internal state transition	$S \rightarrow S$ is the internal transition function	How (Functional) green
( $\lambda$ ) output function	$S \rightarrow Y$ is the output function	How (Functional) green

#### 4.4.10.3 Parallel DEVS with a buffer

An elaboration on the DEVS formalism is the explicit inclusion of a buffer, V, which functions as a queue for holding an arbitrary input set. “A processor that has a buffer is defined in Parallel DEVS as:  $DEVS_{processing\_time} = (X_m, Y_m, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta)$  (Zeigler, et al., 2000). The Queue (V) was classified as a where interrogative. Interestingly, there are no other explicit types that fall into this classification, although this category is implied by virtue of object association to the functional and state categories. Table 36 shows the elements of DEVS with a buffer. The V Queue is labeled as a where, or node interrogative element.

Table 36 - DEVS Processor with a Buffer

Code	Definition	Interrogatives +
$(X_m)$ set of input ports and values	set of input values and ports	Relationship yellow What (Product) orange
$(Y_m)$ set of output ports and values	set of output values and ports	Relationship yellow What (Product) orange
(V) Queue	V is a queue that holds an arbitrary set or a bag	Where (Node) Aqua
(ta) time advance function	$S \rightarrow R_{0,\infty}^+$ is the set positive reals with 0 and $\infty$	When (Event- timing) blue
(S) set of states	Set of states	What (State) green
$(\lambda)$ output function	$S \rightarrow Y$ is the output function	How (Functional) green
$(\delta_{int})$ internal state transition	$S \rightarrow S$ is the internal transition function	How (Functional) green
$(\delta_{ext})$ external state transition	$Q \times X \rightarrow S$ is the external transition function	How (Functional) green
$(\delta_{con})$ confluent transition function	Decides next state if collision between external and internal even	How (Functional) green

#### 4.4.10.4 Classic Coupled DEVS

Classic Coupled DEVS is an elaboration on the Classic DEVS, providing a means to build complex models from component models. The specification for DEVS with ports includes the external interface (input and output ports and values), the components (which must be DEVS models), and the coupling relations:  $N = \{X, Y, D, \{M_d \mid d \in D\}, \text{EIC, EOC, IC, Select}\}$  (Zeigler, et al., 2000). From an interrogative classification point of view, in comparison to Classic DEVS, the addition of input and output ports and values results in additional Relationship elements. Table 37 shows the Classic Coupled DEVS Elements.

Table 37 - Classic Coupled DEVS Elements

Code	Definition	Interrogatives +
(D) component names	Set of the component names	
(IC) internal coupling	Connects component outputs to component inputs	Relationship yellow
(EOC) external output coupling	Connects component outputs to external outputs	Relationship yellow
(EIC) external input coupling	Connects external inputs to component inputs	Relationship yellow
$(X_d)$ set of input ports and values	set of input values and ports	Relationship yellow What (Product) orange
$(Y_d)$ set of output ports and values	set of output values and ports	Relationship yellow What (Product) orange
(Y) output ports and values	Set of output ports and values $Y = \{(p, v) \mid p \in OPorts, v \in Y_p\}$	Relationship yellow
(X) input ports and values	Set of input ports and values $X = \{(p, v) \mid p \in IPorts, v \in X_p\}$	Relationship yellow
$(M_d)$ DEVS Model	$M_d = (X_d, Y_d, S, \delta_{ext}, \delta_{int}), \lambda, ta)$ is a DEVS	Is (Type) green
$X_d$	$X_d = \{(p, v) \mid p \in IPorts_d, v \in X_p\}$	
$Y_d$	$Y_d = \{(p, v) \mid p \in OPorts_d, v \in Y_p\}$	
Select	Tie-breaking function (used in Classic DEVS	How (Functional) green

Table 38 is a composite listing of all DEVS elements: Classic DEVS, Parallel DEVS (with a buffer), and Classic Coupled DEVS. This represents a union set, which is the broadest possible set of DEVS elemental possibilities. Elements in this table were annotated with the interrogatives to support DEVS union set comparisons with the composite UPDM-Language meta-model, as a plausibility check.

When this table was originally constructed, the going in argument was agnosticism with respect to whether the DEVS element was a process modeling element or implementation specific. Since DEVS was being used as a plausibility check, it made sense to use the broadest possible set. It is now evident that some of the elements, such as the Confluent Transition Function and the Time Advance Function are implementation-level components.

Figure 40 provides a top-down depiction of the DEVS elements. Each element has an annotated tag attached to it that lists the candidate composite UPDM-Language elements. Each DEVS element was traced to the corresponding elements in the composite model, and the result set is represented in Table 38. Figure 41 shows the traces between the top-down DEVS model (from Figure 40) elements and the composite UPDM-Language meta-model elements.

Table 38 - Composite DEVS Elements

Code	Model Origin	Interrogatives +
(D) Component names	Classic DEVS Coupled Models	Other (diagram)
$(\delta_{CON})$ confluent transition function	Parallel DEVS	How (Functional) green
$(\delta_{ext})$ External Transition Function	Classic DEVS	How (Functional) green
$(\delta_{int})$ Internal Transition Function	Classic DEVS	How (Functional) green
(e) Time Elapsed Since Last Transition	Classic DEVS	When (Event- timing) blue
(EIC) external input coupling	Classic DEVS Coupled Models	Relationship yellow
(EOC) external output coupling	Classic DEVS Coupled Models	Relationship yellow
(IC) internal coupling	Classic DEVS Coupled Models	Relationship yellow
$(\lambda)$ Output function	Classic DEVS	How (Functional) green
(Q) TotalStateSet	Classic DEVS	What (State) brown
(S) Set of States	Classic DEVS	What (State) brown
(ta) Time advance function	Classic DEVS	When (Event- timing) blue
(V+) Queue	DEVS Processor with Buffer	Where (Node) Aqua
$(X_m)$ set of input ports and values	Parallel DEVS	Relationship yellow
$(Y_m)$ set of output ports and values	Parallel DEVS	Relationship yellow
Select	Classic DEVS Coupled Models	How (Functional) green

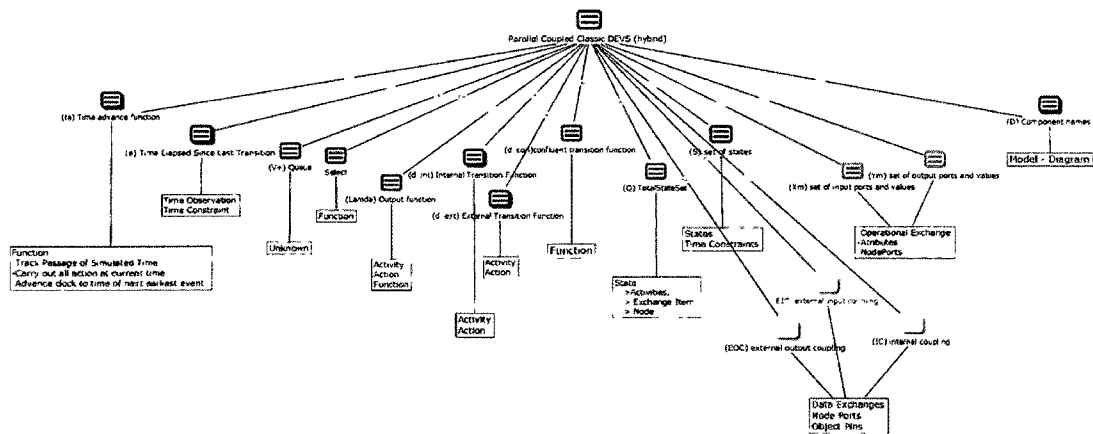
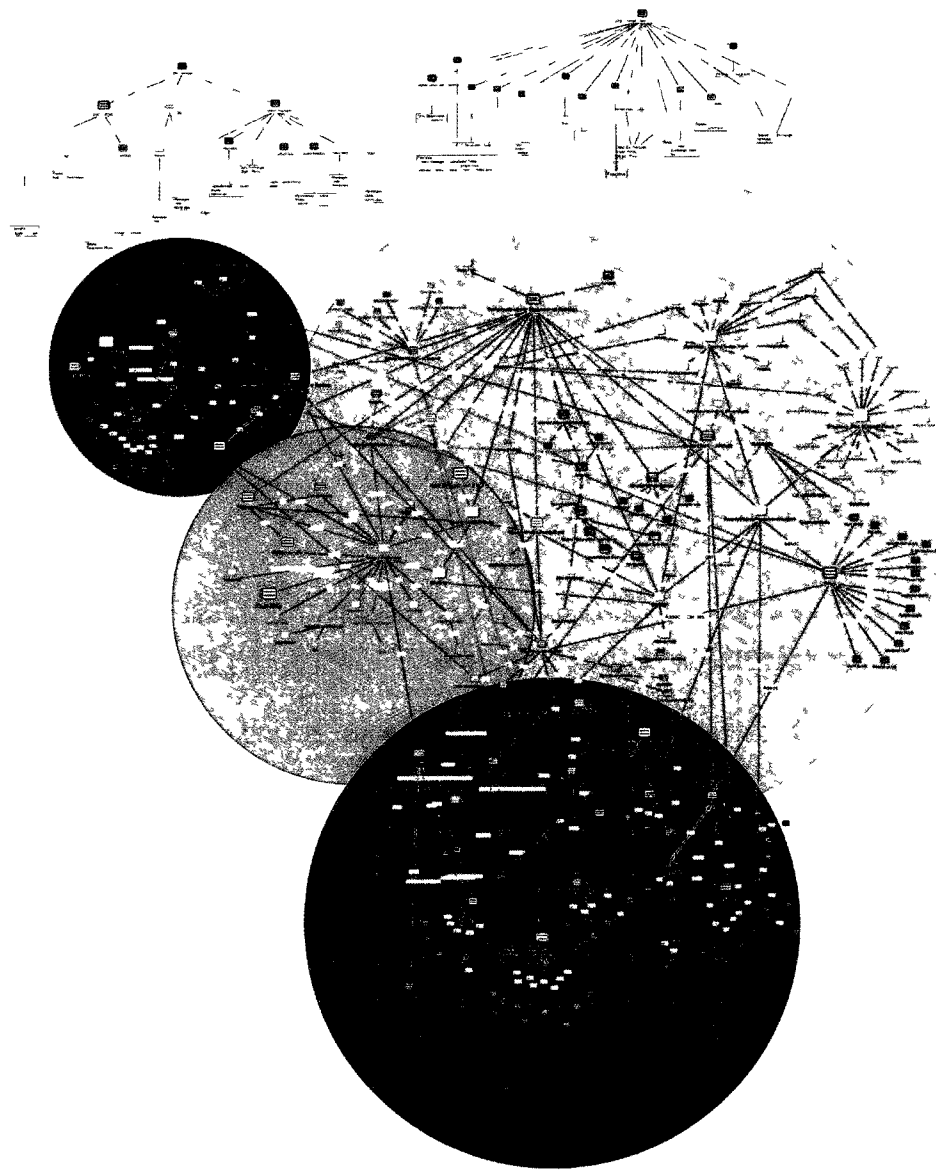


Figure 40 - Composite DEVS

Table 39 shows the results of the traces between DEVS elements and composite UPDM-Language meta-model elements. It lists candidate composite model associations

in the columns entitled candidates 1-3. These columns actually represent populated attributes in MAXQDA such that Table 39 was produced as a report set. The “Interrogatives +” column provides the interrogative or color classification of the DEVS elements. This classification provided a basis for finding candidate elements in the composite model. The only interrogative category not directly represented in the DEVS composite table is the Rule category. Referring back to Table 10, that particular category was defined as a process modifier, similar semantically to an adverb, which modifies a verb (function or process). As such, the rule category may be viewed as subsumed by or as part of the process category. None of the DEVS elements is without a composite meta-model element association. However, there are many elements in the UPDM-Language meta-model that go beyond the *prima facie* associations under DEVS. This is to be expected, as the DEVS formalism is intentionally minimalistic and reductionist.

It was interesting that in Mittal’s (2006) research there were fewer direct correspondences between UML (used to model DODAF) and DEVS elements than one would expect, and this invited further exploration. The purpose of this investigation was to develop a holistic specification for executable architectures, with sufficient depth and richness of *semantic* and *syntactic* detail while exploring a method for doing so. As such, the results could be used to define a future Architecture Framework that would support executable architecture. One of the findings of this investigation is that the level of granularity in DEVS is not sufficient for describing executable architectures. An Architecture Framework requires both static and dynamic modeling along with sufficient specificity, which goes beyond Discrete Event Simulation; it must also provide a common frame of reference, so that as far as possible ambiguities are avoided. *The end state of an Architecture Framework is development of Models and Simulations that support Systems Engineering in complex system of systems engineering spaces, which by definition requires collaborative development of systems engineering products. This is so because in system of systems engineering, the systems are not under the purview of any one person or group, and therefore the modeling of those systems must be done in partnership with others, requiring a common lingua franca, for sharing of these views and simulations.*



**Figure 41 - EAS with Formalism Traces**

Table 39 - DEVS Element Comparisons

Code	Model Origin	Interrogatives +	UPDM – Language Composite Model Elements		
			Candidate 1	Candidate 2	Candidate 3
(D) Component names	Classic DEVS Coupled Models	Other (diagram)	Model		
$(\delta_{CON})$ confluent transition function	Parallel DEVS	How (Functional) green	Function		
$(\delta_{ext})$ External Transition Function	Classic DEVS	How (Functional) green	Activity	Action	Sequence
$(\delta_{int})$ Internal Transition Function	Classic DEVS	How (Functional) green	Activity	Action	Sequence
(e) Time Elapsed Since Last Transition	Classic DEVS	When (Event-timing) blue	Time Observation	Time Constraint	
(EIC) external input coupling	Classic DEVS Coupled Models	Relationship yellow	Operational Exchange	Node Ports	
(EOC) external output coupling	Classic DEVS Coupled Models	Relationship yellow	Operational Exchange	Node Ports	
(IC) internal coupling	Classic DEVS Coupled Models	Relationship yellow	Operational Exchange	Node Ports	PIn
( $\lambda$ ) Output function	Classic DEVS	How (Functional) green	Activity	Action	Function
(Q) TotalStateSet	Classic DEVS	What (State) brown	State		
(S) Set of States	Classic DEVS	What (State) brown	State	Time Constraints	
(ta) Time advance function	Classic DEVS	When (Event-timing) blue	Function	Timer	
(V+) Queue	DEVS Processor with Buffer	Where (Node) Aqua	Activity Parameter Node	PIn	
$(X_m)$ set of input ports and values	Parallel DEVS	Relationship yellow What (Product) orange	Operational Exchange	Operational Exchange Item	Node Ports
$(Y_m)$ set of output ports and values	Parallel DEVS	Relationship yellow What (Product) orange	Operational Exchange	Operational Exchange Item	Node Ports
Select	Classic DEVS Coupled Models	How (Functional) green	Function		

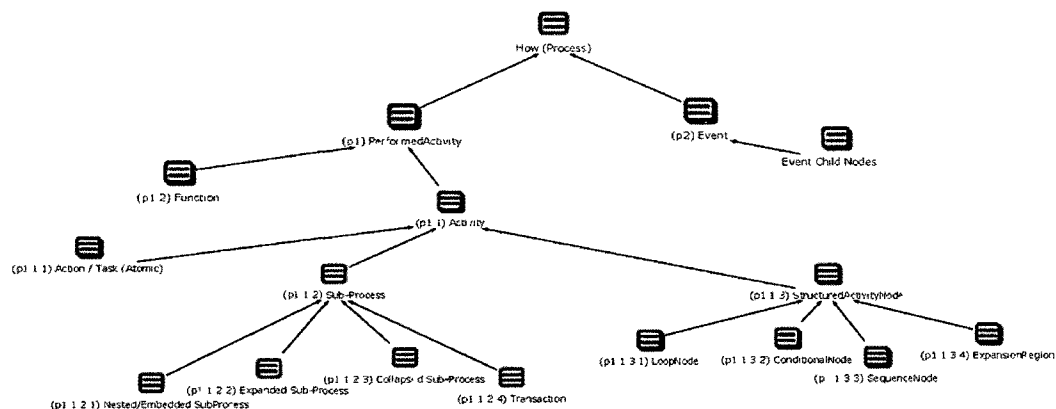
#### 4.4.11 Step 10: Define Elementals as Sets Based on Ontologies & Interrogatives

Step 10 is the further defining of the EAS meta-model as an ontology. This may be seen in Figure 42, which starts with the How (i.e., process) interrogative element, as the root node, and branches down in terms of parent-child relationships.

Figure 42 is the process category, organized ontologically. Parent-child relationships were derived from the EAS meta-model, which is captured in a meta-model

drawing. The original source of these relationships was both the authoritative source for the contributing model (i.e., UML, SysML, BMNN, IDEF), and new relationships that were discovered by creating the composite meta-model. The ontology influence the EAS meta-model, and the development of the two were an iterative, complementary process.

Parent-child relationships in the meta-model are indicated with annotations on the relationship lines between parent and child, with the arrow head pointing to the parent and the line annotated with “gen.” for generalization (e.g., see Figure 34).



**Figure 42 - Parent-Child Depiction**

Table 40 and 41 depict the same kind of elemental relationship in tabular form for all 9 interrogative categories. The tables are organized from left to right with Interrogative Category (e.g., How) in column A; Name and Designator in column B, e.g., Behavior (P); followed by the first Fork / Node with Designator and Name in column C, e.g., p1 Performed Activity; followed by the second Fork / Node, e.g., p1.1 Activity, and so on. After the first Fork / Node, the alpha numeric dot designator is used, e.g., p1.1 to indicate Fork / Node levels.

A series of tree graphs could be generated for each interrogative by traversing the table from root to leaf nodes. Additionally, the definition for each element was recorded with coded elements in the MAXQDA database. A composite tree-view ontology was

not constructed in a separate hierarchy in MAXQDA, because this is viewed as a further practical implementation of the method.

There are nine interrogative classifications shown in Table 40, and 43: How (Process), Why (Rule), Where (Node), Who (Resource), What (Product), Being (State), When (Timing), Relationship, and Hybrid. As previously discussed, this investigation started with the six interrogatives but expanded to the nine to address those categories that did not fit into the six. Most elemental classifications were straightforward. However, in some cases the classification is a bit messy and definitely not perfect; for example, Gateways & Control Nodes have characteristics of both the relational category and the functional category. For this study, the relational category was chosen for both, to support elemental comparisons. Categorization may be viewed as a useful tool that reflects the ontological nature of the element, and while useful, as it provides an organizational mechanism for building a schema that accounts for elements. Some of the elements have characteristics that could allow for classification according to more than one type.

Table 40 - Elements 1 (Executable Architecture Tree)

Root Node (Fork 1)	Fork 2/Node	Fork 3/Node	Fork 4/Node	Fork 5/Node	Node
Interrogative Category	Designator & Name	Designator & Name	Designator & Name	Designator & Name	Designator & Name
How Processes (P)	p1 Performed Activity	<u>p1.1 Activity</u> p1.2 Function	<u>p1.1.1 Action / Task</u> <u>p1.1.2 Subprocess</u>	p1.1.1.1 Send-SignalAction p1.1.1.2 Receive-SignalAction <u>p1.1.1.3 Structured-Activity-Node</u>  p1.1.2.1 Nested/Embedded SubProcess p1.1.2.2 Expanded Sub-Process p1.1.2.3 Collapsed Sub-Process p1.1.2.4 Transaction	p1.1.1.3.1 Loop Node p1.1.1.3.2 Conditional Node p1.1.1.3.3 SequenceNode p1.1.1.3.4 ExpansionRegion
	p2 Event (Token Generation)	p2.1 Cancel p2.2 Compensation p2.3 Conditional p2.4 Error p2.5 Escalation p2.6 Link <u>p2.7 Message</u> p2.8 Multiple p2.9 None p2.10 Parallel Multiple <u>p2.11 Signal</u> p2.12 Terminate	p2.7.1 Trigger p2.7.2 CallEvent  p2.11.1 SendSignalEvent p2.11.2 ReceiveSignalEvent		
Why Rule (M)	m1 Communication Diagram Control	m1.1 Sequence Expression m1.2 Iteration m1.3 Guard m1.4 Condition Clause			
	m2 Sequence Diagram Logical Constraints	m2.1 Sequence m2.2 Parallel m2.3 Loop m2.4 Option m2.5 Ignorance/Condition m2.6 Negative			
	m3 Operational Constraint	m3.1 Viewpoint m3.2 Context m3.3 Misuse m3.4 Expression			
	m4 Activity Control Elements (logical)	m4.4.1 Probability (edge) m4.4.2 ControlOperator m4.4.3 LocalPreandPostConditions m4.4.4 BehaviorFeature m4.4.5 ExpansionKind m4.4.6 Constraint block			
	m5 Pseudostate - State Control	m5.1 Fork m5.2 Join m5.3 Junction m5.4 Choice m5.5 Entry Point m5.6 Exit Point m5.7 Terminate m5.8 Shallow History m5.9 Deep History m5.10 Initial pseudostate			
When Timing Constraints (T)	t1 OperationalEventTrace-Sequence				
	t2 Event Timer	t2.1 TimeDate t2.2 TimeCycle t2.3 TimeDuration			
	t3 Control Elements (Time)	<u>t3.3.1 Rate</u> <u>t3.3.2 Timing Diagram</u> <u>t3.3.3 Simple Time</u>	t3.3.1.1 Continuous t3.3.2 Discrete  t3.3.2.1 Destruction Event t3.3.2.2 GeneralOrdering  t3.3.3.1 Time Constraint t3.3.3.2 Duration t3.3.3.3 DurationInterval t3.3.3.4 Interval Constraint t3.3.3.5 Observation t3.3.3.6 TimeEvent t3.3.3.7 TimeExpression	t3.3.3.1 Duration-Constraint t3.3.3.2 Time-Constraint	
	t4 Sequence Diagram Timing Constraints	t4.4.1 Duration Constraint t4.4.2 TimeConstraint t4.4.3 TimeObservation t4.4.4 DurationObservation t4.4.5 GeneralOrdering t4.4.6 DestructionEvent t4.4.7 General Value Lifeline t4.4.8 CreationEvent			
Being State (S)	s1 State				

Table 41 - Elements 2 (Executable Architecture Element Tree)

Root Node (Fork 1)	Fork 2/Node	Fork 3/Node	Fork 4/Node	Fork 5/Node	Node
Interrogative Category	Designator & Name	Designator & Name	Designator & Name	Designator & Name	Designator & Name
Information Knowledge (I)	n1 Node / Performer	n1.1 Umlife n1.2 Partition n1.3 NodeChild n1.4 OrganizationalResource	n1.2.1 Pool n1.2.2 Swimlane n1.2.3 Region n1.4.1 Port n1.4.2 Organization		
	n2 Pin				
	n3 Node Port				
	n4 Gate				
	n5 Queue				
Resource Knowledge (R)	r1 Resource				
	r2 ResourceRole				
What Knowledge (Q)	q1 Operational Exchange Item	q1.1 Energy q1.2 OrganizationalResource q1.3 Resource Artifact q1.4 Information Element (Data)			
	q2.4 Information Element (Data)	q2.4.1 Message q2.4.2 Data Object q2.4.3 DataObject References q2.4.4 Parameter q2.4.5 ParameterSet q2.4.6 Data Store	q2.4.1.1 Found Messages q2.4.1.2 Lost Message q2.4.1.3 Object creation Message q2.4.1.4 Reply message q2.4.1.5 Synchronous Messages q2.4.1.6 Asynchronous Message		
	q3 Data Characteristics - Attributes	q3.1 Properties q3.2 Structure q3.3 Data State			
Relationship (C)	C1 Gateways & Control Nodes	c1.1 Initial c1.2 Signal c1.3 Exclusive c1.4 Decision c1.5 Complex c1.6 Merge c1.7 Join c1.8 Fork c1.9 Inclusive c1.10 Parallel c1.11 Event-based c1.12 Control Operator c1.13 Parallel Event-based			
	C2 Operational Activity Edge	c2.1 Sequence Flow (Control Flow) c2.2 Object Flow	c2.1.1 Uncontrolled Flow c2.1.2 Conditional Flow c2.1.3 Merging c2.1.4 Fork c2.1.5 Join c2.1.6 Normal flow c2.1.7 Looping c2.1.8 Mechanism c2.1.9 Control c2.1.10 Unneeded c2.1.11 Compensation Association c2.1.12 Exception Flow c2.1.13 Bundling / Unbundling c2.1.14 Default c2.1.15 Internal Arrow c2.1.16 Boundary Arrow	c2.2.1 Message Flow c2.2.2 Data Association c2.2.3 Operational EventTrace Sequence	
	c3 Operational-Exchange				
	c4 Needline				
	c5 State Transition				
Hybrid (H)	h1 Actual-MeasurementSet				
	h2 Capability				

The motivation for classification of an interrogative category as information or knowledge is shown in the motivation column, such that if the interrogative is the result of associations of information and is therefore complex, or not discrete, it is described as knowledge; otherwise it is information. These interrogatives may be described as the

essential information and knowledge descriptive categories. From a classification perspective there is the suggestion here of holism with respect to elemental categories and interrogative categories: if these interrogatives are the primary *information* and *knowledge* ontological groups for the architecture, then from a category point of view, they should contain all the useful elements for the architecture.

**Table 42 - The Nine Information and Knowledge Interrogatives**

Category	Motivation	Interrogative	Description
Information	Discrete	What	Product
Information	Discrete	Who	Resource
Information	Discrete	Where	Node
Information	Discrete	When	Timing
Knowledge	Complex	How	Process
Knowledge	Complex	Why	Rule, Context
Knowledge	Complex	Hybrid	Combination
Knowledge	Complex	Relationship	Linking Object
Knowledge	Complex	State/Being	Condition

The result of this kind of elemental analysis and synthesis of the data is the development of an organizational ontology of Executable Architecture Specification Elements, based on nine interrogative classifications, where the elements can be described in terms of *information* and *knowledge* categories, as shown in Table 42. This categorization is an expansion of the information and knowledge elements describe by Sage (2009).

Up to this point, the study has been conducted through inductive data analysis by developing a composite UPDM-Language meta-model, called the EAS. In the process, it was validated against formalism elements and compared to a UPDM composite meta-model to see potential language contributions to executable architectures. Next, the study will go into the deductive phase as the results are explored and synthesized through the use of the EAS meta-model.

#### 4.5 EAS Intermediate-level Model

The previous section provided an organizational ontology of Executable Architecture Specification Elements based on the nine interrogative classifications. As stated previously, these interrogatives are the primary *information* and *knowledge* groups for the architecture; from a category point of view, they contain all the useful elements for the architecture.

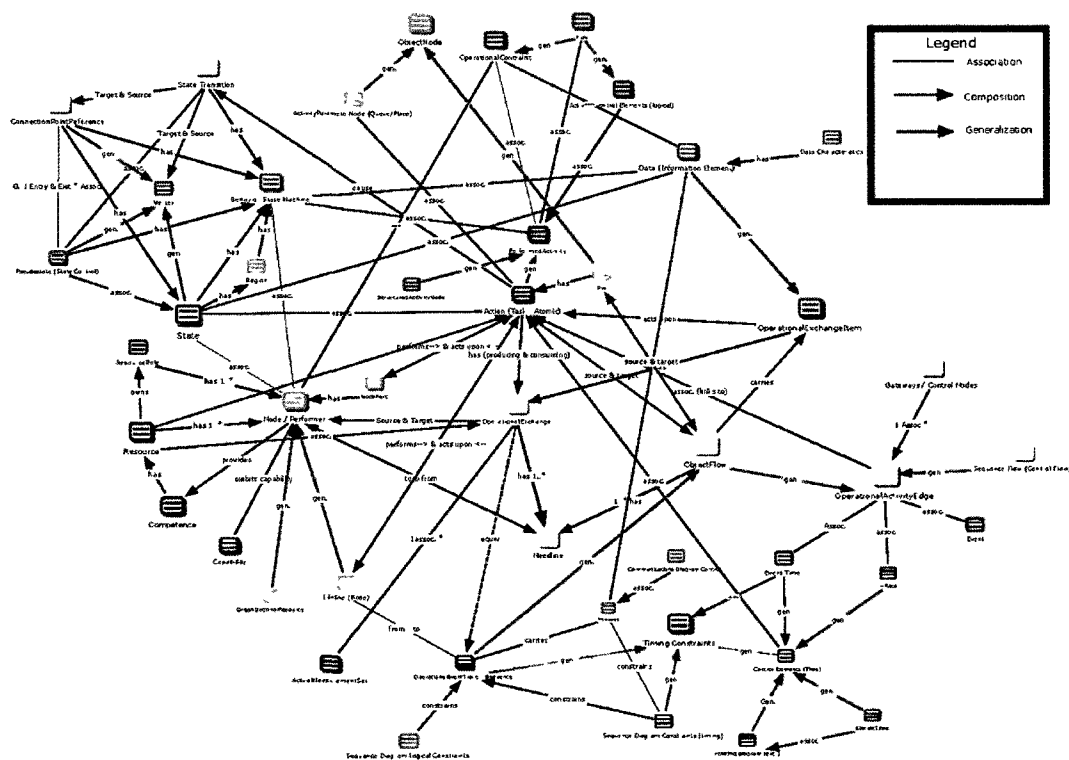


Figure 43 - EAS Intermediate (EASI)

Once a detailed EAS meta-model had been developed, it became apparent that by reducing the detail down to the second fork in the tree structure of the ontology, it would become possible to recognize the elements of highest potential; they became more visible and observable as the less important details were removed. Figure 43 is the resulting

intermediate-level meta-model, called the EAS Intermediate (EASI). The model contains color coded elements and color coded relationships (legend).

The EAS-Intermediate level meta-model appears to be a holistic construct that should support the development of integrated Executable Architectures. Holism here means that both static elements, as defined by the Architecture Framework, are there -- and dynamic elements, as provided by Modeling Language contributions, assessed against M&S Formalism are present -- in the context of the whole, thereby enabling a dynamic modeling construct that is integrated into the reference Architecture Framework.

Tables 43 and 44 will help to validate the holism of this assertion, by comparing the elements from the EASI in the context of the ontological interrogatives (which form the basis for inquiry) against the requirements of the M&S Formalisms. We know from comparisons of the EAS to CP-net (in Table 32) and DEVS (Table 39) that all formalism elements are either present or there by virtue of end-state or effect. Now, we will look at the EASI, which provides a more streamlined view of the EAS, to assess for holism in this revised context.

Table 43 and Table 44 contain the Intermediate-level meta-model elements derived from the EAS-I, from root interrogatives to second level Nodes, as rows. The two tables divide the meta-model elements into static and dynamic elements. The tables have the following principle columns: UPDM, Classic CP-net, DEVS, and EAS-I for element comparison purposes. The colors in the spotlight show the level of element availability in *red*, *yellow*, and *green*. For example, the Node element is present or green in all four implementations: Architecture Frameworks, Classic CP-net, DEVS, and EAS-I.

These tables provide side-by-side comparisons of each element's availability. The comparison to CP-net and DEVS shows the degree to which the element is addressed in the respective formalism. The table indicates that the element is present in green, and not present in red; partial or non-specific availability is indicated by yellow. An annotation of partial means some aspect of the element is not implemented. If it is annotated as non-specific, this means that the element is present but is described in a less specific way; in other words, the description is at a high level of abstraction and less useful for building Executable Architectures.

The EAS-I meta-model is designed to answer the question, which elements are necessary or of high potential for the simulation of process models (i.e., Executable Architectures)? All thirty EAS-Intermediate level elements listed in Table 43 and 44 are considered high potential elements. These potential elements, then, are those which effectively address the interrogative questions across the nine categories: where, who, what, relationship, hybrid, why, when, how, and state.

**Table 43 - EAS Intermediate Static Elements**

Root /Fork 1 Interrogative & Description	#	Node / Fork 2 (EAS Element)	UPDM	Classic CPnet	DEVS	EASI	Notable Characteristics
Where (Node (N))	1	Node	yes	yes	yes	yes	
	2	Pins	yes	no	yes	yes	Activity Connection specificity
	3	Node Port	yes	no	yes	yes	Clear syntax important
	4	Gate	no	no	yes	yes	Sequence Diagram Connectivity
	5	Queues	no	yes			Queue control
Who Resource (R)	6	Resource	yes	non-specific	non-specific	yes	
	7	ResourceRole	yes	non specific	non-specific	yes	
	8	Competence	yes	non-specific	non-specific	yes	Performance measures
What Product (Q)	9	Operational Exchange Item	yes	yes	yes	yes	
	10	Information Element (Data)	yes	yes	yes	yes	
	11	Data Characteristics - Attributes	yes	partial	partial	yes	properties, structure
Relationship (C)	12	Gateways & Control Nodes	partial	no	non-specific	yes	Logical Control, flow control
	13	Operational Activity Edge	yes	yes	yes	yes	
	14	Operational Exchange	yes	non-specific	non-specific	yes	
	15	Needline	yes	non-specific	non-specific	yes	
	16	State Transition	yes	yes	yes	yes	
Hybrid (H)	17	ActualMeasurement-Set	yes	non-specific	non-specific	yes	
	18	Capability	yes	no	no	yes	High level system description

#### 4.5.1 *Static Elements*

Table 43 contains the *static elements*: the Where, Who, What, Relationship, and Hybrid elements. In general, these are the structural elements that do not deal with time. Static elements are described as follows:

**Pins, Port and Gates:** In the static table, in the Node category, ports, pins, and gateways and control nodes are of particular importance in terms of modular, structural design. In the ontology, these elements are Node elements that are used for connecting and are important for building both static and dynamic architectures. The lesser known Gate is similar to a Port for a Sequence Diagram. UPDM does not include Gates, and Classic CP-net does not include Pins, Ports or Gates. However, these constructs are of particular importance in the modular construct of Coupled DEVS where they are referred to as *input and output ports and value*. Pins, Ports, and Gates are also a part of EASI. It should be noted that for these elements to be useful in the context of modular composition, their semantics need greater specificity in order to support modular coupling at the syntactic level.

Table 44 - Intermediate Level Dynamic Elements

Root /Fork 1 Interrogative & Description	#	Node / Fork 2 (EAS Element)	UPDM	Classic CPnet	DEVS	EASI	Notable Characteristics
How Process (P)	19	Performed Activity	yes	yes	yes	yes	
	20	Event (Token Generation)	no	non-specific	non-specific	yes	Token Flow,sequence or timing of Activities of a Process
Why Rule (M)	21	Communication Diagram Control	no	non-specific	non-specific	yes	Message flow control
	22	Sequence Diagram Logical Constraints	no	non-specific	non-specific	yes	Sequencing control
	23	Operational Constraint	yes	non-specific	non-specific	yes	
	24	Activity Control Elements (logical)	no	non-specific	non-specific	yes	Probabilities & Control as Data
	25	Pseudostate - State Control	no	non-specific	non-specific	yes	State Transition Control
When Timing (T)	26	Operational-EventTrace - Sequence (Time)	yes	no	non-specific	yes	Model for sequence depiction
	27	Event Timer	no	non-specific	non-specific	yes	Token Flow,sequence or timing of Activities of a Process
	28	Control Elements (Time)	no	no	yes	yes	Detailed Timing Control
	29	Sequence Diagram Timing Constraints	no	no	non-specific	yes	Detailed Timing Control
Being State (S)	30	State	yes	yes	yes	yes	

**Queues:** Queues provide both a receptacle and a way to manage token arrival. Queues are important to the process modeler, not merely the simulation developer engineer, because specification of queue behavior in terms of ordering (FIFO, LIFO) and in terms of numbers of queues is fundamental to the control of Discrete Event Simulation. Queues and Queue control are critical in process modeling.

**Resource, ResourceRole, and Competence:** In the Resource and Product categories, all elements are represented across UPDM, CP-net, DEVS and EASI, with a few partial or non-specific exceptions, as follows. Resource, ResourceRole, and Competence are addressed by CP-net and DEVS in high level or non-specific ways. Resource and Competence go hand in hand. Both are associated with activity measures of performance. A Resource executes an Activity at a Node. This relationship is described as a triplet (Node, Activity, Role) in DODAF Activities Based Methodology (ABM) (Ring, et al., 2008). However, in this meta-model the relationship has five parts

and is described as a Quintuplet (i.e., Node, Activity, Resource, Competence, and Resource Role). Competence sets the level of performance of a resource. A Resource in UPDM is similar to a Role in DODAF 1.5 (DOD, 2007b); however, it is not limited to human performance in that it includes system actors as well. In BPMN, the Resource Role “defines the resource that will perform or will be responsible for the Activity. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.” (OMG, 2009, p. 154). The following relationships are depicted in the intermediate-level meta-model: *a Resource has a Competence and a Resource Role; a Node has one or more Resources; a Performed Activity is associated with a Resource; and a Performed Activity performs or acts upon a Node.* Because performance measures are critical to process modeling, the Resource, Competence and Resource Role elements should be included with Activity and Node (which are ontologically basic as the How and the Where, respectively) in Executable Architecture Specifications.

**Data Characteristics** are annotated for CP-net and DEVS as partial because data properties are specified but data structure is not. A Data Characteristic is a constituent part of data. In and of itself, it is a vague term that encompasses the attributes of a data entity or of data. Similarly, an “ActualMeasurementSet” is an attribute of a data entity that specifies some measurement such as rate, size, or quantity. The ability to specify attributes associated with data flow, i.e., tokens, is vital to Executable Architecture Specifications.

**Gateways & Control Nodes:** Under the Relationships category, the Gateways & Control Nodes are different for each of the four columns. From left to right, in UPDM, Gateways and Control node functionality is partial in that it offers little control over flow of data and tokens. Classic CP-net does not include control node and gateways. DEVS refers to this capability non-specifically as input and output ports and values, and more obliquely as internal transition functions. EASI, in comparison, has a variety of specific Gateways and Control nodes from contributing languages. Gateways and control nodes are glaringly absent from IDEF0, and very minimal in UML Activity Diagrams. They provide low level logical control flow of tokens in process models. Gateways and

Control Nodes should be considered high potential elements for Executable Architecture Specifications.

**Operational Activity Edges** and **State Transitions** are present in all categories. **Operational Exchanges** and **Needlines** are by definition composite elements in UPDM and EASI, comprised of Nodes, Products, Relationships, and Resources. Both Operational Exchanges and Needlines are key components of most Architecture Frameworks, such as UPDM, DODAF, MODAF, NAF, etc. because these frameworks emphasize interoperability between systems or system of systems constructs, and these elements support the specification and investigation of interoperability within and between systems. In CP-net and DEVS the component parts are there (i.e., Nodes, Products, Relationships, and Resources), but not specifically the composite structures.

**Hybrid:** Within the Hybrid category, Capability, which is a key systems engineering descriptor of system need, is not part of CP-net or DEVS. Arguably, this element could be considered out of scope, as a requirements-like element, but is nevertheless included here as fundamental to Systems Engineering (Buede, 2009). Also, there are a large number of elements in Tables 43 and 44 that have hybrid characteristics but which have been classified under a particular interrogative according to their primary characteristic.

#### 4.5.2 *Dynamic Elements*

Table 44 contains the *dynamic elements*: the How, Why, When, and State elements. In general, these are the behavior elements that deal with time. All of the dynamic elements are very important to building Executable Architectures. Notable deficiencies with respect to Executable Architecture are found in the Process, Rule, and Timing categories, all of which require more specificity.

Reading Table 44 from a vertical perspective, it may be observed that UDPM has deficiencies in the Process, Rule and Time categories. CP-net is deficient in three, and is non-specific in most. DEVS is sufficient in all categories; however, it is non-specific in most. EASI provides sufficient elements in all categories for Executable Architectures, by virtue of the addition of Modeling Language elements.

Reading Table 44 from a horizontal perspective, from left to right, specifics follow: In the How or Process category, the key element **Performed Activity** is present across the board. The **Event or Token Generation** element together with the similar **Event Timer** element (from the Time category) are important in discrete event modeling and to Executable Architectures, for the logical and timing control provided over data flow. The Event element is not addressed in UPDM, and not specifically or fully addressed in CP-net or DEVS. In CP-net, token flow and flow control is basic to the formalism; however, it is predicated on an initial token state (defined by the ***Initialization Function***), and control over timing is not addressed beyond sequencing. In DEVS, the control over data or token flow is addressed, but the notion of a token generator, although inferred, is not specifically defined.

In the How / Process category, both the **Performed Activity** and the Event or Token Generator can generate tokens or data flow. The Event element provides detailed logical control over token, message, signal and data flow. The Event is defined by The Object Modeling Group as:

something that ‘happens’ during the course of a Process. These Events affect the flow of the Process and usually have a cause or an impact. The term ‘event’ is general enough to cover many things in a Process. The start of an Activity, the end of an Activity, the change of state of a document, a Message that arrives, etc., all could be considered Events. However, BPMN has restricted the use of Events to include only those types of Events that will affect the sequence or timing of Activities of a Process (OMG, 2011), p. 83.

It is suggested that “something that happens” be read as a state change. Each of the underlined portions of text above describes a change in state of some kind. The event is a key control element in BPMN. An event is used to define process flow in response to, and in the context of, various stimuli (e.g., message, signal, error, escalation generation). Each of these stimuli may be understood as the arrival of a *token*, as understood and articulated in a Colored Petri-net (Jensen, 1992) context, that is to say, as an attributed object that facilitates process flow in the context of state change.

OMG defines a token as follows:

Throughout this document, we discuss how Sequence Flows are used within a Process. To facilitate this discussion, we employ the concept of a token that will traverse the Sequence Flows and pass through the elements in the Process. A token is a theoretical concept that is used as an aid to define the behavior of a Process that is being performed. The behavior of Process elements can be defined by describing how they interact with a token as it “traverses” the structure of the Process (OMG, 2011), p. 27.

Discrete Event Simulation is a primary method for simulating processes. It is based on the concept that the simulation responds to the arrival and processing of events or tokens at various points in the simulations, from inputs queues, through processing, to output queues, and that time intervals are dictated by the arrival of these events or tokens (Law & Kelton, 2000). As such, event or token control is fundamental to defining dynamic process modeling. For this reason, the Event elements must be included in the Executable Architecture Specification. Finding: The “**Event**” element (both **Logical and Timer**), taken from BPMN, should be included in Executable Architecture Specifications.

In the Rule Category, the **Communication Diagram Control** and the **Sequence Diagram Control** are logical control features derived from UML/SysML that are specifically addressed in EASI but either not at all in UPDM or non-specifically in the other categories. The Sequence Diagram and the related Communications Diagram are vital because they support the sequential diagramming of processes. The UML Communications Diagram, which provides a data or message oriented view of objects, can be derived from the Sequence Diagram. Sequences or Event Traces are generated from the operational nodes, which are represented as lifelines in the Sequence Diagram. The Sequence Diagram is indispensable to modeling sequential processing and is part of UPDM, but the fine grained logical control features that are described as Sequence Diagram Control are not part of UPDM or DODAF. A sequence or event trace is a hybrid element (as shown in Table 45) that includes activity, messaging and time (order). It is nearly impossible to show time ordered sequencing of activities without an event trace, and the ability to specify logical control over the event trace makes Sequence Diagram Control highly desirable as a potential element for Executable Architectures.

**Operational Constraints** also fall under the interrogative Why / Rule category. Operational Constraints were addressed by Garcia (2011) in his dissertation. Operational Constraints provide the operational context, i.e., critical environmental factors that influence the behavior of activities in simulations. They are associated with Performed Activity in the Meta-model. Operational Constraints should be included in Executable Architecture Specifications.

Under the Rule category, **Activity Control Elements (logical)**, is a parent or generalization element for six behavioral controls (one of which is Probability; another is Control Operator) that should be included in Executable Architecture Specifications. This kind of logical control is vital to Executable Architecture specification, and is not addressed in UPDM. The idea of control as data is addressed in CP-net and DEVS, but control as a probability, while it may be inferred, is not directly addressed by either formalism.

**Probability** is a type of Activity Control Element: From a holistic point of view, the “probability stereotype” (in the parlance of UML/SysML), or a probability element or attribute, should be included in an expanded UPDM meta-model, as its consideration would support non-deterministic process controls and token generation. SysML specifically addresses this consideration by introducing probability into activities as “the probability stereotype” -- which may modify both edges and parameter sets, and by extension own “behaviors or operations” (read actions, as part of activities). This stereotype can govern the probability of a given path being taken as an output to a decision node, or the likelihood that values will be output on a parameter set (OMG, 2006). A probability element should be able to support the specification of Probability Distribution Functions (PDFs) across a variety of distribution types, such as Normal, Logarithmic, Weibull, etc. (2001).

**Control Operator** is another type of Activity Control Element that was introduced in SysML. A ControlOperator is a behavior that is intended to represent a complex logical operator that can enable or disable other actions. This kind of control is reminiscent of the mechanism ICOM arrow in IDEF0, and it affords greater specificity in terms of functional control. The ControlOperator should be included in Executable Architecture Specifications.

**Pseudostate - State Control** provides a rich set of state transition control elements and is part of the State Machine. State and State transition are parallel events to activity execution. An activity causes a state transition, of either a product or another activity, or node, or resource. Having a broad set of control options for state transition enables the modeler to provide detailed descriptions of the conditions necessary for making a transition from one state to another, which is vital for state oriented modeling. UPDM and DODAF do not include this rich set of controls, and as a consequence lose the ability to specify state transitions at other than a superficial level. State transition is central to CP-net and DEVS formalisms; however, neither specification offers specific control features such as those that are part of pseudo-state or state control. State Transition Control should be included in Executable Architecture Specifications.

**Control Elements Time:** Under the Timing Category, and under the parent element Control Elements Time, very specific timing controls are listed (i.e., Rate Continuous, Rate Discrete, Time Constraint, Duration, Duration Interval, Interval Constraint, Time Event, Time Expression, Duration-Constraint, and Time-Constraint). Detailed, rule-based, and timing modifiers should be included in Executable Architecture Specifications. Time factors are critical for process control, scheduled resource allocation, and schedule development. These are only addressed in general under the formalisms and not at all in UPDM.

**Operational Event-Trace-Sequence (Time)** provides variety of timing and other logical controls (e.g., looping) for detailed control of sequencing. Event Traces or Sequencing with logical and timing control should be included in Executable Architecture Specifications.

**Observation 1 (Modifiers):** A modifier influences or acts upon another element, similar to the way an adverb modifies a verb, or an adjective modifies a noun in language. As an example, the element “Pseudostate”, for state transition control, modifies state transitions. Whereas, it is true that most of the elements are modifiers, one of the observations from the intermediate level is that the elements that are not modifiers are structural elements. For example Node, NodePort, Pins, Needline, Data Characteristics, and Resource Role are structural elements that do not modify other elements, per se; however, an Activity Control Element, a Performed Activity, a Resource, an Event, and an “OperationalSequence” do modify, or act upon other elements.

**Observation 2 (Hybrids):** Hybrid is an element that has primary characteristics of more than one interrogative type. For example Event, which is classified under the How interrogative (i.e., process) is an element that has process, state, rule and product characteristics, and OperationalEventTrace – Sequence has process, and time characteristics. Hybrids can result in ambiguities in ontological relationships, which can lead to difficulties in building clear categories. The hybrid characteristics were determined after the construction of the elemental ontology; although they were subsequently annotated with hybrid characteristics, they are best left in the original interrogative category, because that is their primary characteristic.

**Observation 3 (Component Parts):** Some elements are parts of other elements. The Node Port, for example, is part of the element Node, and a Pin is part of an Activity. The NodePort and Pin elements are useful in describing model compositions, which is a key focus of the DEVS formalism. Both should be part of an Executable Architecture Specification. It was observed that their structure needs detailed description and specification, so that they can be used to support modular coupling at the syntactic level. This would enable structural relationships to be parsed by a computer, so that dynamic models could be automatically generated from static models. Mittal ((2006) addressed this syntactic deficiency idea in his research, where he pointed out the deficiency of DODAF 1.5, at the time; today, this deficiency remains in the newer UPDM.

In summary, this list of elements has addressed the potential set based on an operational or process modeling delimiting perspective. As a mitigating argument to the question of sufficiency of operational process modeling elements, there is a reasonable probability that, if there are other required elements, they are outside of the nine information and knowledge interrogatives listed in Table 42. As for whether all required elements within the categories are covered, it is suggested that on the basis of data triangulation from numerous well established modeling languages, which included comparison to the formalisms (albeit high level), it is likely that the principle elements have been addressed; the possibility that there are others cannot be excluded. However, because the methodology was holistic in addressing the information and knowledge set interrogatives, and because the method used data triangulation to focus the target data sets from a variety of well-established languages, it is likely that a complete set of potential elements have been defined.

It is clear that the static and dynamic modeling elements that make up the minimal set needed for simulation are present in the EAS-I, as validated by the formalisms. Further, it is clear that there is greater specificity of element descriptions in the EAS-I, than is described in the formalisms, which by comparison are minimalistic or reductionist. That greater specificity is important to driving executable architecture viability with sufficient detail of modeling control, with respect to process, rule, and timing considerations. As such, it may be concluded that the EAS-I is holistic with respect to the dynamic modeling constructs that can support the development of integrated Executable Architectures. With respect to the other elemental constructs that have their origins in the Architecture Framework, sufficiency should be considered domain specific, and holism with respect to EA can be inferred based on an integrated dynamic-static construct (represented in a semantically and syntactically correct meta-model), in the context of the nine interrogatives: five of which are predominately static constructs, and four of which are dynamic.

#### 4.6 Meta-model Use Case

Figure 44 is a meta-model Use-Case designed to provide semantic and syntactic validation against the simple graphically depicted use case shown in the lower right hand corner. The Use Case starts with the firing of a token from an Event Timer in a Node that goes to an Action, which is subsequently processed in accordance with the sequence of activities listed on the following page. For each event, the relevant element in the meta-model is highlighted, and related element -to-element relationships are checked.

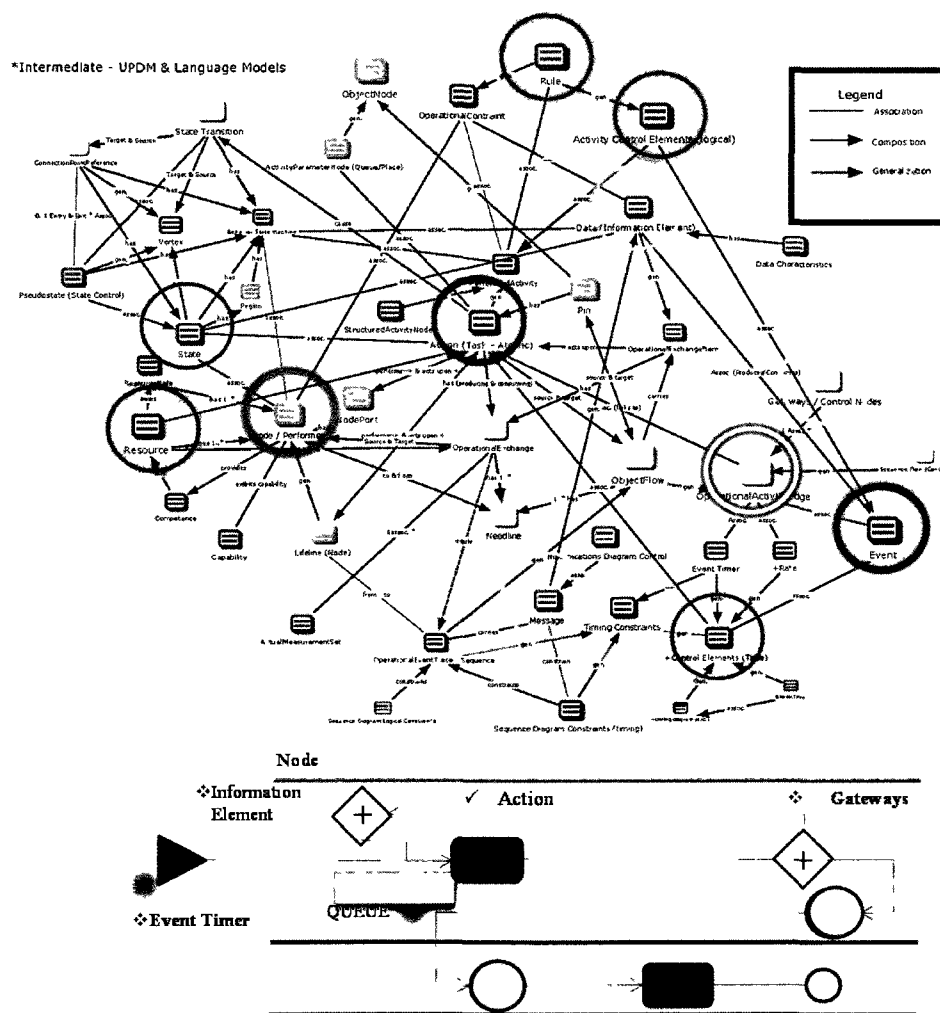


Figure 44 - Meta-model Use Case

The steps in the use case are shown in the following text:

1. **Starting at Node 1**
2. **Event Timer Produces Token (IIR, Const., 10 Sec, for 10 minutes), w/ attributes “x”, Node1**
3. **Token generated**
4. **Event Timer calls Control Elements Time**
5. **Token Traverses OperationalActivityEdge**
6. **Token arrives at Action**
7. **Activity has a resource**
8. **Activity governed by Rule, based resource**
9. **Activity Control Element directs Stochastic behavior**
10. **Normal Distribution PDF (2 minute mean)**
11. **Activity Fires**
12. **Token arrives at Gateway (Decision)**
13. **Token Traverses Edge**
14. **Token Arrives at Message Event (Message generated)**
15. **Message Traverses Edge**
16. **Node 2**
17. **Message arrives at Message Event (token generated)**
18. **Token passes along OperationalActivityEdge**
19. **Token processed by activity (Const. 30 sec.)**
20. **Token State changed to Processed**
21. **Token passes along OperationalActivityEdge**
22. **End Event (Token Consumed)**

Follow-on work could include a series of Use Cases for meta-model validation purposes. This kind of validation would ensure meta-model resiliency and utility.

#### **4.7 EAS – Interrogative Meta-model**

It is possible to define high level theoretical relationships for the nine interrogatives, in terms of a meta-model, as shown in Figure 45. This model was constructed by reducing the intermediate level meta-model down to the nine interrogatives and accounting for child relationships by rolling them up into the parent node. The hybrid category lacks specificity by definition because it is a combination of interrogative types; it requires child nodes to have meaning.



relationships, and understanding complex query design against such tables. For example, a resource at a node, performing a process with certain measurable attributes using a rule based on some timing criteria could be the basis for a query against the supporting data structures. Similarly, an activity in a given state that produces a product could be a logical association of data which would have meaning in terms of a query against the data structures.

#### **4.8 Chapter 4 Conclusions**

In conclusion, the study has produced several meta-models with varying degrees of specificity. There are tradeoffs between greater levels of detail in low level, high specificity models such as the EAS, and the ability to see the key relationships and elements in more simplified, high level models, such as the EAS-Interrogative.

For example, the simplicity of the EAS – Interrogative model conveys some general information about how a rule can influence process behavior, but because of the high level of abstraction, there is no visibility into the kinds of rules that could be used to specify detailed process constraints. At a lower level of specificity, however, such as that which is available in the EAS, we could explore the usefulness of this element more fully. This suggests that a spectrum of meta-model specificity is useful in framing and answering questions derived from theory.

## CHAPTER 5

### CONCLUSIONS AND RECOMMENDATIONS

A review of the literature revealed that all researchers provided valuable solution-specific demonstrations of translations from static to dynamic modeling and also showed the value derived from such an endeavor. These investigations were valuable; however, no common theory underlying these applications can be found in the literature. In addition, no one has attempted to conduct a holistic investigation into the theoretical elements of executable architectures (dynamic models). This is the gap in the body of knowledge which was addressed in this dissertation study.

The purpose of the study was to conduct a holistic investigation into the elements of executable architectures, by means of a qualitative investigative study, utilizing and further exploring a theoretical framework for inquiry into the dimensions of executable architectures. This research began by using inductive reasoning to drive development of the Executable Architecture Concept Triangle (EACT), which is a conceptual framework that was leveraged to design a method for development of the EAS. Use of the framework and method led to deductive reasoning insights with regard to the potential elements of Executable Architectures. The conceptual framework, the EACT, suggests -- and the derived method for building the EAS employs -- data triangulation and thick description to drive elemental convergence in the EAS.

The method employs precision in coding, revealing language element potential contributions to the reference Architecture Framework (UPDM) with respect to Executable Architectures, in the context of validation against M&S Formalisms. (Executable Architecture descriptions require lower level, modeling specific elemental descriptions, whereas, in M&S Formalisms, elemental semantics and syntax are by definition very high level and more general.)

This approach demonstrates that a coding-based, qualitative study is useful in exploring modeling language areas where the data is complex and theory is not well established. This approach further demonstrates that meta-model-based methods can provide a context in which lower level, specific elemental descriptions and relationships can be explored.

The following main contributions have been realized: a refined theoretical framework and method for analysis and development of architecture frameworks in accordance with the objectives for Executable Architectures; the utilization of the theoretical framework resulting in a description of the theoretical elements and their relationships.

The investigation into the Elements of Executable Architectures has produced the following five research results:

1. A well-defined conceptual framework, the Executable Architecture Concept Triangle (EACT), that lends itself to the exploration and development of a method (described in Chapter 4) for derivation of an executable architecture meta-model;
2. The development of a richly detailed meta-model, Executable Architecture Specification (EAS); the result is a composite meta-model for executable architecture, based on architecture elements from the UPDM architecture framework, and drawing from Modeling Language contributions from UML, SysML, BPMN and IDEF, and validated in comparison to M&S formalisms;
3. The development of a detailed Executable Architecture Specification Ontology leveraged to refine the EAS (above), which is an expansion of the six information and knowledge interrogatives to nine;
4. An intermediate-level meta-model Executable Architecture Specification – Intermediate (EAS-I), used to investigate the essential elements of Executable Architecture, that incorporates the static and dynamic elements;
5. An interrogative meta-model that shows the relationships between the nine interrogatives, potentially useful at the abstract, theoretical level.

## 5.1 Synopsis of Research Results

This section provides a brief discussion of the five main research results above:

1. The research produced the Executable Architecture Concept Triangle (EACT, Figure 46), which was further refined over time to an extended version. This extended version is more complete, revealing annotated relationship lines; it better describes the Executable Architecture Specification (EAS) core component; which more clearly reflects the structure based on 9 interrogatives and their

syntactic relationships. The extended version of the EACT provided more clarity in building the EAS, and for deriving a method for development of the EAS.

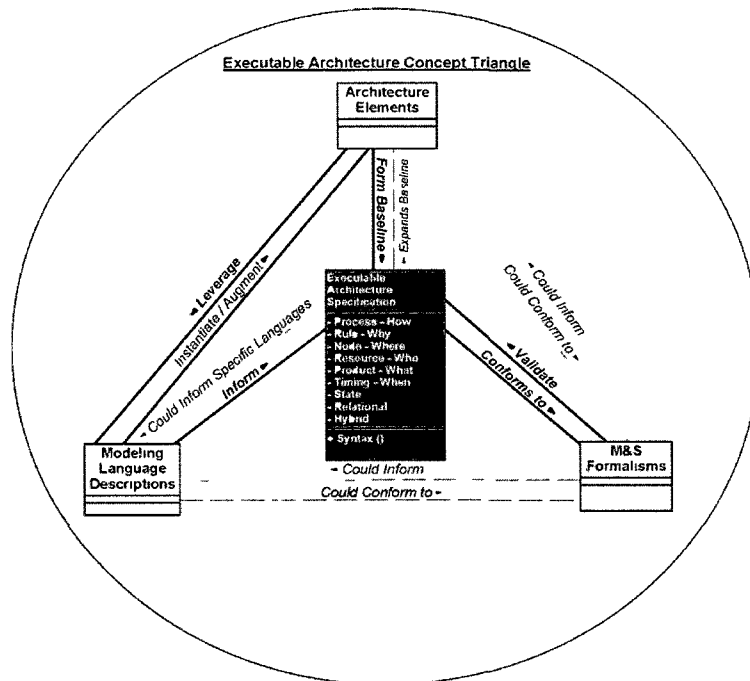


Figure 46 - EACT Summary

2. The research produced a detailed meta-model for Executable Architectures, referred to as an *Executable Architecture Specification (EAS)*, Figure 47) Each element in the meta-model is color coded to reflect the nine interrogative types. It is further comprised of UML generalization, composition, and association relationships between elements, shown as annotated lines and arrows. The meta-model is based on architecture elements and relationships derived from two sources -- the Unified Profile for DODAF and MODAF (UPDM) architecture framework, and key Modeling Languages (UML, SysML, BPMN and IDEF) -- and validated against the M&S Formalisms (CP-net, DEVS).

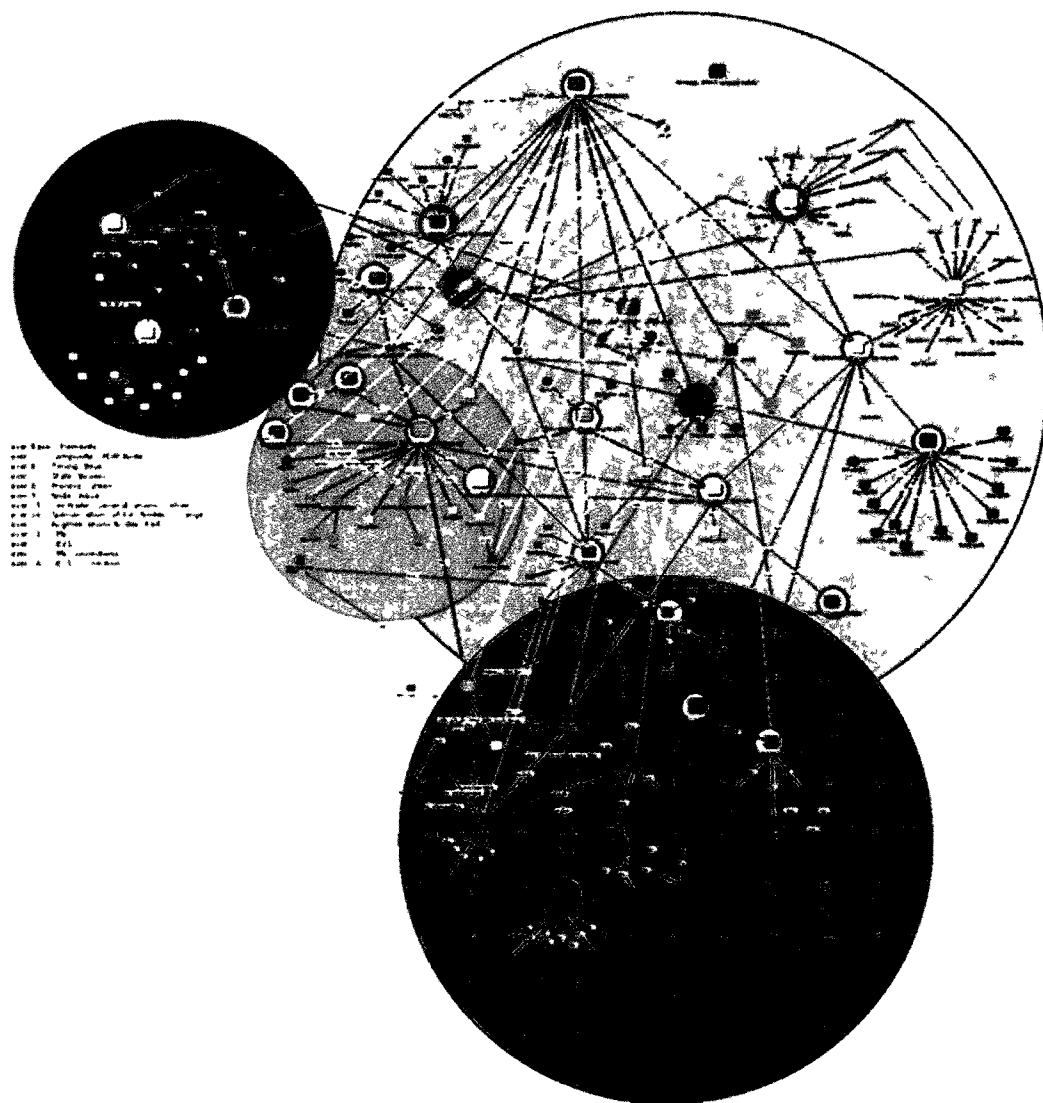
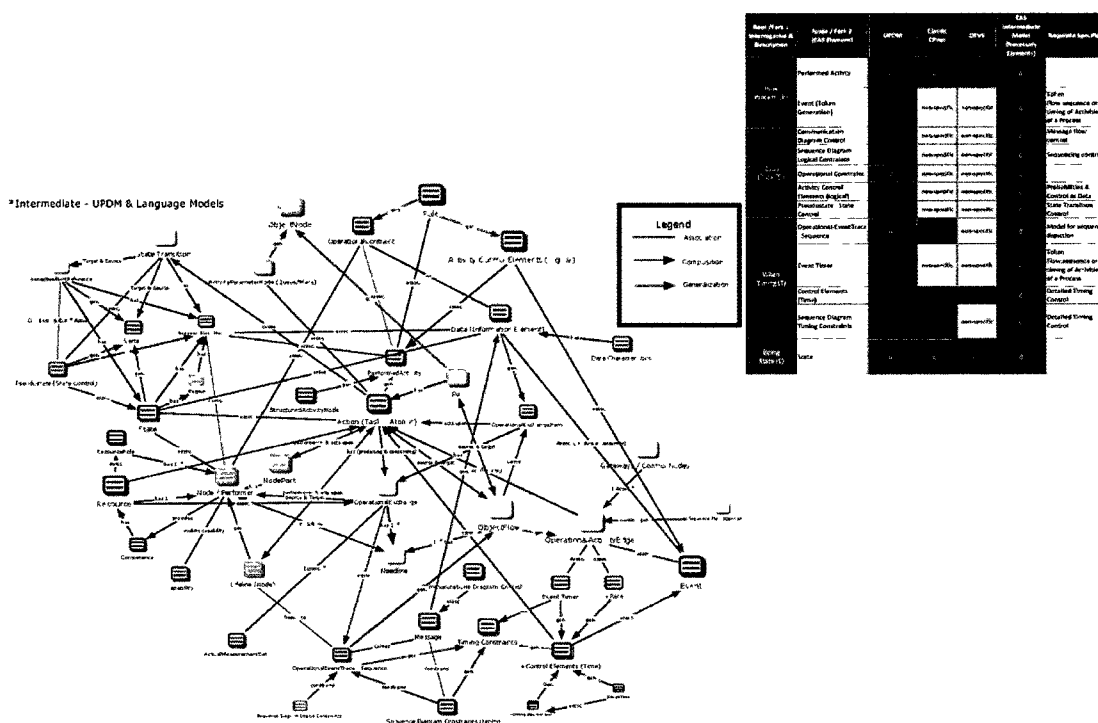


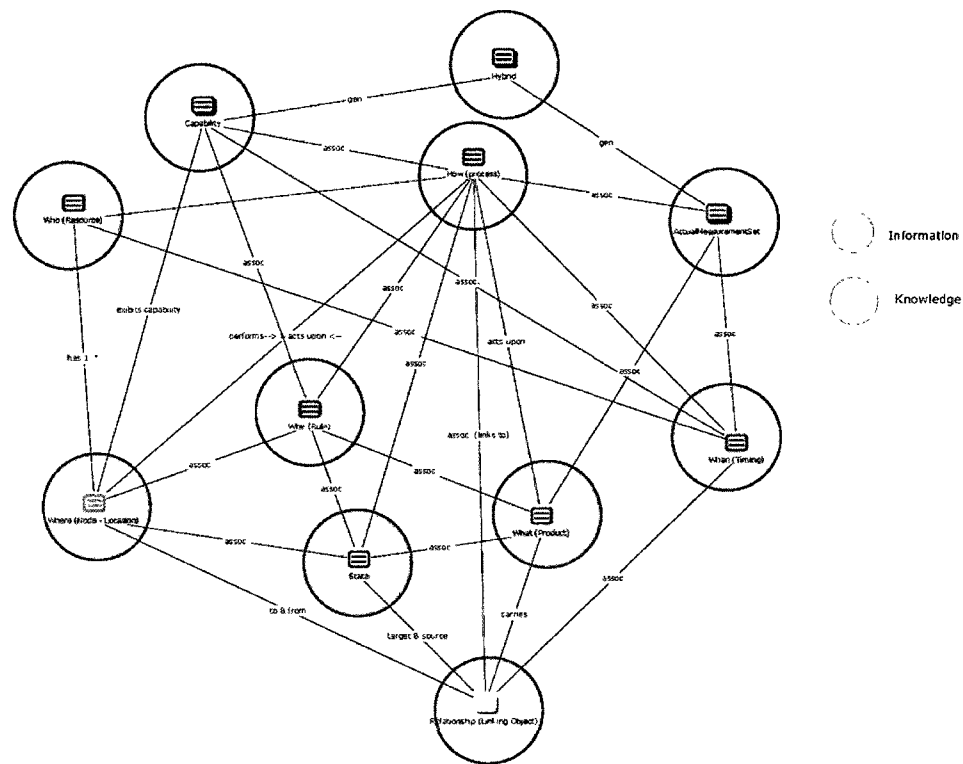
Figure 47 - EAS Summary

3. The research produced a detailed EAS ontology which was derived from the foundational EAS meta-model, and which was used to refine the EAS meta-model. The ontology is a taxonomy of elements that is based on the nine interrogatives used throughout the investigation, and which contains composition and generalization relationships from each interrogative root to child level specifications (see Process Element Node Tree in Figure 48). The six information and knowledge interrogatives, What, Who, Where, When, How, and Why,





5. A meta-model based on the nine interrogative elements, the Executable Architecture Specification – Interrogative (EAS – Interrogative, Figure 50), was derived from the EAS Intermediate-level Meta-model. This meta-model was developed by trimming secondary level detail from the EAS-I. It is highly generalized, but shows key relationships between the interrogatives. High level abstraction meta-models can be used as an aid to understanding generalized relationships in real-world data model implementations without the distraction of detail.



**Figure 50 - EAS - Interrogative Summary**

## 5.2 Potential Elements of Executable Architectures (EA)

In Table 45 and Table 46, the set of 30 potential elements of Executable Architecture are provided in alignment with their interrogative categories, with descriptions and notes about why each is important. These elements were discussed in depth in Chapter 4 and are part of the EAS-I meta-model.

Table 45 - Static Elements of EA

Type	Root /Fork 1 Interrogative & Description	#	Node / Fork 2 (EAS Element)	Description	Why Important
Static	Where Node (N)	1	Node	A Node is an element of the operational architecture that produces, consumes, or processes information	Locus of activity
		2	Pins	A pin is an element and multiplicity element that provides values to actions and accepts result values from them.	Activity Connection specificity
		3	Node Port	A port is a property of a Node that specifies a distinct interaction point between the node and its environment or between the (behavior of the) node and its internal parts.	Node Connection specificity
		4	Gate	A Gate is a connection point for relating a Message outside an InteractionFragment with a Message inside the InteractionFragment. Sequence Diagram Connectivity	Sequence Diagram Connectivity
		5	Queues	Activity parameter nodes are object nodes at the beginning and end of flows that provide a means to accept inputs to an activity and provide outputs from the activity, through the activity parameters.	Manage Token arrival. Queue ordering (FIFO, LIFO)
	Who Resource (R)	6	Resource	OrganisationalResource or FunctionalResource that can contribute towards fulfilling a capability. The Resource is used to specify resources that can be referenced by Activities.	Resource executes activity at a Node. Affects performance
		7	ResourceRole	Defines the resource that will perform or will be responsible for the Activity. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.	Describes resource
		8	Competence	A specific set of abilities defined by knowledge, skills and attitude.	Performance measures
	What Product (P)	9	Operational Exchange Item	An abstract utility element used as common ancestor for: InformationElement, ResourceArtifact, Energy, OrganizationalResource	Generalization for exchange types
		10	Information Element (Data)	A relationship specifying the need to exchange information between nodes	Produced by activity or event, has attributes, i.e., a token
		11	Data Characteristics - Attributes	Data properties, structure	Specifies properties, structure of data/token
	Relationship (C)	12	Gateways & Control Nodes	Gateways are used to control how the Process flows (how Tokens flow) through Sequence Flows as they converge and diverge within a Process.	Logical Control, flow control
		13	Operational Activity Edge	UPDM An extension of «ActivityEdge» that is used to model the flow of control/objects through an OperationalActivity. An OperationalActivityEdge (MODAF::OperationalActivityFlow) is a flow of information, energy or materiel from one activity to another. An activity edge is an abstract class for directed connections between two activities	Provides connectivity: Edge, connector, sequence and data flows
		14	Operational Exchange	Abstract element. An abstract utility element used as common ancestor for: InformationExchange, OrganizationalExchange, EnergyExchange, MaterielExchange An operational exchange is formed when an activity of one operational node consumes items produced by another activity of a different operational node.	Data element produced by an activity at a node, by a resource: hybrid characteristics
		15	Needline	A needline documents the requirement to exchange information between nodes. The needline does not indicate how the information transfer is implemented.	Role-up of information exchanges
		16	State Transition	A transition is a directed relationship between a source vertex and a target vertex. It may be part of a compound transition, which takes the state machine from one state configuration to another, representing the complete response of the state machine to an occurrence of an event of a particular type.	change of state
	Hybrid (H)	17	ActualMeasurement-Set	A set or collection of ActualMeasurement(s): Measurements: Accountability, Interoperability Level Achievable, Classification, Classification Caveat, Criticality, Periodicity, Protection Duration, Protection Suspense Calendar Date, Protection Type Name Timeliness, Transaction Type, Protection Duration Code, Releasability, Size, Throughput.	measures
		18	Capability	A Capability is a high-level specification of an ability or capacity which achieves specific objectives.	High level system description: SE utility

Table 46 - Dynamic Elements EA

Type	Root /Fork 1 Interrogative & Description	#	Node / Fork 2 (EAS Element)	Description	Why Important
Dynamic	How Process (P)	19	Performed Activity	An abstract element that represents a behavior (i.e. a Function or OperationalActivity) that can be performed by a Performer.	Basic unit of behavior
		20	Event (Token Generation)	Events An Event is something that "happens" during the course of a Process. These Events affect the flow of the Process and usually have a cause or an impact.	Token Flow,sequence or timing of Activities of a Process
	Why Rule (M)	21	Communication Diagram Control	Communications diagram control logical controls (e.g., sequence, guard, iteration, etc.)	Message flow control
		22	Sequence Diagram Logical Constraints	Logical control over event traces / sequences (e.g., loop, sequence, parallel)	Sequencing control
		23	Operational Constraint	Generalization element for rules, scope, contex, expressions	Operatioanl Constraints
		24	Activity Control Elements (logical)	Logical control over behavior/activites such as Probabilities & Control as Data	Probabilities & Control as Data
		25	Pseudostate - State Control	State Transition Control	State Transition Control
	When Timing (T)	26	Operational-EventTrace - Sequence (Time)	Timing notations that may be applied to describe time observation and timing constraints, with respect to sequence diagrams	Model for sequence depiction
		27	Event Timer	Token Flow,sequence or timing of Activities of a Process	Token Flow,sequence or timing of Activities of a Process
		28	Control Elements (Time)	Detailed Timing Control	Detailed Timing Control
		29	Sequence Diagram Timing Constraints	Detailed Timing Control, for the sequence diagram	Detailed Timing Control
	Being State (S)	30	State	A state models a situation during which some (usually implicit) invariant condition holds. The invariant may represent a static situation such as an object waiting for some external event to occur. However, it can also model dynamic conditions such as the process of performing some behavior (i.e., the model element under consideration enters the state when the behavior commences and leaves it as soon as the behavior is completed).	Condition

### 5.3 Recommendations

For quality assurance (QA) purposes it is recommended as a follow-on activity to develop a series of use cases, similar in method to the use case explained in section 4.6, for model validation. Feasibility and usefulness of such an effort have been shown in this thesis.

It is also recommended to allocate EAS elements back to a set of revised UPDM models, from the UPDM-Language composite model. This could be facilitated by use of data attributes and query sets in MAXQDA.

In addition, this method offers good traceability with support for detailed composite model development and the ability to cross reference data elements. In addition, the linkage between data objects and visual modeling methods is good. However, because of tool limitations (in that MAXQDA does not support UML compliant modeling), it would be better to implement these models in a UML compliant modeling tool supporting XMI Metadata Interchange (XMI), in order to instantiate these models as physical schemas. By putting these models into a UML Class Diagram, using appropriate relational modeling constructs, it should be possible to produce an XML Metadata Interchange (XMI) serialization of the models. Such a serialization could be used in the generation of the Data Definition Language (DDL) needed for the development of physical data models, data structures, and databases supporting the instantiation of the executable architecture constructs into real database and tool implementations.

Finally, there are tools (e.g., Torque) that could be used to support the transformation of a UML / XMI compliant Class diagrams into DDL. As a practical, follow-on research endeavor and engineering task, it would be valuable to explore the use instantiation of the meta-model as a basis for executable architecture tool exploration and development.

### 5.4 Over-specification Concerns

The Executable Architecture concept is designed to enable additional systems engineering capability. The purpose of the EAS is to build Executable Architecture. Inclusion of process simulation capability in the EAS and in subsequent Frameworks and

tools based on these frameworks should be viewed generally as a multi-level specification capability rather than narrowly as prescriptive. Systems engineering is often approached on a number of levels of modeling specificity, depending on the maturity and stage of the project at hand. Having a meta-model that enables simulation capability should enable object re-use within a project database, as additional complexity in modeling and simulation is required. Furthermore, the inclusion of simulation capabilities in an architecture framework should not require a higher level of general training for the modeling team. As is generally the case today, a variety of experience, from novice architect to simulation engineer can be expected. One of the problems with architecture today is that it is treated as a one size fits all endeavor, rather than as a multi-faceted set of methods and tools and approaches which are the means to good systems engineering. With this in mind, Executable Architecture should be viewed as an additional enabler in a spectrum of integrated modeling and simulation capabilities.

## **5.5 Significance of Study**

This method is extensible to other architecture frameworks, and other language instantiations, as well as other formalisms. With this approach, the key would be to put boundaries on the problem space up front so that the baseline draws from candidate models and formalisms that are relevant to the problem space and desired outcome. In this study, the upfront assumptions were that the focus of the research would be on process modeling, both static and dynamic. Furthermore, the investigation was focused on UPDM for both reasons of practicality (the strength of the starting meta-model) and utility (UPDM is based on DODAF and MODAF, broadly used in the United States Department of Defense and the UK Ministry of Defense). The method also allows for comparisons of similar Architecture Frameworks, such as DODAF 1.5 and DODAF 2.0.

### **5.5.1 *Practical Implementations and Significance***

The study may be informative with respect to the design of future DODAF-like meta-models that include dynamic modeling. Findings may have implications for the development of future modeling tools. The conceptual framework and method may be useful for the evaluation of other architecture frameworks in future studies. There are a

number of potential practical applications for both the method and the results of this investigation.

### 5.5.2 *DODAF 3.0*

The composite meta-model that was developed in the process of exploring this methodology was focused on the operational architecture models. The next major revision to DODAF (DODAF 3.0), MODAF, or UPDM could use both the resultant meta-model of this study and the method. Other military frameworks such as the Canadian Department of National Defence Architecture Framework (DND AF) and NATO Architecture Framework (NAF) could leverage the meta-model developed here and / or the method. Beyond the military domain, this method should be extensible to other architecture such as TOGAF (Open-Group, 2009), which is an industry standard architecture framework. To build a new executable architecture framework, a holistically derived series of model-centric meta-models should be developed to support the new construct. If it were designed along executable architecture inclusive lines, the architects of this new meta-model could take advantage of the composite operational meta-model that was developed here. That meta-model could provide insights into the operational models associated with that future architecture framework.

This investigation only partially explored the systems side of UPDM elements. Systems level objects were coded using in-vivo coding methods, and arranged ontologically based on a first cut assessment in MAXQDA. They were not subsequently modeled graphically to provide that follow-on level of elemental relational investigation, because it was not deemed necessary for the exploration of the method. Because of the intentionally designed operational-systems dichotomy in DODAF (Ring, et al., 2008) (and related frameworks such as UPDM), there is extensive parallelism between systems and operational elemental constructs (e.g., an operational process or activity parallels a system function, and so forth). As such, it stands to reason that with parallelism in elements, it may be inferred that there would be not be obstacles to the application of this method to systems elements and modeling constructs.

### 5.5.3 *SysML (Next Generation)*

Another practical usage of the results and method discussed here could be in a revision to SysML. SysML process models were explored extensively in this investigation. SysML has several, but not all of the elements described in the composite UPDM-Language meta-model. It might be interesting to explore the expansion of SysML in ways that would support simulation modeling of processes through an expanded SysML.

The inclusion of simulation capability could broadly include basic discrete event modeling elements and constructs, which would apply to both general process modeling and systems process modeling. Beyond that, the method could potentially be extended to continuous modeling methods and physics-based modeling and simulation problem domains.

### 5.5.4 *Tool Mediation*

Lastly, the EAS meta-model and the method for developing it could be used to spin off holistic executable architecture-based Modeling and Simulation tool development. There are tools in the market place that support some elements of dynamic modeling such as iGrafx and System Architect. iGrafx supports modeling and simulation of BPMN based models, and System Architect supports simulation of both BPMN and process flow models. Neither, however, supports an integrated architecture-based approach to modeling and simulation. This is probably because executable architectures have not been defined from the meta-model perspective. Apart from that kind of lead from an authoritative developing body, such as DOD or OMG, a specific tool implementation could result in a practical proto-type implementation or proprietary development effort.

## 5.6 Conclusion

In conclusion, this dissertation has successfully explored a method for holistically developing Executable Architecture Specifications, using the Executable Architecture Concept Triangle as a framework for guiding data triangulation. UPDM Architecture Elements, Modeling Languages, and Modeling and Simulation Formalisms were used as a basis for systematic development of a detailed Executable Architecture Specification (EAS), containing detailed semantic and syntactic information. This study has explored

and described the elements of architecture in terms of a set of nine information interrogatives, using this set to build an executable architecture information ontology to describe those elements. Lastly, the EAS meta-model and ontology were utilized to investigate and describe a set of 30 **potential elements** for executable architecture through the EAS-Intermediate meta-model.

## REFERENCES

- Alberts, D. (2002). *Code of Best Practice: Experimentation*. Washington, D.C.: Command and Control Research Program (CCRP).
- Bienvenu, M., Shin, I., & Levis, A. (2000). C4ISR Architectures III: An Object-Oriented Approach for Architecture Design. *Journal of Systems Engineering*, 3(No. 4).
- Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The unified modeling language user guide*. Reading Mass.: Addison-Wesley.
- Buede, D. M. (2009). *The Engineering Design of Systems : Models and Methods*. Hoboken: Wiley.
- CCRP. (2002). *NATO Code of Best Practice for C2 Assessment*. Washington, D.C.: DoD Command and Control Research Program.
- Charmaz, K. (2000). *Grounded theory: Objectivist and constructivist methods*. Thousand Oaks, CA: Sage.
- CJCSI. (2009). *JOINT CAPABILITIES INTEGRATION AND DEVELOPMENT SYSTEM CJCSI 3170.01G*. (CJCSI 3170.01G). Washington, D.C.: Chairman of the Joint Chiefs of Staff.
- Clemson, B. (1984). *Cybernetics : a new management tool*. Tunbridge Wells, Kent: Abacus Press.
- Corbin, J. M., & Strauss, A. L. (2008). *Basics of qualitative research : techniques and procedures for developing grounded theory*. Los Angeles, Calif.: Sage Publications.
- Creswell, J. (2009). *Research design: qualitative, quantitative, and mixed methods approaches*. Thousand Oaks, California: Sage Publications.
- Cutcliffe, J. R. (2000). Methodological issues in grounded theory. *Journal of advanced nursing*, 31, 1476-1484.
- DeMarco, T. (1979). Structured analysis and system specification *Classics in software engineering* (pp. 409-424): Yourdon Press.
- DeMarco, T. (1979). Structured analysis and system specification *Classics in software engineering* (pp. 409-424). Saddle River, NJ: Yourdon Press.
- DOD. (2007a). *DOD Architecture Framework Version 1.5 Volume I: Definitions and Guidelines*. Washington, D.C.: Department of Defense.
- DOD. (2007b). *DOD Architecture Framework Version 1.5 Volume II: Product Descriptions*. Washington, D.C.: Department of Defense.
- DOD. (2009). *DOD Architecture Framework Version 2.0*. Washington, D.C.: Department of Defense.
- Dori, D. (2002). *Object-process methodology: a holistics systems paradigm*. Berlin Heidelberg New York: Springer-Verlag.
- Fishwick, P. A. (1995). *Simulation model design and execution: building digital worlds*. Englewood Cliffs, N.J.: Prentice Hall.
- Flood, R. L., & Carson, E. R. (1993). *Dealing with complexity: an introduction to the theory and application of systems science*. New York and London: Plenum Press.
- Garcia, J. (2011). *Adding Executable Content to Executable Architectures: Endabling an Executable Context Simulation Framework (ECSF)*. PhD, Old Dominion University, Norfolk. (AAT 3459272)

- Glaser, B. G. (1991). *Basics of Grounded Theory Analysis: Emergence vs. Forcing*. Mill Valley, California: Sociology Press.
- IDEF. (2010). Integrated Definition Methods (IDEF) 2010, from <http://www.idef.com>
- IEEE. (Ed.) (1990) IEEE Std 610.12-1990.
- Jensen, K. (1992a). *Coloured Petri Nets Basic Concepts, Analysis Methods and Practical Use* (Second Edition ed. Vol. 1). Berlin Heidelberg New York: Springer-Verlag.
- Jensen, K. (1992). Coloured Petri nets basic concepts, analysis methods, and practical use
- Kelton, D., Sadowski, R. P., & Sadowski, D. A. (2001). *Simulation with Arena*: McGraw-Hill.
- Law, A. M., & Kelton, W. D. (2000). *Simulation modeling and analysis*: McGraw-Hill.
- Leedy, P. D., & Ormrod, J. E. (2010). *Practical Research Planning and Design* (Ninth Edition ed.). Saddle River, New Jersey: Pearson Education, Inc.
- Levis, A., & Wagenhals, L. (2000). C4ISR architectures. I: Developing a process for C4ISR architecture design. *Syst Eng* 3, 225-247.
- Lewins, A., & Silver, C. (2007). *Using software in qualitative research: a step-by-step guide*: SAGE.
- MAXDQA10. (2011). MAXQDA Retrieved Oct 9, 2011, from <http://www.maxqda.com/>
- meta-model. (Ed.) (2011) Object Management Group Terms and Acronyms OMG.
- MindManager. (2011). Mindjet Retrieved October 9, 2011, 2011, from <http://www.mindjet.com/>
- Mittal, S. (2006). Extending DODAF to allow integrated DEVS-based modeling and simulation. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 3(2), 95.
- Mittal, S. (2007). *Devs unified process for integrated development and testing of service oriented architectures*. Ph. D. Dissertation, University of Arizona.
- Mittal, S., Mitra, A., Gupta, A., & Zeigler, B. (2006, 16-18 Sept. 2006). *Strengthening OV-6a Semantics with Rule-Based Meta-models in DEVS/DODAF based Life-cycle Architectures Development*. Paper presented at the Information Reuse and Integration, 2006 IEEE International Conference on.
- Mittal, S., Risco, J., & Zeigler, B. (2007). *DEVS-based simulation web services for net-centric T&E*. Paper presented at the SCSC Proceedings of the 2007 Summer Computer Simulation Conference, Madrid, Spain.
- Mittal, S., Zeigler, B., Risco Martín, J., Sahin, F., & Jamshidi, M. (2008). Modeling and Simulation for Systems of Systems Engineering. In M. Jamshidi (Ed.), *System of Systems – Innovations for the 21st Century*. Hoboken, New Jersey: John Wiley & Sons, Inc.
- Morse, J. M. U. (Ed.). (1994). *Critical issues in qualitative research methods*. London: Sage Publications.
- Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4), 541-580.
- necessary. (Ed.) (2011) Oxford English Dictionary (Third edition, June 2003; online version September 2011 ed.). Oxford University Press.
- OMG. (2003). MDA Guide Version 1.0.1. Needham, MA: Object Management Group (OMG).
- OMG. (2006). SysML Specification Version 1.1 . Needham, MA: Object Management Group (OMG).

- OMG. (2009). Business Process Model and Notation (BPMN) Version 1.2. Needham, MA: Object Management Group (OMG).
- OMG. (2009). Unified Modeling Language (OMG UML) Superstructure, Version 2.2. Needham, MA: Object Management Group (OMG).
- OMG. (2009a). Unified Profile for Department of Defense Architecture Framework (DODAF) and the Ministry of Defense Architecture Framework (MODAF) Needham, MA: Object Management Group (OMG).
- OMG. (2011). Business Process Model and Notation (BPMN) Version 2.0. Needham, MA: Object Management Group (OMG).
- Open-Group. (2009). TOGAF Version 9.0. Reading Berkshire, RG1 1AX United Kingdom.
- Pawlowski III, T., Barr, P., Ring, S., Vitkevich, J., Leach, M., & Segarra, S. (2004). *Executable Architecture Methodology for Analysis, FY04 Final Report*. MITRE, Washington C3 Center. McLean, Virginia.
- Peirce, C. S. (1998). *The Essential Peirce* (Vol. 2). Bloomington, IN: Indiana University Press.
- Petri, C. (1962). *Kommunikation mit Automaten*. Institut für Instrumentelle Mathematik, Bonn. Vol. 1 Suppl. 1, database. (Schriften des IIM Nr. 2)
- Petty, M. D., McKenzie, F. D., & Qingwen, X. (2002, 2002). *Using a software architecture description language to model the architecture and run-time performance of a federate*. Paper presented at the Distributed Simulation and Real-Time Applications, 2002. Proceedings. Sixth IEEE International Workshop on.
- Renzhong, W., & Dagli, C. H. (2008, 7-10 April 2008). *An Executable System Architecture Approach to Discrete Events System Modeling Using SysML in Conjunction with Colored Petri Net*. Paper presented at the Systems Conference, 2008 2nd Annual IEEE.
- Ring, S. J., Nicholson, D., & S, P. (2008). Activity-Based Methodology for Development and Analysis of Integrated DOD Architectures. In S. Pallab (Ed.), *Handbook of Enterprise Systems Architecture in Practice* (pp. 85-113): Information Science Reference.
- Risco-Martin, J., De La Cruz, J., Mittal, S., & Zeigler, B. (2009). eUDEVS: Executable UML with DEVS Theory of Modeling and Simulation. *Simulation*, 85(11-12), 750-777. doi: 10.1177/0037549709104727
- Sage, A. P., & Rouse, W. B. (2009). *Handbook of Systems Engineering and Management*. Hoboken, NJ: John Wiley & Sons.
- Shaw, M., & Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*: Prentice Hall.
- Sheehan, J., Deitz, P., Bray, B., Harris, B., & Wong, A. (2003). *The Military Missions and Means Framework*. Paper presented at the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).
- Shuman, E. (2010). *Understanding Executable Architectures Through An Examination of Language Model Elements*. Paper presented at the Proceedings of the 2010 Summer Simulation Conference, Ottawa, CA.

- Tolk, A., Garcia, J., & Shuman, E. (2010). *Executable Architecture Research at Old Dominion University*. Virginia Beach, VA.
- W3C. (2004). Web Services Architecture W3C Working Group Note 11 February 2004.
- Wagenhals, L., Haider, S., & Levis, A. (2002). *Synthesizing executable models of object oriented architectures*. Paper presented at the Proceedings of the conference on Application and theory of petri nets: formal methods in software engineering and defence systems - Volume 12, Adelaide, Australia.
- Wagenhals, L., Shin, I., Kim, D., & Levis, A. (2000). C4ISR architectures: II. A structured analysis approach for architecture design. *Systems Engineering*, 3(4), 248-287.
- Zachman, J. A. (1999). A framework for information systems architecture. *IBM Systems Journal*, 38(2.3), 454-470.
- Zeigler, B., & Mittal, S. (2005, 10-12 Oct. 2005). *Enhancing DODAF with a DEVS-based system lifecycle development process*. Paper presented at the Systems, Man and Cybernetics, 2005 IEEE International Conference on.
- Zeigler, B., & Mittal, S. (2006). *Enhancing DODAF with a DEVS-based system lifecycle development process*.
- Zeigler, B., Praehofer, H., & Kim, T. (2000). *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. San Diego, CA: Academic Press.
- Zinn, A. (2004). *The Use of Integrated Architectures to Support Agent Based Simulation An Initial Investigation*. Master's Thesis, Air Force Institute of Technology.

## APPENDIXES

### A. Element Comparison Tables

#### Element Comparison Tables

Table 47 - Activity Comparison Table

Group: Function (F) Subgroup: PerformedActivity (pa)	BPMN	UML Act	SysML Act	IDEF0	OV-5	OV-6a	Count	Comparison Classification	Comment
<b>BPMN Process &amp; Collab</b>									
Activities	x						1	IE	
Task (Atomic)	x						2	IE	
Human Interaction	x							SEC (PerformedActivity)	
Sub-Process	x						3	IE	
Nested/Embedded SubProcess	x						4	IE	
Expanded Sub-Process	x						5	IE	
Collapsed Sub-Process	x						6	IE	
Transaction	x						7	IE	
<b>IDEF 0</b>									
Function				x				SEC (PerformedActivity)	
<b>OV-5</b>									
PerformedActivity					x		8	IE	Parent to OperationalActivity and
OperationalActivityAction					x			SEC (PerformedActivity)	
OperationalActivity					x			SEC (PerformedActivity)	
StandardOperationalActivity					x			SEC (PerformedActivity)	
<b>OV-6a</b>									
OperationalActivity						x		D (OperationalActivity)	
<b>SysML &amp; UML Activity</b>									
Action		x	x					D (Task)	
StructuredActivityNode		x	x				9	IX (Sub-Process)	
ConditionalNode		x	x				10	IE	
ExpansionRegion		x	x				11	IE	
LoopNode		x	x				12	IE	
SequenceNode		x	x				13	IE	
<b>OV-4</b>									UPDM ref to function here
Function		x	x				14	IE	
							<b>Elemental Comparative Classification</b>		
							x	<b>Individual Elements (IE)</b>	
							x	<b>Duplicate (D)</b>	
							x	<b>Same Equivalent Class (SEC)</b>	
							x	<b>Individual Extension (IX)</b>	

Table 48 - Product Comparison Group

Code: Product: Q	BPMN	UML Act	SysML Act	Timing Diagram	IDEF0	OV-5	OV-6a	Count	Comment
<b>SysML &amp; UML Activity</b>									
Parameter		x	x					1	IE
ParameterSet		x	x					2	IE
<b>BPMN Process &amp; Collab</b>									
Data									
Data Objects	x							3	IE
Data Object References	x							4	IE
Data Stores	x							5	IE
Message	x							6	IE
<b>OV-5</b>									
OperationalParameter						x			D (OperationalExchangeItem)
OperationalExchangeItem						x		7	IE
InformationElement						x		8	IX
OrganizationalResource						x		9	IX
Energy						x		10	IX
ResourceArtifact						x		11	IX
<b>OV-6a</b>									
InformationElement							x		D (Information Element)
<b>OV-7</b>									
InformationElement						x			D (Information Element)
EntityItem						x		13	IE
EntityAttribute						x		12	IE
EntityRelationship						x		14	IE
DataModel									
<b>BPMN Process &amp; Collab</b>									
Data Characteristics									
Data Structure ("ItemDefinition")	x							13	IE
DataState	x							14	IE
								<b>Elemental Comparative Classification</b>	
								x	Individual Elements (IE)
								x	Duplicate (I)
								x	Same Equivalent Class (SEC)
								x	Individual Extension (IX)

Table 49 - Rule Comparison Group

Group: Rule (M)	BPMN	UML Act	SysML Act	IDEF0	OV-5	OV-6a	Count	Comparison Classification	Comment
<b>UML Act</b>									
BehavioralFeature		x	x				1	IE	possible implementation
LocalPreandPostConditions		x	x				2	IE	possible implementation
ExpansionKind (Expansion Region control)		x	x				3	IE	possible implementation
<b>BPMN Process &amp; Collab</b>									
Rule	x						4	IE	
Scope	x							SEC (Context)	
Expressions	x						5	IE	
<b>IDEF0</b>									
Viewpoint				x			6	IE	
Context				x			7	IE	
<b>OV-6a</b>									
Mission						x	8	IE	
OperationalConstraint						x	9	IE	
<b>SysML Activity</b>									
(+)ControlOperator			x				10	IE	possible implementation
LocalPreandPostConditions			x					D (LocalPreand PostConditions)	
ExpansionKind (ExpansionRegion control)			x					D (LocalPreand PostConditions)	
(+)Probability (edge control)			x				11	IE	
<b>Parameter Diagram</b>									
Constraint Block			x				12	IE	
								<b>Elemental Comparative Classification</b>	
								x	Individual Elements (IE)
								x	Duplicate (D)
								x	Same Equivalent Class (SEC)
								x	Individual Extension (IX)

Table 50 - Time Comparison Group

Group: Time (T)	Code (BPMN)	Code (UML Act)	Code (SysML Act)	Timing Diagram	Code (IDEF0)	Code (OV-5)	Code (OV-6a)	Count	Comparison Classification	Comment
<b>SysML Activity</b>										
+Timing constraint notes (ext.)										
+Rate			x					1	IE	
+Discrete (ext.)			x					2	IE	
+Continuous (ext.)			x					3	IE	
+Timing Diagram (ext.)										
Time constraint				x				4	IE	
Duration Constraint				x				5	IE	D Event Timer TimeDuration
DestructionEvent				x				6	IE	
GeneralOrdering				x				7	IE	
<b>BPMN Process &amp; Collab</b>										
Event Timer		x	x					8	IX	IX Time Constraint
Event Timer timeDate	x							9	IX	
Event Timer timeCycle	x							10	IX	
Event Timer timeDuration	x							11	IX	
TimeEvent										
									<b>Elemental Comparative Classification</b>	
									x	Individual Elements (IE)
									x	Duplicate (D)
									x	Same Equivalent Class (SEC)
									x	Individual Extension (IX)

Table 51 - Control Node Comparison Group

Subgroup: Relational: C	BPMN	UML Act	SysML	IDEF0	OV-5	OV-6a	OV-2	OV-3	Count	Comparison Classification
<b>SysML &amp; UML</b>										
<b>ControlNode (Gateway) c1</b>		x	x						0	IE
DecisionNode		x	x						1	IE
ForkNode	x	x	x						2	IE
InitialNode	x	x	x						3	IE
JoinNode	x	x	x						4	IE
MergeNode	x	x	x						5	IE
FinalNode		x	x						6	IE
FlowFinal		x	x						7	IE
ActivityFinal		x	x						8	IE
<b>BPMN</b>										
<b>Gateways</b>	x								9	IE
Complex	x			x					10	IE
Event Based	x			x					11	IE
Exclusive	x								12	IE
Inclusive	x								13	IE
Parallel	x								14	IE
Parallel Event-based	x								15	IE
<b>Activity Edge: c2</b>	x				x				16	IE
ControlFlow (Sequence Flow)	x	x	x	x	x				17	IE
ObjectFlow	x	x	x	x	x				18	IE
<b>Sequence Flow (Control Flow)</b>	x								19	IE
Merging	x								20	IE
Looping	x								21	IE
Fork	x			x					22	IE
Join	x			x					23	IE
Normal Flow	x								24	D (ControlFlow)
Conditional flow	x								25	IE
Default flow	x								26	IE
Exception flow	x								27	IE
Compensation Association	x								28	IE
Uncontrolled flow	x								29	IE
Data Flow	x								30	D to ObjectFlow
Message Flow	x								31	IX to ObjectFlow - add
Data Associations	x								32	IX to ObjectFlow - add
<b>IDEF</b>										
<b>(Sequence) Flows</b>										
Arrow (Flow)				x					33	D to Control Flow
Arrow Segment				x					34	IE
Boundary Arrow				x					35	IE
Branch										
Fork				x					36	D Fork
Join				x					37	D Join
Bundling/Unbundling				x					38	IE
Control Arrow				x					39	IE
Input Arrow				x					40	IX ActivityEdge, ControlFlow
Output Arrow				x					41	IX ActivityEdge, ControlFlow
Internal Arrow				x					42	IX ActivityEdge, ControlFlow
Mechanism Arrow				x					43	IE
Tunnelled Arrow				x					44	IE
<b>OV-5</b>										
<b>Edge (Sequence Flow)</b>										
OperationalActivityEdge					x				45	Identical to Activity Edge
OperationalExchange c3					x				46	1 OperationalExchange (below)
<b>OV-2</b>										
Needline c4							x		47	IE
<b>OV-3</b>										
OperationalExchange c4								x	48	IE
									<b>Elemental Comparative</b>	
									x	Individual Elements (IE)
									x	Duplicate (D)
									x	Same Equivalent Class (SEC)
									x	Individual Extension (IX)

Table 52 - Node Comparison Group

Subgroups Node N	BPMN	UML Act	SysML Act	Timing Diagram	Code IDEFO	OV-5	OV-6a	OV-6b	OV-6c	Count	Comparison Classification	Comment
<b>SysML &amp; UML Activity</b>			X									
Activity Partition											1 Swimlanes	
<b>BPMN Process &amp; Collab</b>												
Swimlanes	X									1	IE	
Pools	X									2	IE	
Lanes	X									3	IE	
<b>OV-5</b>												
Performer						X					D (Node)	
Node						X				4	IE	
ActivitySubject						X						Non node abstraction for object class
<b>OV-6a</b>												
Node							X				D (Node)	
<b>OV-6c</b>												
Lifelines												
<b>OV-6b</b>												
Region								X		5	IE	
											<b>Elemental Comparative Classification</b>	
											X	Individual Elements (IE)
											X	Duplicate (D)
											X	Same Equivalent Class (SEC)
											X	Individual Extension (IX)

## **B. Dissertation Electronic Files**

This dissertation includes a CD (entitled Electronic Files for Understanding the Elements of Executable Architectures) of various dissertation related files:

- (1) the dissertation MAXQDQ database;
- (2) MAXQDA-Reader;
- (3) PDF files of the EAS and EAS-Intermediate meta-models.

To view the dissertation database, place the MAXQDA Reader on your computer and install it. This will allow you read-only access of the dissertation database.

## **VITA**

Mr. Edwin Shuman received his Bachelor of Arts (1980) from the University of Virginia. He received his Master of Science in Computer Systems Management (1990) from the Naval Postgraduate School, in Monterey, California. He retired as a Commander from the United States Navy (1983-2005). He works for the MITRE Corporation in Modeling and Simulation Engineering.