DEVS UNIFIED PROCESS FOR INTEGRATED DEVELOPMENT AND TESTING OF SERVICE ORIENTED ARCHITECTURES

By

Saurabh Mittal

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

2007

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Saurabh Mittal
entitled DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures
and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy in Electrical and Computer Engineering.

____ Date: 04/30/07
Bernard P. Zeigler

____ Date: 04/30/07
Jerzy Rozenblit

____ Date: 04/30/07
Salim Hariri

____ Date: 04/30/07
Larry Head

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.
I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

____ Date: 04/30/07
Dissertation Director: Bernard P. Zeigler

STATEMENT BY AUTHOR

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. Bernard Zeigler, for his endless support, encouragement, mentoring and invaluable guidance over the years. He gave me the freedom to develop an original line of thinking and pursue independent ideas.

I express my thanks to the committee members Prof. Jerzy Rozenblit, Dr. Salim Hariri, and Dr. Larry Head for providing suggestions enhancing the content of this dissertation.

My sincere thanks to my friend Prof. Jose Luis Risco-Martin at University Complutense de Spain for his inspiration, hard-work and collaborative effort.

Special thanks to Dr. Jerry M. Couretas at Lockheed Corporation for providing me feedback and research directions.

I express my thanks to the team at Northrup Grumman that includes Dr. Phillip Hammonds, Kimberly Nunn, Eddie Mak, John Lee, Dale Fulton, Brett Lindskog, Dasia Benson. Working as a team-member in many of the projects was a learning and rewarding experience.

Thanks to Robin Moore at Joint Interoperability Test Command (JITC) for his support and encouragement during the development of GENETSCOPE project.

I would also like to express thanks to my colleagues at ACIMS lab, Chungman Seo, Mahesh Veena, Dr. Saehoon Cheon, Dr. D.H. Kim, Dr. Moon Ho Hwang, Dr. Fahad Bait Shiginah, Dr. Xiaolin Hu and Dr. James Nutaro for helping out over the years and staying together. Many of them are now leading their respective fields.

Last but not the least, my wife Vandana, my family back home and friends who stood by me during this endeavor. Many many thanks to them for their continuous support, understanding and encouragement.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS **9**

LIST OF TABLES **12**

ACRONYMS **13**

ABSTRACT **16**

CHAPTER 1: INTRODUCTION **18**

1.1 Problem Definition **24**

1.2 Thesis Organization **28**

CHAPTER 2: RELATED TECHNOLOGIES AND EARLIER WORK **29**

2.1 Model-Based Software Engineering Process **29**

2.2 Model-Based Testing Methodologies **32**

2.3 Automated Test Case Generation using UML Constructs **36**

2.4 DEVS-Based Bifurcated Model-Continuity Process **41**

2.5 Distributed Modeling and Simulation **45**

CHAPTER 3: DEVS MODELING AND SIMULATION FRAMEWORK **49**

3.1 DEVS System Specifications **51**
3.1.1 Hierarchy of System Specifications 51
3.1.2 Framework for Modeling & Simulation 54
3.1.3 Model Continuity 55

3.2 Model/View/Controller (MVC) Paradigm and DEVS Framework **56**
3.2.1 Real-Time Control and Visualization Limitations of Existing Network Simulators 57
3.2.2 Enhanced MVC 59

3.3 Dynamic Model and Simulation Reconfiguration **61**
3.3.1 Variable Structure DEVS 61
3.3.2 Implementation of Variable Structure in Extended MVC 64
3.3.3 Notion of System Steady State 65

TABLE OF CONTENTS - *CONTINUED*

3.4 Dynamic Simulation Control 67
3.4.1 DEVS Simulation Engine..67
3.4.2 Interrupt Handling..69
3.4.3 The Notion of "Simulation Control" Explored..70
3.4.4 Parameter Control 72
3.4.5 Synopsis 73

CHAPTER 4: REQUIREMENT SPECIFICATIONS AND AUTOMATED DEVS MODEL GENERATION 75

4.1 State-Based System Specifications..77
4.1.1 Sample Example..80

4.2 Message-Based System Specifications with Restricted Natural Language Processing 85
4.2.1 Sample Example: 86
4.2.2 Transformation of Rules to universal Primitives:..89
4.2.3 Design of Entity Node model with multiple message streams: 90

4.3 BPEL/BPMN-Based System Requirement Specifications..95

4.4 Scenario-Based Systems using DoDAF 101
4.4.1 DODAF Specifications 103
4.4.2 Motivation for DoDAF-to-DEVS mapping..107
4.4.3 From OV-6 UML diagrams to DEVS component behavior specifications..110
4.4.4 Representing DoDAF within the System Entity Structure: Multiple Aspects..116
4.4.5 Deriving testable behaviors from DoDAF specification 118

CHAPTER 5: AUTOMATED MODEL-BASED TEST CASE GENERATION ... 125

5.1 Automated Test Case Generator: Concept..126

5.2 Automated Testing Methodology..131
5.2.1 Test Model Generator..132
5.2.2 Test Driver 134

5.3 Synopsis..137

CHAPTER 6: NET-CENTRIC MODEL EXECUTION USING SERVICE ORIENTED ARCHITECTURE.. 140

6.1 DEVSML: Automating DEVS Execution over SOA Towards Transparent Simulators......140
6.1.1 Overview of DEVSML 142
6.1.2 DEVS DTDs and their Standardization..146
6.1.3 Web Services Architecture for DEVSML 152

TABLE OF CONTENTS - *CONTINUED*

6.2 SOADEVS: Remote Execution of DEVS using Simulation Service..156

6.2.1 WWW and Distributed Simulation 157
6.2.2 Abstraction of a Coupled model as an Atomic model with DEVS State Machine..................162
6.2.3 Message Serialization..164
6.2.4 Details about the server architecture 166
6.2.5 DEVSML and SOADEVS 172

CHAPTER 7: DEVS UNIFIED PROCESS: PUTTING IT ALL TOGETHER..... 174

7.1 Automated DEVS Model Generation and DEVSML..177

7.2 DEVSML Collaborative Development...180

7.3 Automated Test-case Generation from DEVS models...182

7.4 SOADEVS: Net-centric Execution using Simulation Service..183

7.5 The Complete Process...185

CHAPTER 8: PROJECTS FROM WHICH DUNIP EVOLVED 187

8.1 Joint Close Air Support (JCAS) Model..190

8.1.1 State-based approach...190
8.1.2 BPMN/BPEL based approach..193
8.1.3 Message-Based Restricted NLP-based approach 197
8.1.4 Automated test case generation for JCAS 198
8.1.5 Net-centric Execution of JCAS 199

8.2 DoDAF-based Activity Scenario 202

8.2.1 Example: Implementation of an Activity Component...202
8.2.2 Activity taken from Zinn as an example 204
8.2.3 DEVS Interpretation of Activity 6 207
8.2.4 Synopsis 217

8.3 Link-16 ATC-Gen Project at JITC...218

8.3.1 Auto Correlation Scenario...219
8.3.2 Auto Correlation Experiment Setup & Results 220
8.3.3 Testing Status 223

8.4 GENETSCOPE Project at JITC..224

8.4.1 SCOPE Command and DoDAF 228
8.4.2 SCOPE Architecture Implementation Using Enhanced MVC 234
8.4.3 Implications of the Example Above and NR-KPP 244

TABLE OF CONTENTS - *CONTINUED*

CHAPTER 9: DISCUSSION 248

9.1 MDA and DUNIP 248

9.2 DUNIP and SCR 252

CHAPTER 10: CONCLUSIONS AND FUTURE WORK 253

10.1 Future Work 258

REFERENCES 261

LIST OF ILLUSTRATIONS

Figure 1.1: Bifurcated Model-Continuity based System Life-cycle Process.................. 25
Figure 2.1: Graphical process extended further from [Utt06] 36
Figure 2.2: Summarizing Model-based Testing.. 38
Figure 2.3: Test Scenario Generation based on requirement specifications.................... 39
Figure 2.4: Bifurcated DEVS-to-DODAF System Lifecycle Development Process 43
Figure 3.1: Framework entities and relationships.. 54
Figure 3.2: Enhanced MVC paradigm with DEVS M&S framework............................ 60
Figure 3.3. DEVS simulation protocol.. 68
Figure 3.4: Hierarchical simulator assignment for a hierarchical model........................ 68
Figure 3.5: Automated test suite execution 70
Figure 4.1: DEVS state machine Document Type Description (statemachine.dtd) 78
Figure 4.2: XML transformation of JTAC state machine described in tabular format ... 82
Figure 4.3: Generated DEVSJAVA code from valid jtac.xml in Figure 4.2.................. 84
Figure 4.4: Rules for Restricted NLP based Requirement Specifications....................... 86
Figure 4.5: Simon Says in English language 87
Figure 4.6: Universal State Machine (USM) for Rule-base Requirement Specifications 90
Figure 4.7: Graphical structure of internals of node entity with two message streams .. 92
Figure 4.8: Constructor for Node entity of the node diagram in Figure 4.7.................... 93
Figure 4.9: Various library functions supporting automated node coupling relations 94
Figure 4.10: Sample BPMN diagram 96
Figure 4.11: View of Web Service implemented as Web Service (courtesy: IBM)........ 97
Figure 4.12: Overview of BPEL-to-DEVS process 98
Figure 4.13: BPEL-to-DEVS transformation... 99
Figure 4.14: WSDL-to-DEVS transformation.. 100
Figure 4.15: Snapshot of a BPMN-to-DEVS Transformation tool 101
Figure 4.16: Linkages among Views 105
Figure 4.17: DoDAF/DEVS execution roadmap... 107
Figure 4.18: Development of DEVS Description model from UML Timing-Sequence Thread 111
Figure 4.19: Representing DoD AF within the SES framework 118
Figure 4.20: SES for enhanced DoDAF with a focus on OV 119
Figure 4.21: DEVS Model generation from various types of Requirement Specifications 124
Figure 5.1: ATC-Gen Development 128
Figure 5.2: IF-THEN rule format 128
Figure 5.3: XML RuleSet 130
Figure 5.4: Overview of ATC-Gen Tool Development.. 132
Figure 5.5: Test Model Generator... 134
Figure 5.6: Enhanced MSVC paradigm with multiple controllers 136
Figure 6.1: DEVS Transparency and model interoperability using DEVSML 143

LIST OF ILLUSTRATIONS - *CONTINUED*

Figure 6.2: Operations leading to model composability using DEVSML.................... 145
Figure 6.3: an SOA object capable of DEVS modeling 148
Figure 6.4: Automated XML snippet for a DEVS atomic model. 149
Figure 6.5: DEVS atomic DTD.. 151
Figure 6.6: DEVS coupled DTD.. 151
Figure 6.7: Web service Architecture for DEVSML Implementation.......................... 153
Figure 6.8: Client side implementation using interfaces. 155
Figure 6.9: DEVS/SOA distributed architecture. 158
Figure 6.10: Hierarchical simulator assignment for a hierarchical model.................... 163
Figure 6.11: Hierarchical simulator assignment with Digraph2Atomic adapter........... 163
Figure 6.12: Communication among services 165
Figure 6.13: Execution of DEVS SOA-Based M&S.. 166
Figure 6.14: Server's package structure for DEVS SOA 167
Figure 6.15: Adapter package containing Digraph to Atomic adapters 168
Figure 6.16: devsml Modeling package for DEVS SOA... 168
Figure 6.17: simulation package in DEVS SOA 169
Figure 6.18: Service package in DEVS SOA 170
Figure 6.19: Proxy package in DEVS SOA.. 171
Figure 6.20: DEVSML implementation over SOADEVS.. 172
Figure 6.21: DEVSML and SOADEVS integrated 173
Figure 7.1: Bifurcated Model-Continuity based System Life-cycle Process................ 177
Figure 7.2: Netcentric collaboration and execution using DEVSML and SOADEVS . 179
Figure 7.3: Client application snapshot implemented as an applet............................... 181
Figure 7.5: GUI snapshot of SOADEVS client hosting distributed simulation 184
Figure 7.6: Server Assignment to Models 185
Figure 7.7: The Complete DEVS Unified Process 186
Figure 8.1: JCAS Operational Scenario.. 191
Figure 8.2: Coupled scenario for JCAS model 192
Figure 8.3: DEVS Execution of JCAS model on console 193
Figure 8.4: JCAS BPMN scenario description 194
Figure 8.5: Snapshot of a BPMN-to-DEVS Transformation tool 195
Figure 8.6: Message-based Restricted NLP description of JCAS scenario.................. 197
Figure 8.7: State-based specification of model CAOC... 198
Figure 8.8: State-machine for CAOC Observer.. 199
Figure 8.9: SOADEVS client running the JCAS model using Simulation services...... 200
Figure 8.10: Simulation output at client's application using SOADEVS client............ 201
Figure 8.11: OV-5 diagram for "select contractor" in IDEF0 notation 203
Figure 8.12: OV-6a diagram for "select contractor" in IDEF3 notation 203
Figure 8.13: Pseudo Code as per Zinn's interpretation and integration procedure........ 203
Figure 8.14: Activity Report Model for Activity 6 generated thru Popkin SA 204

LIST OF ILLUSTRATIONS - *CONTINUED*

Figure 8.15: IDEF3 representation of Activity 6 ("Conduct Dynamic Assessment of Target" TCT 2005 Architecture, 2003: OV-6a) [Zin04] 205
Figure 8.16: Pseudocode for Activity 6 – based on IDEF3 diagram 206
Figure 8.17: DEVS interrelationships of Activity 6 with other Activities. 210
Figure 8.18: DEVS description of Activity 6 in relation to Table 6 components. 211
Figure 8.19: Automated Testing 219
Figure 8.20: Auto Correlation Sequential Diagram... 220
Figure 8.21: Minimal Testable I/O pairs for Auto Correlation.................................... 220
Figure 8.22 Test Drivers Setup Diagram 221
Figure 8.23: Test Model Test Driver successful Auto Correlation scenario 222
Figure 8.24: SUT Test Driver successful Auto Correlation scenario 222
Figure 8.25: Geographic locations of fixed stations 224
Figure 8.26: Communication flow diagram for SCOPE command............................. 224
Figure 8.27: System entity structure for SCOPE command system showing the fixed and mobile (aircraft) stations 227
Figure 8.28: GENETSCOPE simulation architecture for SCOPE command............... 228
Figure 8.29: DEVS M&S and the existing SCOPE command system......................... 230
Figure 8.30: OV-5 for activity sounding 232
Figure 8.31: Simulation architecture for the SCOPE command network..................... 235
Figure 8.32: Experimental frame for GENETSCOPE... 237
Figure 8.33: Ground station configuration screen for Naval Air Station Sigonella 239
Figure 8.34: Mobile station configuration screen where the total count is bounded by the Experimental frame.. 240
Figure 8.35: Callsign entry for a mobile station 240
Figure 8.36: Flight path of mobile aircraft and other details.. 240
Figure 8.37: Experimental frame and ICEPAC data configuration 241
Figure 8.38: Run-time simulation visualization screen for rapid feedback.................. 242
Figure 10.1: The Complete DEVS Unified Process 256

LIST OF TABLES

Table 3.1: DEVS on addressing M&S issues 51
Table 3.2: Hierarchy of system specifications 52
Table 4.1: Tabular structure for State-based specifications 79
Table 4.2: State-based specifications for entity JTAC 80
Table 4.3: Mapping of Rules 1-8 to universal primitives in Universal State Machine (USM) 89
Table 4.4: Mapping of DoDAF with UML and DEVS M&S Elements 116
Table 4.5: Summarizing the contribution of OV-8, 9 to DEVS M&S 123
Table 8.1: Overview of DUNIP application in available case-studies 188
Table 8.2: State machine for component JTAC 191
Table 8.3: Activity-ID mapping for OV-8 and OV-9 207
Table 8.4: Sample OV-8 document 209
Table 8.5: Inner components within Operational Nodes and their mapping with 'standardized' DEVS models 214
Table 8.6: OV-9 description document mapping the Entity component inside Operational Node O1 with the Activity Components defined in OV-8 with port-interfaces 216
Table 8.7: Link 16 functionalities vs. Systems 223
Table 8.8: Activity 4ID mapping for OV-8 and OV-9 231
Table 8.9: Sample OV-8 document 232
Table 8.10: Inner components within operational nodes and their mapping with "standardized" DEVS models 233
Table 8.11: Sample OV-9 Document 234
Table 9.1: Comparison of MDA and DUNIP 250

ACRONYMS

ATC-Gen	Automated Test Case Generator
ALE	Automated Link Establishment
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language For Web Services
BPMN	Business Process Modeling Notation
CDE	Collaborative Development Environment
CJCSI	Chairman of Joint Chief of Staff Instructions
CORBA	Common Object Request Broker Architecture
COTS	Common Off The Shelf
CPN	Colored Petri Nets
CWM	Common Warehouse Model
DEVS	Discrete Event System Specification
DESS	Differential Equations System Specification
DEVSML	DEVS Modeling Language
DoD	Department of Defense
DoDAF	Department of Defense Architecture Framework
DTD	Document Type Definition
DTSS	Discrete Time System Specification
FSM	Finite State Machine
GENETSCOPE	Generic Network System Capable of Planned Expansion
HIL	Hardware-in-the-loop
HF	High Frequency
HLA	High Level Architecture
JAVAML	JAVA Modeling Language
JCAS	Joint Close Air Support
JCIDS	Joint Capabilities Integration and Development System
JITC	Joint Interoperability Test Command
JNI	JAVA Native Interface

ACRONYMS - *CONTINUED*

KIP	Key Interface Profile
LQA	Link Quality Analysis
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MDD	Model Driven Development
MIL-STD	Military Standard
MOE	Measures of Effectiveness
MOF	Meta Object Facility
MVC	Model View Controller
MSVC	Model Simulator View Controller
NCES	Network Centric Enterprise Services
NLP	Natural Language Processing
NR-KPP	Net Ready Key Performance Parameters
OMG	Object Management Group
OV	Operational View
PES	Pruned Entity Structure
PIM	Platform Independent Model
PDM	Platform Domain Model
PSM	Platform Specific Model
RMI	Remote Method Invocation
SCOPE	System Capable of Planned Expansion
SOA	Service Oriented Architecture
SOADEVS	Service Oriented Architecture DEVS framework
SOAP	Simple Object Access Protocol
SES	System Entity Structure
SESM	System Entity Structure Modeler
SSN	Sun Spot Number
SUT	System Under Test
SV	System View

ACRONYMS - *CONTINUED*

TD	Test Driver
TMG	Test Model Generator
TPL	Tactical Programming Language
TV	Technical View
UML	Unified Modeling Language
USAF	United States Air Force
USM	Universal State Machine
V&V	Verification and Validation
WSDL	Web Service Description Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

ABSTRACT

Service Oriented Architectures (SOA) present challenges to current model-based software engineering methodologies such as Rational Unified Process (RUP). In this research effort we propose a process called DEVS Unified Process (DUNIP) that uses the DEVS formalism as a basis for automated generation of models from various requirement specifications and realization as SOA collaborative services. DEVS is inherently based on object oriented methodology and systems theory, and categorically separates the Model, the Simulator and the Experimental frame, and has been used for systems Modeling & Simulation over the years. DUNIP integrates these concepts into DEVS-based Bifurcated Model-Continuity life-cycle development methodology. The life-cycle begins by specifying the system requirements in a number of different formats such as state-based, BPMN/BPEL-based, message-based requirement specifications. DUNIP then automates the generation of DEVS models capable for distributed collaboration. The collaboration uses an XML-based DEVS Modeling Language (DEVSML) framework that provides the capability to integrate models that may be expressed in different DEVS implementation languages. The models are also made available for remote and distributed real-time execution over the SOA middleware in a manner transparent to the user. A prototype simulation framework has been implemented and is illustrated with an application to a system of collaborating military systems implemented and tested using Bifurcated Model-Continuity methodology. We also show how the Department of

Defense Architecture Framework (DoDAF) can be enhanced to incorporate simulation based executable models using the DUNIP process.

CHAPTER 1: INTRODUCTION

In an editorial [Car05], Carstairs asserts an acute need for a new testing paradigm that could provide answers to several challenges described in a three-tier structure. The lowest level, containing the individual systems or programs, does not present a problem. The second tier, consisting of systems of systems in which interoperability is critical, has not been addressed in a systematic manner. The third tier, the enterprise level, where joint and coalition operations are conducted, is even more problematic. Although current test and evaluation (T&E) systems are approaching adequacy for tier-two challenges, they are not sufficiently well integrated with defined architectures focusing on interoperability to meet those of tier three. To address mission thread testing at the second and third tiers, Carstairs advocates a collaborative distributed environment (CDE), which is a federation of new and existing facilities from commercial, military, and not-for-profit organizations. In such an environment, modeling and simulation (M&S) technologies can be exploited to support model-continuity [Hux04] and model-driven design (MDD) development [Weg02], making test and evaluation an integral part of the design and operations life-cycle.

The performance and acceptance of any software system depends on the validation by the customer that is in part supported by the quality of the test-suite that conducts tests on it. Consequently, it also depends on the quality of the test cases used during the validation process. In this particular methodology, the test-cases are automatedly generated that are created with respect to the software requirement set. Modeling languages are used to specify the requirement set and generate test cases [Pra05]. UML is the most widely used and preferred means of such specification. However, the information collected is insufficient as it lacks pragmatic details and the diagrams must be augmented to be used by test programmers. Other approach he suggested is to build a standardized library but then again it requires collaborative effort that spans the entire domain-industry.

Model-based Software Engineering process is commonly referred as Model Drive Architecture (MDA) or Model-Driven Engineering or MDD. The basic idea behind this approach is to develop model before the actual artifact or product is designed and then transform the model itself to the actual product. The MDA is pushed forward by Object Management Group (OMG) since 2001. The MDA approach defines system functionality using platform-independent model (PIM) using an appropriate domain-specific language. Despite such positive benefits of MDA, it lacks sufficient foundation needed to realize this vision. It is underpinned by a variety of standards, some of which have to specified (e.g. executable UML). It is too idealistic and doesn’t involve round-trip iterative nature of software engineering and systems engineering perspective. CORBA also pushed forward by OMG failed to provide distributed collaborative environment and execution.

DEVS formalism [Zei00] exists in many implementations, primarily in DEVS/C++ and DEVSJAVA [ACI06]. Extensions of these implementations are available as DEVS/HLA

[Sar01], DEVS/CORBA [Cho01], cell-DEVS [Wai01], and DEVS/RMI [Zha05]. Since DEVS is inherently based on object oriented methodology, and categorically separates the model, the Simulator and the Experimental frame. However, one of the major problems in this kind of mutually exclusively system is that the formalism implementation is itself limited by the underlying programming language. In other words, the model and the simulator exist in the same programming language. Consequently, legacy models as well as models that are available in one implementation are hard to translate from one language to another even though both the implementations are object oriented. Other constraints like libraries inherent in C++ and Java are another source of bottleneck that prevents such interoperability.

In this research effort we propose a new process called DEVS Unified Process (DUNIP) that utilized the Bifurcated Model-Continuity based life-cycle methodology for a model-based design, execution and collaboration for DEVS models. The life-cycle begins by specifying the system requirements in structured and restricted English that facilitate the requirements gathering from the user. Further, methodologies are also developed to generate DEVS models from BPMN/BPEL-based and message-based requirement specifications. The DEVS models are auto-generated from the specifications and are made available for distributed collaboration using the DEVS Modeling Language (DEVSML) framework. The motivation for this work stems from this need of model interoperability between the disparate simulator implementations and provides a means to make the simulator transparent to model execution. We propose DEVS Modeling

Language (DEVSML) that is built on eXtensible Markup Language (XML) as the preferred means to provide such transparent simulator implementation. The models are also made available for remote and distributed execution using the Service Oriented Architecture (SOA) framework through our developed SOADEVS architecture. A prototype simulation framework has been implemented using web services technology. The central point resides in executing the simulator as a web service. The development of this kind of frameworks will help to solve large-scale problems and guarantees interoperability among different networked systems and specifically DEVS-validated models.

Having developed the complete application framework DUNIP that is net-centric capable, we focus our research effort to a problem equal in magnitude as this but has far reaching usage. A recent DoD mandate requires that the DoD Architecture Framework (DoDAF) be adopted to express high level system and operational requirements and architectures [Dod03a]. DoDAF is the basis for the integrated architectures mandated in DOD Instruction 5000.2 [Dod03b] and provides broad levels of specification related to Operational, System, and Technical views. Integrated architectures are the foundation for interoperability in the joint Capabilities Integration and Development System (JCIDS) prescribed in CJCSI 3170.01D and further described in CJCSI 6212.01D [CJC04, CJC06]. DoDAF and other DoD mandates pose significant challenges to the DoD system and operational architecture development and testing communities since DoDAF specifications must be evaluated to see if they meet requirements and objectives, yet they are not expressed in a form that is amenable to such evaluation. However, DoDAF-compliant system and operational architectures do have the necessary information to construct high-fidelity simulations. Such simulations become, in effect, the executable architectures referred to in the DODAF document. DoDAF is mandated for large procurement projects in the Command and Control domain but its use in relation to M&S is not explicitly mentioned in the documentation [5,8]. Operational views capture the requirements of the architecture being evaluated and System views provide its technical attributes. Together these views form the basis for semi-automated construction of the needed simulation models.

DoDAF is a framework prescribing high level design artifacts, but leaves open the form in which the views are expressed. A large number of representational languages are candidates for such expression. For example, the Unified Modeling Language, (UML) and Colored Petri Nets (CPN) are widely employed in software development and in systems engineering. Each popular representation has strengths that support specific kinds of objectives and cater to its user community needs. By going to a higher level of abstraction, DoDAF seeks to overcome the plethora of “stove-piped” design models that have emerged. Integration of such legacy models is necessary for two reasons. One is that, as systems, families of systems, and systems-of-systems become more broad and heterogeneous in their capabilities, the problems of integrating design models developed in languages with different syntax and semantics has become a serious bottleneck to progress. The second is that another recent DoD mandate also intended to break down this “stove-piped” culture requires the adoption of the Service Oriented Architecture (SOA) paradigm as supported in the development of Network Centric Enterprise Services (NCES). However, anecdotal evidence suggests that a major revision of the DoDAF to support net-centricity is widely considered to be needed. Indeed, under DoD direction, several contractors have begun to design and implement the NCES to support this strategy on Global Information Grid. The result is that system development and testing must align with this mandate – requiring that all systems interoperate in a net-centric environment – a goal that can best be done by having the design languages be subsumed within a more abstract framework that can offer common concepts to relate to. However, as stated before, DoDAF does not provide a formal algorithmically-enabled process to support such integration at higher resolutions. Lacking such processes, DoDAF is inapplicable to the SOA domain and GIG in particular. There have been efforts like [7] that have tried to map DoDAF products to SOA but as it stands out there is no clear-cut methodology to develop an SOA directly from DoDAF, rest aside their testing and evaluation.

We also propose a mapping of DoDAF architectures into a computational environment that incorporates dynamical systems theory and a modeling and simulation (M&S) framework. The methodology will support complex information systems specification and evaluation using advanced simulation capabilities. Specifically, the Discrete Event System Specification (DEVS) formalism will provide the basis for the computational environment with the systems theory and M&S attributes necessary for design modeling and evaluation. We will demonstrate how this information is added and harnessed from the available DoDAF products towards development of an extended DoDAF integrated architecture that is “Executable”. In our attempt to augment the current DoDAF our focus shall remain to add minimal information that would enable DoDAF to become the executable architecture. There are potential advantages of making DoDAF, a DEVS compliant system. We explore the problem of DoDAF using our developed DUNIP framework.

We also demonstrate applications of DUNIP in many active and ongoing research projects. To name a few: the GENETSCOPE project [Gen06] and the ATC-Gen project [Mak06] are in current use at Joint Interoperability Test Command (JITC).

1.1 Problem Definition

This research effort started with the following basic questions:

1. Is there a mechanism by which requirement specifications in English language can give way to a DEVS model that can be simulated?
2. Can various scenario requirement specification methodologies like BPMN/BPEL be used to generate DEVS models?
3. Is DEVS framework dynamically reconfigurable, and collaborative?
4. Is DEVS model net-centric capable?
5. Can you provide a prototype solution that can be used by system designers and can answer some of the requirements of Carstairs[1]?

6. DoDAF architectures are very complex and specified in high-level language in both textual as well as graphical format. Can you employ your solution towards making DoDAF 'executable' over a net-centric platform such as SOA?

The solution to the top five problems resulted in a framework called DEVS Unified Process, a.k.a. DUNIP, which is the thesis. It is built on the Bifurcated Model-Continuity based Life-cycle methodology shown in Figure 1.1. Chapter 7 contains detailed description of each of the elements of DUNIP. In a nutshell, this process employs parallel development of the system model along with the semi-automated test-suite to perform validation and verification studies.

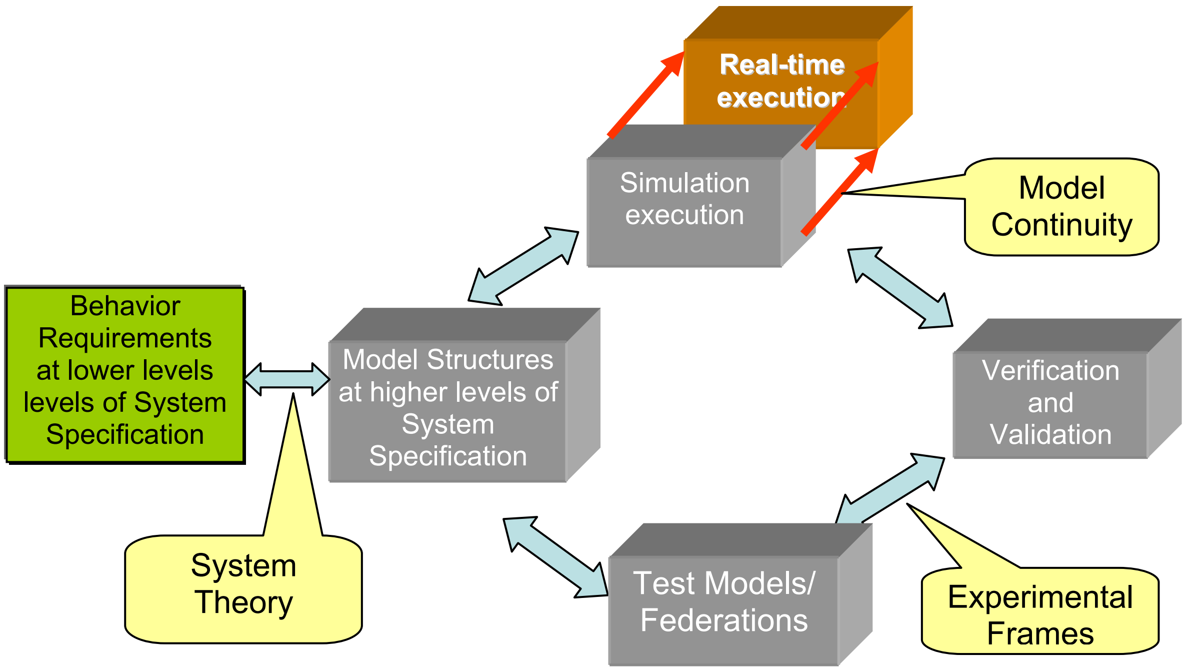

Real-time execution
Simulation execution
Model Continuity
Behavior Requirements at lower levels levels of System Specification
Model Structures at higher levels of System Specification
Verification and Validation
System Theory
Test Models/ Federations
Experimental Frames

Figure 1.1: Bifurcated Model-Continuity based System Life-cycle Process

Beginning towards the solutions, the first two questions raised another series of questions such as:

1. How will you contain the amount of unstructured information that is present in English?
2. How will you extract information from requirement specification that is in different message-based standards?
3. Are there any better means to specify requirements, e.g. Business Process Modeling Notation (BPMN) or restricted English?
4. How will you organize the information set so that object-oriented hierarchical DEVS modeling system could be auto-generated?

The third question required enhancements in the DEVSJAVA framework wherein, dynamic model reconfiguration, dynamic simulation control i.e. ability to steer the simulation in 'right' direction, and DEVS Modeling Language were implemented. DEVSML provided the net-centric collaboration of DEVS models using XML as a middleware.

The fourth question stems forth another phase in the development of DEVS technology wherein DEVS was made executable over Service Oriented Architecture. Layered architecture was proposed and implemented as SOADEVS.

The fifth question resulted in an integrated framework named DUNIP that provides answers to all the previous questions.

The last question demanded DoDAF to be looked into great depth. This effort unearthed various gaps in the current DoDAF document, lapses in high-level model and what information set must be augmented with any specified DoDAF architecture to make it DEVS compliant. The complete process of augmenting this information is described. Finally, the application of DUNIP is aligned with the execution of DoDAF architectures.

This dissertation makes the following research contributions:

1. Enhance the DEVS modeling software DEVSJAVA towards integrated layered Model/View/Controller paradigm for usability and improved visualization technologies
2. Empower DEVS with automated model generation mechanism for multitude of requirement specification formats
3. Development of platform independent DEVS Modeling Language (DEVSML) framework based on XML to provide seamless model integration, reuse and collaboration
4. Development of semi-automated Test case generation from existing DEVS models to advance model-based testing.
5. Development of Simulation Service framework to execute model over net-centric Service Oriented Architecture (SOA)
6. Development of DEVS Unified Process as a tool prototype that provides means to generate models from various requirement specifications formats and execute on SOA.

1.2 Thesis Organization

The dissertation is organized in three chapters following this section. Chapter 2 deals with related technologies and earlier work done in areas relevant to unified process research. Chapter 3 deals with advances made in DEVS technology and current state of DEVSJAVA M&S software Version 3.1. Chapter 4 deals with automated DEVS model generation that includes DoDAF enhancements as well. Chapter 5 describes the automated test case generation methodology. Chapter 6 contains the net-centric execution of DEVS models and details about DEVSML and SOA. Chapter 7 puts it all together in the unifying framework of Figure 1.1 and provides a prototype solution named as DUNIP. Chapter 8 deals with many of the applications of DUNIP. Chapter 9 brings about some of the comparative evaluation of DUNIP with the foundational Model-Driven Architecture approach. Finally, Chapter 10 presents Conclusions and open research directions.

CHAPTER 2: RELATED TECHNOLOGIES AND EARLIER WORK

This chapter provides an overview of current state of the art in the area of model based design, model based testing, automated test case generation, UML constructs and distributed component based simulation. Section 2.1 deals with the OMG effort in pushing Model Driven Engineering and various proposals and concerns that are associated with the paradigm. Section 2.2 deals with model-based testing and various methodologies that are used to develop test cases and generate test-data. Section 2.3 deals with the support provided by UML and the contributions from various research groups in using UML as a means to generate test cases. Section 2.4 deals with the DEVS Bifurcated Model-continuity process which describes the development of semi-automated test-suite developed simultaneously with the development of system model. The last Section 2.5 provides an overview of the concepts in the area of distributed component based simulation and how our research effort fit in.

2.1 Model-Based Software Engineering Process

Model-based Software Engineering process is commonly referred as Model Driven Architecture (MDA) or Model-Driven Engineering. The basic idea behind this approach is to develop model before the actual artifact or product is designed and then transform the model itself to the actual product. The MDA is pushed forward by Object Management Group (OMG) since 2001. The MDA approach defines system functionality using platform-independent model (PIM) using an appropriate domain-specific language. Then given a Platform Definition Model (PDM), the PIM is translated to one or more platform-specific models (PSMs). The OMG documents the overall process in a document called MDA guide.

MDA is a collection of various standards like the Unified Modeling Language (UML), the Meta-Object Facility (MOF), the XML Metadata Interchange (XMI), Common Warehouse Model (CWM) and a couple of others. OMG focuses Model-driven architecture on forward engineering i.e. producing code from abstract, human-elaborated specifications [ref Wiki].

An MDA tool is used to develop, interpret, compare, align etc. models or meta-models. A 'model' is interpreted as meaning any kind of models (e.g. a UML model) or metamodel (e.g. CWM metamodel). An MDA tool may be one or more of the following types:

1. Creation tool: Used to elicit initial models and /or edit derived models
2. Analysis tool: Used to check models for completeness, inconsistencies or define any model metrics
3. Transformation tool: Used to transform models into other models or into code and documentation
4. Composition tool: Used to compose several source models, preferably conforming to the same metamodel

5. Test tool: Used to “test” models. A mechanism in which test cases are derived in whole or in part from a model that describes some aspects of system under test (SUT)
6. Simulation tool: Used to simulate the execution of system represented by a given model. Simply speaking, is the mechanism by which model is ‘executed’ using a programming language
7. Reverse Engineering tool: Intended to transform a particular legacy or information artifact into full-fledged models.

It is not required that one tool may contain all of the features needed for Model Driven Engineering. UML is a small subset of much broader scope of UML. Being a subset of MDA, the UML is bounded by its own UML metamodel. Progress has been made to develop executable UML models but it has not gained industry wide mainstream acceptance for the same limited scope. Potential concerns with the current MDA state of art include:

1. MDA approach is underpinned by a variety of technical standards, some of which are yet to be specified (e.g. executable UML)
2. Tools developed my many vendors are not interoperable
3. MDA approach is considered too-idealistic lacking iterative nature of Software Engineering process
4. MDA practice requires skilled practitioners and design requires engineering discipline not commonly available to code developers.

5. OMG sponsored CORBA project after much promises but it failed to materialize as a widely accepted standard.

2.2 Model-Based Testing Methodologies

Software Testing is not a new area. Many texts have been written in this area and several methodologies have been developed. However, the idea of testing Software Architectures (SA) is comparatively new and requires more rigorous effort. Testers must not only have good development skill but also be knowledgeable in formal language, graph theory, and algorithms [Whi00]. The software testing is usually approached in four phases: 1. Modeling the software's environment, 2. Selecting test scenarios, 3. Running and evaluating test scenarios, and 4. Measuring the testing process. This serves as partition the entire process of testing, similar to the STEP model is given by [Eic96] and [Tor05]. There have been plethora of books on software testing since the first text by Myers in 1978 [Mye78] that address tough testing issues, but the area of Software Architecture Testing has not resulted in a mature methodology that is stable. Research is continuing in the current area. From code-level testing, the testing area has grown to include model-based testing, UML as means to support the modeling, to development of Software Architecture Analysis Methods (SAAM) framework. However, the transition has not been smooth and appears as two separate classes of methodologies. The former is focused towards code level testing, and coverage analysis while frameworks like SAAM is focused towards the entire evaluation and effectiveness of any particular SA. This section summarizes the various efforts that have been put in the recent years in these two disparate classes and argue that Discrete Event Specification-based Modeling & Simulation provides an integral framework that helps align these two fields in coherence.

Of the four part process mentioned above, selecting test scenarios appear to be the most time consuming, rigorous and well attended in the literature. Test execution is assumed to be simpler until DEVS M&S provides a mathematical framework to conduct test-model execution in a formalized manner.

Based on the technique used, the literature is classified into the following categories [Jur04] when generation of test cases is considered:

1. **Random**

   Test cases are generated at random and it stops when there is enough, or a given number is reached or is a user-defined objective has been reached.

2. **Functional**

   Same as Partitioning methodology described above

3. **Control-flow**

   Similar to Path-oriented coverage described above. Test cases are generated until all the program sentences are executed atleast once. However, a full execution is not recommended as it is cost- prohibitive

4. **Data-flow**

   Test cases are generated to cover definitions of each variable for atleast one use of the variable. Many variation of this particular process exist that limit the number of variables and number of paths traversed by this variable are considered

5. **Mutation**

   Test cases are generated based on the mutation operators defined for the programming language in question. Depending on the resources available either all of the mutants are used or only a subset of them (after selective prioritization).

6. **Regression**

   Selection of test cases from an already existing test suite is made through selection criteria or all inclusive methodology. Additions may be suggested that would contribute to the test-suite itself

Two broad categories cover the classical methodology section that involves automated procedures. Specification-based approach and statistical [Tor05] ("intelligent" as described by Pargas [Par99]). Specification based test case generation and selection technique can use a formal [Off99b, Avr95] or natural language [Lut00] to automatically or semi-automatically generate test cases. Many other authors have contributed to this approach [Pas01, Sin03, Sit02, Sir03]. The statistical based techniques consist of Mutation analysis [Bau02] and genetic algorithms [Lin01, Jon96, Mic97].

The next step that comes in line after generation of test data and test cases in automated or semi-automated manner is their selection. Prioritization of such test cases is discussed by Rothermel et.al [Rot01].

Model-based Testing is a variant of testing that relies on explicit behavior models that encode the intended behavior of the system and possibly the behavior of its environment [Utt06]. Pairs of input and output of the model of the implementation are interpreted as test-cases for this implementation: the output of the model is the expected output of the system under test (SUT). This testing methodology must take into account the involved abstractions and the design issues that deals with lumping different aspects as these can not be tested individually using the developed model.

Following is the process for Model-based testing technique [Utt06] as shown in Figure 2.1:

1. a model of the SUT is built on existing requirements specification with desired abstraction levels
2. Test selection criteria are defined with an objective to detect severe and likely faults at an acceptable cost. These criteria informally describe the guidelines for a test suite.
3. Test selection criteria are then translated into test case specifications. It is an activity where a textual document is turned ‘operational’. Automatic test case generators fall into this step of execution.

4. A test suite is 'generated' that is built upon the underlying model and test case specifications.
5. Test cases from the generated test suite are run on the SUT after suitable prioritization and selection mechanism. Each run results in a verdict of 'passed' or 'failed' or 'inconclusive'.

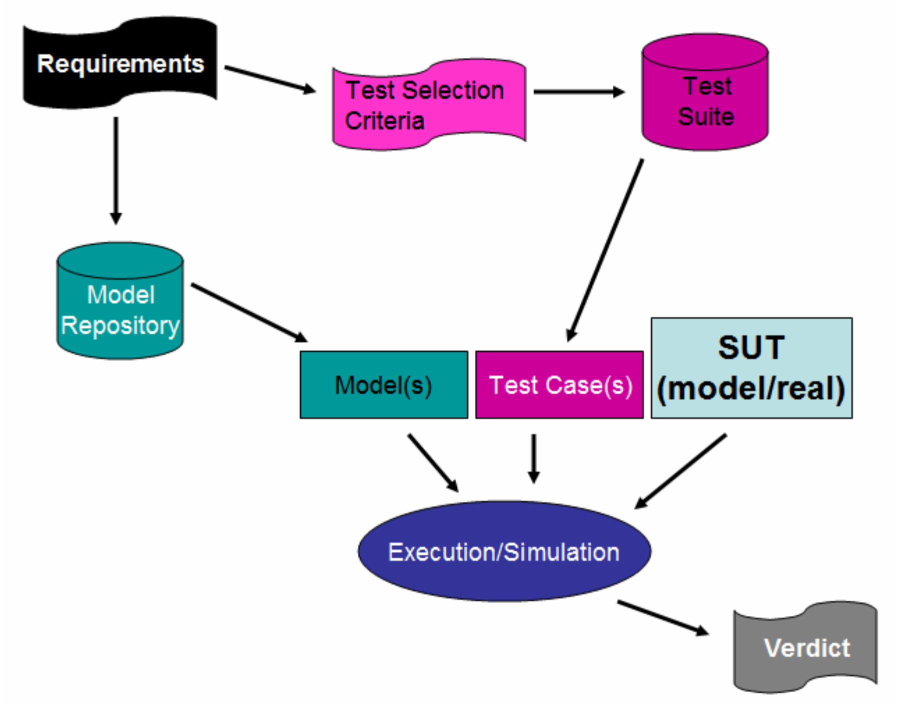


Figure 2.1: Graphical process extended further from [Utt06]

A summary of contributions to the Model-based Testing domain can be seen at [Utt06].

2.3 Automated Test Case Generation using UML Constructs

The performance and acceptance of any software system depends on the validation by the customer that is in part supported by the quality of the test-suite that conducts tests on it. Consequently, it also depends on the quality of the test cases used during the validation process. In this particular methodology, the test-cases are automatedly generated that are created with respect to the software requirement set. Modeling languages are used to specify the requirement set and generate test cases [Pra05]. UML is the most widely used and preferred means of such specification. Williams [Wil02] was the first one to present UML as a test planning tool. However, he also concluded that the information collected is insufficient as it lacks pragmatic details and the diagrams must be augmented to be used by test programmers. Other approach he suggested is to build a standardized library but then again it requires collaborative effort that spans the entire domain-industry.

Offut et al [Off99a, Off03] proposed techniques that adapt predefined state based specifications to generate test cases from UML statecharts. This resulted in the development of UMLTEST – a test data generation tool was integrated with Rational Rose [Rose]. Another parallel effort was done by [Mar] using the same concept of UML statecharts that resulted in the development of Design and Specification-Based Object-Oriented Testing (DAS-BOOT). The java class to be tested is compared with the statechart specification of the class-behavior, thereby defining the association between the code and the specification. Offutt [Off00, Off04] extended their system-level testing work to integration-level testing using UML Collaboration diagrams. Message path coverage criterion was used to generate test cases from UML Sequence diagrams. They concluded that at the unit level, state charts were better compared to sequence charts, but at the integration level, it was vice versa.

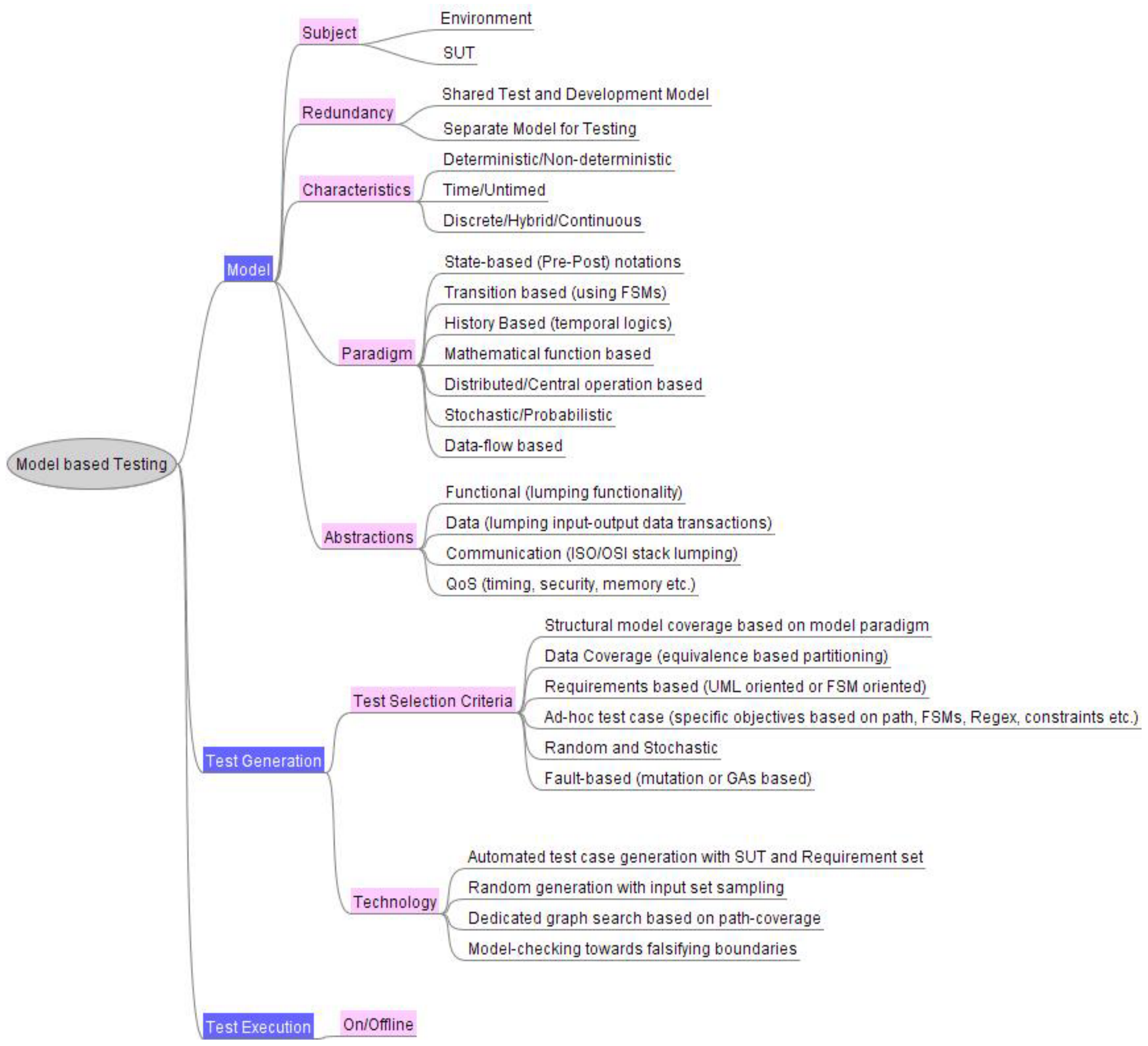

Model based Testing
Model
Subject
Environment
SUT
Redundancy
Shared Test and Development Model
Separate Model for Testing
Characteristics
Deterministic/Non-deterministic
Time/Untimed
Discrete/Hybrid/Continuous
Paradigm
State-based (Pre-Post) notations
Transition based (using FSMs)
History Based (temporal logics)
Mathematical function based
Distributed/Central operation based
Stochastic/Probabilistic
Data-flow based
Abstractions
Functional (lumping functionality)
Data (lumping input-output data transactions)
Communication (ISO/OSI stack lumping)
QoS (timing, security, memory etc.)
Test Generation
Test Selection Criteria
Structural model coverage based on model paradigm
Data Coverage (equivalence based partitioning)
Requirements based (UML oriented or FSM oriented)
Ad-hoc test case (specific objectives based on path, FSMs, Regex, constraints etc.)
Random and Stochastic
Fault-based (mutation or GAs based)
Technology
Automated test case generation with SUT and Requirement set
Random generation with input set sampling
Dedicated graph search based on path-coverage
Model-checking towards falsifying boundaries
Test Execution
On/Offline

Figure 2.2: Summarizing Model-based Testing

Riebish et al [Rie] presented a procedure for iterative software development process in generating test cases with Sequence Diagrams and Use-cases as inputs for requirements engineering. They established that obtaining test-cases systematically can help in documentation of software's usage and interactive behavior.

Another effort by Hartman [Har] led to the development of a tool that integrates with UML to automatically generate black box conformance tests early in the development life cycle. For unit and integration testing, the authors derived tests from State-chart and Sequence Diagrams and for system level they used Use-case and Activity diagrams. The derived test cases were then executed using JUnit or system test tool.

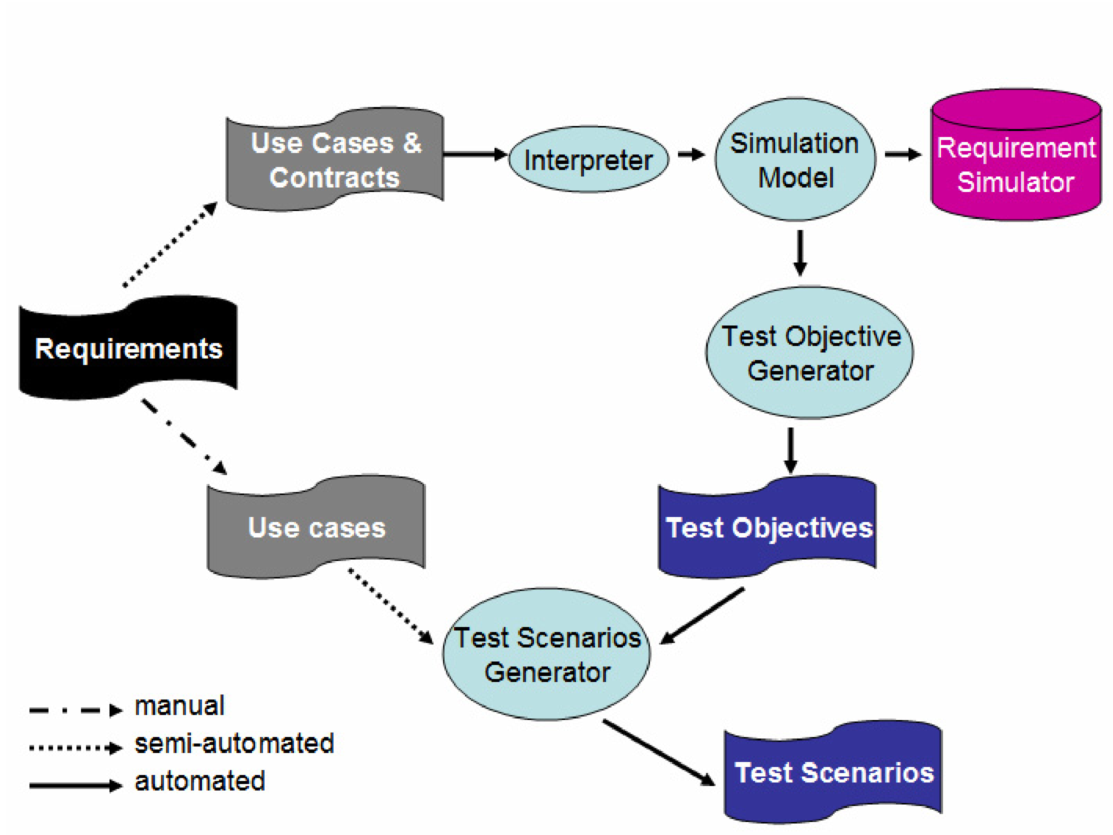


Figure 2.3: Test Scenario Generation based on requirement specifications

One more approach using Use-case was presented by Salem [Sal04]. Use-cases were documented with pre-condition, post-condition, basic and alternate flows and resulted in a traceability matrix. Indeed, this approach provides validation of the requirement set.

Framework for model level testing of behavioral UML model was proposed in another study by Toth et al [Tot03]. This process allowed different UML designs to be tested and design flaws be detected in the modeling phase of the development process. One similar detailed effort was done by Nebut et.al [Neb06] where they employed UML Use-case contracts (Figure 2.3) as the starting point for construction of test cases. They enhanced use-cases with contracts (based on use cases pre and post conditions) as they are defined in [Sou99] and [Coc97]. Building up on the idea by Meyer's [Mey92] at the requirement level, they made these contracts executable by incorporating requirement-level logical expressions. Finally, they constructed a simulation model from these semi-formalized use-cases. The simulation model resulted in the extraction of relevant paths using coverage criteria. These paths are termed 'test objectives'. Each use-case is then described using a UML Sequence diagram and results in 'test scenarios'. Their requirement-based automatic test generation is summarized as in figure above. Other approaches [Bri02, Bas02] also propose to automatedly generate test scenarios from use cases and use-case scenarios.

Automating methods to derive tests from fuzzy descriptions of the use cases is a formidable task. The requirements-based testing techniques already existing [Ber91, Dic93, Leg02, Tah01] are based on formal methods that are difficult to maintain as well as rigorous, only to suitable for mission-critical applications. In [Rys98] it is suggested that for practical purposes, the testers need to focus on methods of systematic test approach. Among varied efforts in proposing test cases [Off99a, Kim99], only a few [Bri02, Bas02, Froh00, Rys99, Rie02] address the system level testing.

Model based testing is a valuable methodology that helps test automation in conjunction with system development. Models allow testers to get more testing accomplished in shorter time. Model based design development, supported by Model continuity when integrated with model-based testing provides the best of all options. The next section presents these integrated ideas.

2.4 DEVS-Based Bifurcated Model-Continuity Process

The *Bifurcated Model-Continuity-based Life-cycle Process* [Zei05a, Zei05b, Mit06] combines the systems theory, M&S framework, and model-continuity concepts reviewed earlier. As illustrated in Figure 2.4, the process bifurcates into two streams – system development and test suite development – that converge in the system testing phase. The Process has the following characteristics:

Requirement Specifications: As described in greater detail below, requirement descriptions are created by designers. Although initially ill-formulated, as the process proceeds, iterative development allows refinement of the requirements and increasingly rigorous formulation resulting from the formalization and subsequent phases.

Formalization by Mapping into DEVS: Concurrently with the formulation or capture of DoDAF specifications, they are formalized as DEVS model components that are coupled together to form an overall Reference Master Model.

Reference Master Model: The master DEVS model serves as a reference model for any implementation of the behavior requirements. This model can be analyzed and simulated with the DEVS simulation protocol to study logical and performance attributes. Using model continuity, it can be executed with the DEVS real-time execution protocol and provides a proof-of-concept prototype for an operational system.

Semi-automated test suite design: Branching in the lower path from the formalized specification, we can develop a test suite consisting of experimental frames called test models that can interact with a System Under Test (SUT) to test its behavior relative to the specified requirements.

Simulation based testing: The test suite is implemented in a Net-centric simulation infrastructure and executed against the SUT. The test suite provides explicit pass/fail/unresolved results with leads as to components/ that might be sources of failure.

Optimization and Fielded execution: The reference model provides a basis for correct implementation of the requirements in a wide variety of technologies. The test suite provides a basis for testing such implementations in a suitable test infrastructure. Test tools should carry into the fielding and operational tests of the system, and provide operationally realistic test cases and scenarios.

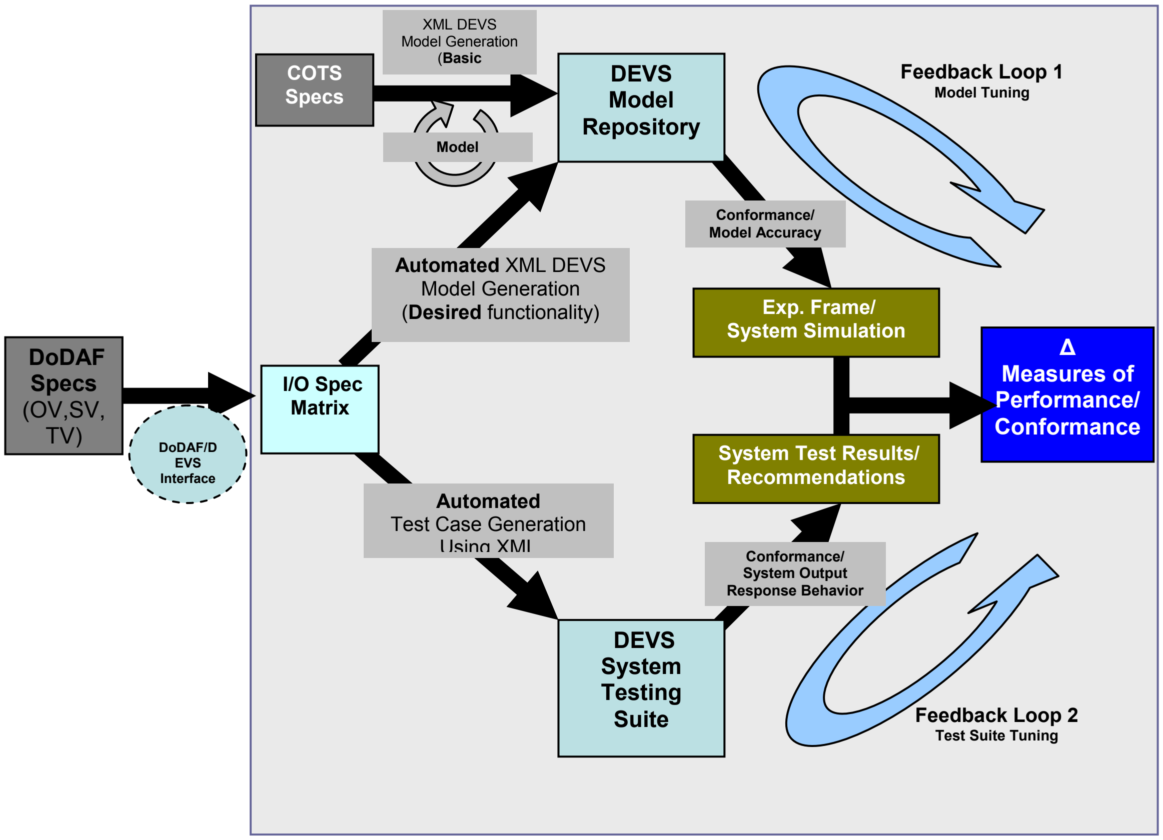


Figure 2.4: Bifurcated DEVS-to-DODAF System Lifecycle Development Process

Iterative nature of development: The process is iterative allowing return to modify the master DEVS-model and its DoDAF precursor requirements specification. Model continuity minimizes the artifacts that have to be modified as the process proceeds. The design methodology provides a process to transform the DoDAF description of architecture to a DEVS representation supporting evaluation and recommendations for a feasible design. Briefly described steps are as follows:

1. The architecture specifications are presented in DoDAF description (or System Requirement Specification) format as Operational Views, System Views and Technical Views.
2. The system specifications are then mapped to DEVS specifications according to the translation described in [Zei05b] that maps the DoDAF views to corresponding DEVS elements. The mapping is illustrated with UML elements and is expressed in XML [Cur02].
3. Test suites for implementations of the design are developed in the test develop stream.
4. Simulation results and their analysis provide the recommendations for a feasible design.
5. Components are developed from the models using Model-continuity principles and the design is verified by the Technical View specifications developed earlier as a part of DoDAF process.

Creation of DEVS Model Repository and DEVS Test Suite occur in a concurrent manner. The DEVS Repository serves as a collection of models that are used to develop scenarios, experimental frames and conduct other simulation oriented analysis. DEVS Test Suite is designed to ensure that the required behavior as expressed in input-output pairs is correctly implemented when integrated in the system with timing constraints. One such semi-automated Test-suite called Automated Test-case Generator (ATC-Gen) has been developed at JITC by Zeigler [Zei05a] and has been applied for Link-16 testing [Mak06]. Analysis of the Experimental frame simulations and the System Test results are compared and evaluated to determine departure from required behavior. This error margin is called the ***Conformance Measure***. Ideally the designed model has a 100% conformance with the Test Suite. If the departure exceeds a given tolerance, the model is revised to increase the model-test conformance. All this assumes that the initial DoDAF specifications have been cast in stone. Typically however, the iterative process will also suggest new or modified specifications at the DoDAF level. The iterative loops can be seen in Figure 4. Finally, when the models conform to the system test specifications, the Test Suite presents the design and performance recommendations as the outcome of this data-centric process. The Model Repository serves as the basis of design of components based on Model-continuity principles and the Test Suite serves as the benchmark for performance evaluation and matching the Technical specifications as developed in the Technical View DoDAF descriptions.

2.5 Distributed Modeling and Simulation

There have been a lot of efforts in the area of distributed simulation using parallelized DEVS formalism. Issues like 'causal dependency' [1] and 'synchronization problem' [11] have been adequately dealt with solutions like: 1. restriction of global simulation clock until all the models are in sync, or 2. rolling back the simulation of the model that has resulted in the causality error. Our chosen method of web centric simulation does not address these problems as they fall in a different domain. In our proposed work, the simulation engine rests solely on the Server. Consequently, the coordinator and the model simulators are always in sync.

Most of the existing web-centric simulation efforts consist of the following components:

1. *the Application*: the top level coupled model with (optional) integrated visualization.
2. *Model partitioner*: Element that partitions the model into various smaller coupled models to be executed at a different remote location
3. *Model deployer*: Element that deployed the smaller partitioned models to different locations
4. *Model initializer*: Element that initializes the partitioned model and make it ready for simulation
5. *Model Simulator*: Element that coordinate with root coordinator about the execution of partitioned model execution.

The Model Simulator design is almost same in all of the implementation and is derived directly from parallel DEVS formalism [1]. There are however, different methods to implement the former four elements. DEVS/Grid [12] uses all the components above. DEVS/P2P [13] implements step 2 using hierarchical model partitioning based on cost-based metric. DEVS/RMI [6] has a configuring engine that integrates the functionality of step 1, 2 and 3 above. DEVS/Cluster [14] is a multi-threaded distributed DEVS simulator built on CORBA, which again, is focused towards development of simulation engine.

As stated earlier, the efforts have been in the area of using the parallel DEVS and implementing the simulator engine in the same language as that of the model. Our present work is not focused in this area. It is focused towards interoperability at the application level, specifically, at the model level and hiding the simulator engine as a whole.

The research of DEVS Standardization group [15] can be divided into four basic areas [8]:

1. Standardization of DEVS formalism
2. Standardization of DEVS models
3. Standardization of the interface of DEVS Simulator
4. Standardization of libraries of DEVS models

Members of Standardization group have worked concerning area 2 where the model's structure is based on XML [16, 17]. However, their general modeling tool ATOM3 [17] is based on meta-meta-modeling. It is based on graph grammars and allows transformation of model to different formalism. Vladimir's [8] work is concerning areas 2 and 4. His implementation of DEVS meta model is based on underlying JAVA Modeling Language (JAVAML) [18]. Vladimir presents a prototype of a modeling tool that aims towards model interoperability but the paper lacks sufficient details and any working example. Our earlier work presents the detailed W3C Schema for DEVS atomic and coupled models [19] as intended by Vladimir. Other research effort using XML description is done by [20] called as DEVSW fits areas 2 and 4 but the code for transition functions is provided by means of pseudo code.

These efforts are in no means similar to what we are proposing in this research, except some of ideas presented by Vladimir. The mentioned efforts are aimed towards development of an independent meta-language that would aid the user to write models effectively and easily and then the process of model generation and simulation is automated using XML. We are focused towards taking XML just as a communication middleware, as used in SOAP, for existing DEVS models. We would like the user or designer to code the behavior in any of the programming languages and let the DEVSML SOA architecture be responsible to create a coupled model, integrating code in either of the languages and delivering us with an executable model that can be simulated. The user need not learn any new syntax, any new language; however, what he must use is the standardized version of DEVS implementation such as DEVSJAVA Version 3.0 [2] (maintained at www.acims.arizona.edu).

This kind of capability where the user can integrate his model from models stored in any web repository, whether it contained public models of legacy systems or proprietary standardized models will provide more benefit to the industry as well as to the user, thereby truly realizing the model-reuse paradigm.

Our work spans areas 2, 3, and 4. In further sections we will provide details about DEVS atomic and coupled DTDs, design of DEVS Simulator interface and standardized libraries used in our implementation.

CHAPTER 3: DEVS MODELING AND SIMULATION FRAMEWORK

This chapter begins by providing an overview of the current DEVS technology and the way in which DEVS is positioned to address the need for a net-centric paradigm for test and evaluation at the system-of-systems and enterprise systems levels. DEVS environments such as DEVSJAVA, DEVS-C++, and others [ACI06] are embedded in object-oriented implementations; they support the goal of representing executable model architectures in an object-oriented representational language. As a mathematical formalism, DEVS is platform independent, and its implementations adhere to the DEVS protocol so that DEVS models easily translate from one form (e.g., C++) to another (e.g., Java) [Zei00]. Moreover, DEVS environments, such as DEVSJAVA, execute on commercial, off-the-shelf desktops or workstations and employ state-of-the-art libraries to produce graphical output that complies with industry and international standards. DEVS environments are typically open architectures that have been extended to execute on various middleware such as the DoD's HLA standard, CORBA, SOAP, and others and can be readily interfaced to other engineering and simulation and modeling tools [Zei00, Bus98, Sar01a, Tol03, Zei03, Sar01b, Cho01]. Furthermore, DEVS operation over web middleware (SOAP) enables it to fully participate in the net-centric environment of the Global Information Grid [CJC06]. As a result of recent advances, DEVS can support model continuity through a simulation-based development and testing life cycle [Hux05]. This means that the mapping of high-level requirement specifications into lower-level

DEVS formalizations enables such specifications to be thoroughly tested in virtual simulation environments before being easily and consistently transitioned to operate in a real environment for further testing and fielding.

Section 3.1 provides basic DEVS theory. Section 3.2 presents the enhanced Model/View/Controller paradigm that encourages complete application development using DEVS. Section 3.3 and 3.4 describes the additions made in latest DEVSJAVA version 3.1 related to dynamic model reconfiguration and dynamic simulation run-time control. It also discusses the inclusion of DEVS Experimental Frame in the enhanced MVC framework and how parameters derived from requirements can find their place at top-level model and simulator configuration. Notion of steady-state of a complex system is also dealt with in Section 3.3.

Brief Overview of Capabilities Provided by DEVS

To provide a brief overview of the current capabilities, Table 3.1 outlines how it could provide solutions to the challenges in net-centric design and evaluation. The net-centric DEVS framework requires advancement to the basic DEVS capabilities, which are provided in later sub-sections.

Desired M&S Capability	**Solutions Provided by DEVS Technology**
Requirement coherence and prioritization	1. Control a simulation on the fly [Mit05b]. 2. Reconfigure a simulation on the fly [Mit04c]. 3. Provide dynamic variable-structure component modeling [Mit04c, Hux03]. 4. Separate a model from the act of simulation
MIL-worth analysis (M&S executable architectures)	
Enhanced user	

capabilities	itself, which can be executed on single or multiple distributed platforms [Zei00]. 5. Simulation architecture is layered to accomplish the technology migration or run different technological scenarios [Sar01b, Mit03d]. 6. With its bifurcated test and development process, automated test generation is integral to this methodology [Zei05]. 7. Provide dynamic simulation tuning, interoperability testing and benchmarking [Mit04c]. 8. Provide rapid means of deployment using model-continuity principles and concepts like "simulation becomes the reality" [Hux05]. 9. Provide net-centric collaboration and integration of DEVS 'validated' models using Web Services [Mit07e]
Execution road maps	
Source selection	
Technology application/transition	
Test support including vulnerability analysis	
Interoperability and integration assurance	
Hierarchical modular construction of models aiding system-of-systems testing	
Provide collaborative distributed environment for M&S	

Table 3.1: DEVS on addressing M&S issues

3.1 DEVS System Specifications

3.1.1 Hierarchy of System Specifications

Systems theory deals with a hierarchy of system specifications that defines levels at which a system may be known or specified. Table 3.2 shows this hierarchy of system specifications (in simplified form; see [Zei00]).

- At level 0 we deal with the input and output interface of a system.
- At level 1 we deal with purely observational recordings of the behavior of a system. This is an input/output (I/O) relation that consists of a set of pairs of input behaviors and associated output behaviors.

- At level 2 we have knowledge of the initial state when the input is applied. This allows partitioning the I/O pairs of level 1 into non-overlapping subsets, with each subset associated with a different starting state.
- At level 3 the system is described by state space and state transition functions. The transition function describes the state-to-state transitions caused by the inputs and the outputs generated thereupon.
- At level 4 a system is specified by a set of components and a coupling structure. The components are systems on their own with their own state set and state transition functions. A coupling structure defines how those interact. A property of coupled systems, which is called "closure under coupling," guarantees that a coupled system at level 3 itself specifies a system. This property allows hierarchical construction of systems, i.e., that coupled systems can be used as components in larger coupled systems.

Level	**Name**	**What We Specify at This Level**
4	Coupled systems	System built up by several component systems that are coupled together
3	I/O system	System with state-space and state transitions to generate the behavior
2	I/O function	Collection of I/O pairs constituting the allowed behavior partitioned according to the initial state the system is in when the input is applied
1	I/O behavior	Collection of I/O pairs constituting the allowed behavior of the system from an external black box view
0	I/O frame	Input and output variables and ports together with allowed values

Table 3.2: Hierarchy of system specifications

As we shall see in a moment, the system specification hierarchy provides a mathematical underpinning to define a framework for modeling and simulation. Each of the entities (e.g., real world, model, simulation, and experimental frame) will be described as a system known or specified at some level of specification. The essence of modeling and simulation lies in establishing relations between pairs of system descriptions. These relations pertain to the validity of a system description at one level of specification relative to another system description at a different (higher, lower, or equal) level of specification.

Based on the arrangement of system levels as shown in Table 3.2, we distinguish between vertical and horizontal relations. A vertical relation is called an association mapping and takes a system at one level of specification and generates its counterpart at another level of specification. The downward motion in the structure-to-behavior direction formally represents the process by which the behavior of a model is generated. This is relevant in simulation and testing when the model generates the behavior which then can be compared with the desired behavior.

The opposite upward mapping relates a system description at a lower level with one at a higher level of specification. While the downward association of specifications is straightforward, the upward association is much less so. This is because in the upward direction information is introduced while in the downward direction information is reduced. Many structures exhibit the same behavior, and recovering a unique structure from a given behavior is not possible. The upward direction, however, is fundamental in the design process where a structure (system at level 3) has to be found which is capable of generating the desired behavior (system at level 1).

3.1.2 Framework for Modeling & Simulation

The framework for M&S as described by Zeigler et al. [Zei00] establishes entities and their relationships that are central to the M&S enterprise (see Figure 3.1). The entities of the framework are source system, experimental frame, model, and simulator; they are linked by the modeling and the simulation relationships. Each entity is formally characterized as a system at an appropriate level of specification within a generic dynamic system. See [Zei00] for a detailed discussion.

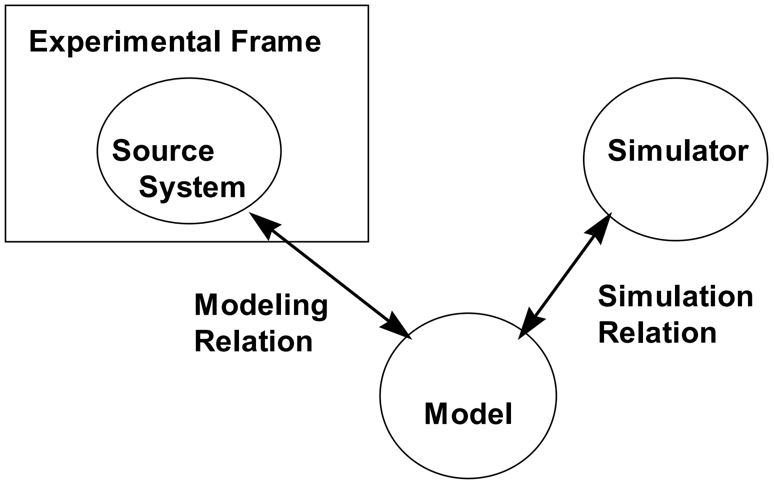


Figure 3.1: Framework entities and relationships

3.1.3 Model Continuity

Model continuity refers to the ability to transition as much as possible of a model specification through the stages of a development process. This is the opposite of the discontinuity problem where artifacts of different design stages are disjointed and thus cannot be effectively consumed by each other. This discontinuity between the artifacts of different design stages is a common deficiency of most design methods and results in inherent inconsistency among analysis, design, test, and implementation artifacts [Cho01]. Model continuity allows component models of a distributed real-time system to be tested incrementally, and then deployed to a distributed environment for execution. It supports a design and test process having four steps (see [Hux05]):

- Conventional simulation to analyze the system being tested within a model of the environment linked by abstract sensor/actuator interfaces;
- Real-time simulation, in which simulators are replaced by a real-time execution engine while leaving the models unchanged;
- Hardware-in-the-loop (HIL) simulation, in which the environment model is simulated by a DEVS real-time simulator on one computer while the model being tested is executed by a DEVS real-time execution engine on the real hardware;
- Real execution, in which DEVS models interact with the real environment through the earlier established sensor/actuator interfaces that have been appropriately instantiated under DEVS real-time execution.

Model continuity reduces the occurrence of design discrepancies along the development process, thus increasing the confidence that the final system realizes the specification as desired. Furthermore, it makes the design process easier to manage since continuity between models of different design stages is retained.

3.2 Model/View/Controller (MVC) Paradigm and DEVS Framework

Although a number of commercial and academic simulators are available for complex network studies, none have the capability to tune the simulation while it is in execution. Due to tight coupling between the network model and the simulation engine in such simulators, the capability to introduce changes in parameter values during execution is limited or non-existent. The work described here has the objective of developing a DEVS-based network modeling and simulation environment with dynamic simulation control and queue visualization. The DEVS modeling and simulation framework separates model, experimental frame, and simulator. This modularity facilitates the development of a simulation framework supporting run-time simulation tuning. The motivation behind providing "real-time" intervention is to support a rapid feedback cycle that allows experimentation with network parameters and structures. This can result in an effective network configuration that is difficult to achieve when turnaround requires hours or days. Furthermore, such instantaneous observation and control enables important transient situations to be recognized and considered.

3.2.1 Real-Time Control and Visualization Limitations of Existing Network Simulators

Some of the limitations of existing network simulation packages are as follows:

- Everything has to be programmed prior to simulating the network.
- User interfaces are not easily customized; they provide largely textual interfaces.
- There is no support for changing parameters and component structures during simulation.
- Simulation run times tend to be long (a few hours); more importantly, if a run ends in a crash, there is no way to intervene and readjust the system.
- There is little run-time visualization of the system behavior to aid understanding and to steer the simulation in a productive direction.
- Model and simulation calibration is a new concept, largely unattended by the legacy and current simulators.
- Model-driven design and development is a new technology supported by only a handful of simulation frameworks.
- Distributed M&S and concepts like model repository are not supported in most of the frameworks.
- Treating an M&S T&E framework as an “online” system by itself is non-existent and unaddressed by current simulators.
- Performance-oriented simulation frameworks are non-existent. Most are bounded by initial model configuration.

To develop a network modeling and simulation environment that addresses these limitations, we extended the existing Discrete Event System Specification (DEVS) software, DEVSJAVA. We discuss the layered architecture underlying the network simulation environment. After describing this architecture, we discuss some proposed run control and visualization techniques intended to greatly improve user understanding of, and ability to control, the complex structural and behavioral relationships characteristic of large network behaviors.

Nutaro [Nut05] proposed the Model/Simulator/View/Controller (MSVC) paradigm, as an extension of MVC. He promoted the separation of model and simulator and has listed many advantages that come about with this idea, most important being the reuse of simulation software, especially in the context of distributed simulations. The other problems that are solved by this paradigm are as follows:

- Distributed simulation protocol changes can be encapsulated within the controller (input and time management policies) and viewer (output policies) objects.
- By separating the viewer and controller it is straightforward to add displays, logging tools, and other output processing devices to the simulator.
- Modeling, simulation, distribution or parallelization, and user interface issues can be addressed separately.

Nutaro demonstrated an application of middleware simulation, wherein the simulator was tuned to display the behavior of certain middleware by incorporating effects such as RTI latency (with reference to distributed simulation HLA framework). In his methodology, the simulator is a thread derived from the controller thread that contains the platform (RTI latency) delay parameter. As the controller thread generates this event, it is communicated to the simulator as well as to the viewer using inter-thread communication. Although Nutaro did not consider model updating or model control, his work constitutes a part of our enhanced MVC framework, where there is full capability in the controller to modify the model as well as the simulator.

Our work is implemented in DEVSJAVA and has a super-thread that runs at the root-coordinator level that monitors the experimental frame for any user-generated activity controls. There exists no viewer thread as the viewer objects are created hierarchically as delegated classes of the model as well as the simulator object. Any modification in their state is also reflected in the contained viewer object. The viewer object displays are derived from the java.awt package. Consequently, they inherently have independent thread that repaints.

3.2.2 Enhanced MVC

Figure 3.2 below provides the graphical representation of an enhanced MVC paradigm. It has been represented with respect to the DEVS M&S framework components. Model and View take their usual functions and meanings. The Control in MVC is explored in more detail and is mapped to the DEVS Experimental frame. Internally, the Experimental frame has a modular structure with a basic control component and controller A and B as derived components.

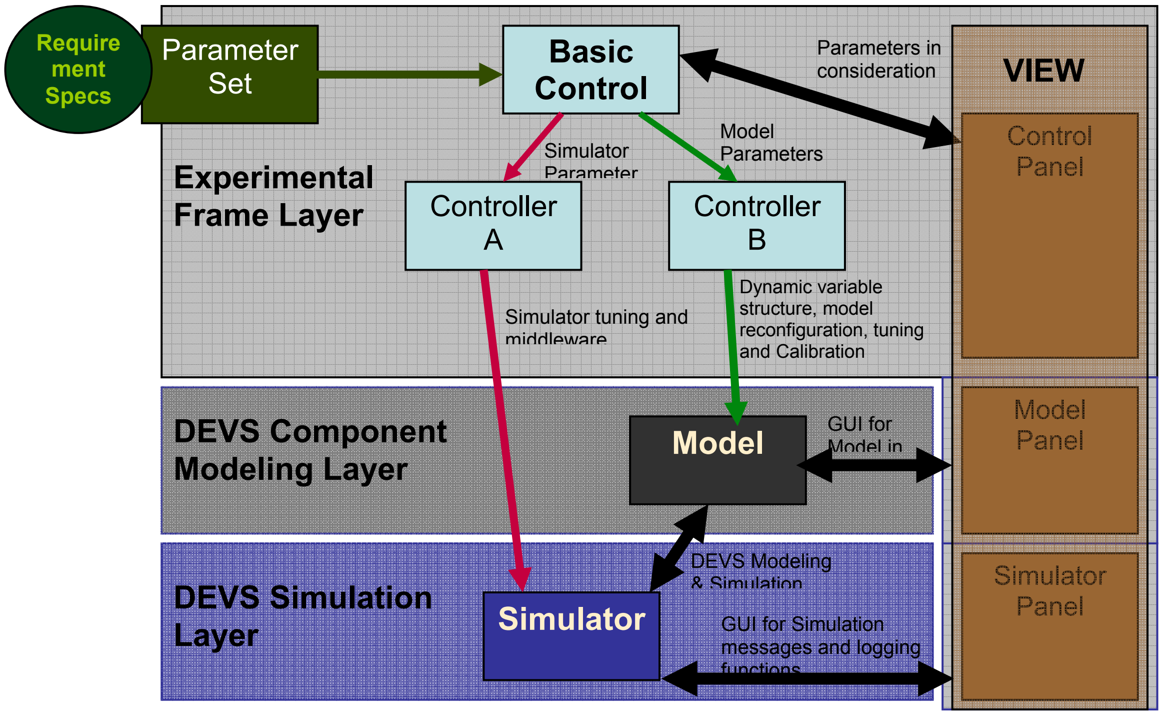

Require ment Specs
Parameter Set
Basic Control
Parameters in consideration
VIEW
Control Panel
Simulator Parameter
Model Parameters
Experimental Frame Layer
Controller A
Controller B
Simulator tuning and middleware
Dynamic variable structure, model reconfiguration, tuning and Calibration
DEVS Component Modeling Layer
Model
GUI for Model in
Model Panel
DEVS Modeling & Simulation
DEVS Simulation Layer
Simulator
GUI for Simulation messages and logging functions
Simulator Panel

Figure 3.2: Enhanced MVC paradigm with DEVS M&S framework

The basic control component translates the information contained in parameter set coming from requirement specifications. It is specialized into two components, one dedicated to simulator middleware control and the other dedicated to model control. It also assigns different parameters to the appropriate controller. In Nutaro [Nut05], controller A provides tools to control the DEVS simulator, more appropriately the middleware aspect of simulation. Controller B provides the toolset to control the model. Details about middleware control can be seen in [Nut05]. Controller B provides functionality to vary the number of components, in addition to the parameters in a component, both at the component and subsystem level. The parameter set for both the controllers is made available to the user as a sliding bar in the controller frame in the

View panel that enables the user to tune the active simulation toward optimum performance

The enhanced MVC has exhaustive control expressed in the experimental frame domain. The Experimental frame component in the DEVS M&S framework is a key construct that enables the user to drive and maneuver the simulation in the “right” direction. The concept of experimental frame, i.e., a mechanism by which an experimental scenario is designed for the model architecture, is further enhanced to enable the user to reconfigure and tune the simulation itself. Benefits of user intervention are explored in more details in [Mit06a] Given that the user has the capability to control the simulation parameters, the issue of extraction and identification of those parameters is taken care by the basic control component that interfaces with the Requirement Specifications document (e.g. DoDAF) in restricted Natural Language. Consequently, the Experimental frame now provides rich control equipment that the operational test designer can use to his advantage.

3.3 Dynamic Model and Simulation Reconfiguration

3.3.1 Variable Structure DEVS

A component is “a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. It conforms to and provides the physical realization of a set of interfaces” [Bro98]. A component system is built by composition of various independent components and by establishing relationships among them. As each component has a high degree of autonomy and has well-defined interfaces, variable structure of components can be achieved during run time. For component-based modeling and simulation, variable structure provides several advantages:

- It provides a natural and effective way to model those complex systems that exhibit structure and behavior changes to adapt to different situations. Examples of these systems include distributed computing systems, reconfigurable computer architectures [Zei85, Zei93], fault tolerant computers [Che90], and ecological systems [Uhr93]. Structure changing and component upgrading is an essential part of these systems. Without the variable structure capability it is very hard, if not impossible, to model and simulate them, let alone study the transition effect that the system incurs when new components are added in a real deployed system.
- From the design point of view, variable structure provides the additional flexibility to design and analyze a system under development. For example, it allows one to design and simulate a system in which the components are added or removed incrementally and form dynamic relationships with existing components.
- It allows one to load only a subset of a system's components during simulation. This is very useful to simulate very large systems with a tremendous number of components, as only the active components need to be loaded dynamically to conduct the simulation. Otherwise, the entire system has to be loaded before the simulation begins.

In general, there are six forms of reconfiguration of component-based systems [Che02]:

1. Addition of a component,
2. Removal of a component,
3. Addition of a connection between two or more components,
4. Removal of a connection between two or more components,
5. Migration of a component, and
6. Update of a component.

The first two operations result in an update of the modeled system where there is a change in the number of components in the system, the next three result in a reconfiguration of the existing system, and the last one results in the modification of the component itself, either its behavior or its interface structure. In DEVS these are collectively known as variable-structure modeling. More details about said operations can be found in [Hux03].

As variable structure changes a component-based simulation during run time, boundary conditions and the limits to which a component affects other components need to be specified with said operation. With reference to Table 1, the model reconfiguration can be implemented at any of the specified levels. These issues are very well addressed in [Hux03]. The variable structure provides the flexibility to design and analyze a complex hierarchical system under development, as well as during a running simulation, as supported by the dynamic structure SES capability.

3.3.2 Implementation of Variable Structure in Extended MVC

Variable structure essentially deals with modification of the component as well as of the number of components that specify the modeled system. Its power lies in its run-time implementation that gives us the capability to study the transition effects when the system is presented with a different number of components and interrelationships. This is entirely a modeling issue and is independent of how the system is simulated when presented with such changes. With the DEVS modeling approach, this is brought to fruition in its modeling layer. With the proposed MVC approach, as is quite obvious, this is implemented in the modeling layer that is in control of the Experimental frame controller layer. The modeling layer that holds the system model, its configuration, and the inter-component relationships receives commands from the Experimental frame on modifying the system. The user is in charge of the Experimental frame. Consequently, if he wishes to modify the system structure he is given the toolbox to modify the model from the experimental frame. Of course, the toolbox is also designed by the modeling designer who decides if the system is to be analyzed and the chosen component plays a significant role in system dynamics and performance. With the closure under coupling property inherent in DEVS formalism, an entire subsystem or an individual component in system can be added as a "component" in the model, in addition to its relationships with other existing components. This property aids in adding a complete system model as a component in a running simulation. With reference to Figure 3.2, the Experimental frame view will contain the controls that the user can perform to modify the structure of the model.

3.3.3 Notion of System Steady State

Evolution is a discipline by which one can understand the growth of a “system” with respect to time. Modeling growth is a difficult concept, let aside simulating “growth.” Biological evolution is studied through looking into the past and seeing how different species have changed according to their environment. In computer systems, the Internet is one such system that has “evolved” over time and has resulted in a World Wide Web that now sustains heterogeneous components sustaining together. Evidently, no one could foresee during its conception days that it had the potential to become the Internet of today with over one billion hosts. In order to model growth, one has to have the capability to modify the structure of constituent components—its interfaces on how it changes when the component is placed in different environments. Biological organisms survive by a process of adaptation, and transmitting this information to progeny with encoded information unlike the computer systems. The computer systems are characterized by rigid interfaces through which they communicate with the “environment.” Certainly we are not focused toward modeling adaptation, though it can be done with the current DEVS suite, but trying to understand the response of system when another component is introduced in the system is of prime importance. The response time of a system is defined as the time taken by the system to display any effect once the model has been modified. There are legacy systems, and the new technology is bringing new components that need to be backward compatible. The situation with respect to IPV4 and IPV6 is one such example in which the communication network has a new standard that needs to be deployed. IPV6 has been around for more than ten years, and according to various sources, it will take another ten years for the current Internet to be completely IPV6 compliant. Testing of IPV6 in conjunction with IPV4 is a big limitation [Dic05]. The analysis of these kinds of situations can be very readily done with the current capability by introducing links and components to the existing network model and observing how the system responds.

The steady state of any network system can be defined as the situation when the computer network is stable and there is constant throughput, network latency, and there are no overflowing buffers in routers. In essence, it boils down to the efficient utilization of bandwidth across all links such that there are no blockages. Total data transmitted from network components is received at the designated destinations, with allowable errors. Consequently, capacity planning is one study that results in quantifying the bandwidth in order to make the system stable with a specified number of components. Looking at it in inverse perspective, finding the number of components that can be sustained by any particular deployed network is of equal interest. The question arises: How can we model a network system in which the system can simulate the growth of this network, arriving at a steady state and providing us with the result that the network can sustain a particular number of components? The current variable structure capability provides us with the needed functionality in which the Experimental frame is given the control to "arrive at steady state." What it actually means is: once a small model of the network system is simulated and utilization is reported, the system continues to keep adding new (preordained) components, along with their relationships, to the existing system until the system reaches a specified network throughput. At what rate the new components are added is a tunable parameter, made available in the experimental frame. This whole exercise shows, given a certain system exhibiting certain behavior, how the system would perform and evolve if let loose, or what the maximum number of components is that the system can be loaded to so that it maintains a steady state! To determine at what result-set the system would break, or if it has a "survivable" nature, is worth conducting analysis. The run-time capability gives us a window to monitor the effects the system incurs when it is modified by external effects like the rate of growth of the system.

3.4 Dynamic Simulation Control

3.4.1 DEVS Simulation Engine

DEVS has been erected on a framework that exploits the hierarchical, modular construction underlying the high level of system specifications. The basis specification structure in all the associated DEVS derived formalisms, e.g., DTSS, DESS, is supported by a class of atomic models. An atomic model is an irreducible component in DEVS framework that implements the behavior of a component. It executes the state-machine and interacts with other components using its defined inports and outports. Each such atomic class has its own simulator class. A network of these atomic models constitutes a *coupled* model that maintains the coupling relationships between the constituent atomic components. Each such coupled model class has its own simulator class, known as a *coordinator.* Assignment of coordinators and simulators follows the lines of hierarchical model structure, with simulators handling the atomic-level components and coordinators handling the successive levels until the root of the tree are reached. These simulators and coordinators form the DEVS simulation engine, and they exchange messages by adhering to what is known as DEVS simulation protocol (see Figure 3.3). The message exchange is depicted in the figure below. For more details about the simulation protocol refer to chapter 8 of [Zei00]. The figure below shows the mapping of a hierarchical model to an abstract simulator associated with it. Atomic simulators are associated with the leaves of the hierarchical model. Coordinators are assigned to the coupled models at the inner nodes of the hierarchy. At the top of hierarchy there is a root-coordinator that is in charge of initiating the simulation cycles (see Figure 3.4).

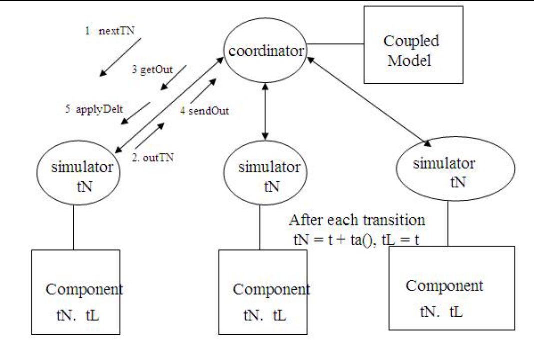


Figure 3.3. DEVS simulation protocol

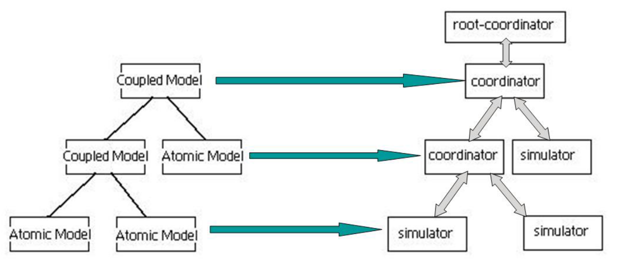


Figure 3.4: Hierarchical simulator assignment for a hierarchical model

Since the DEVS model is based on DEVS formalism that is based on mathematical systems theory, the behavior expressed through DEVS can be translated to any other formalism, though there exist no other theoretical M&S frameworks. With the separation of the model from the simulator, the advantage is that it *supports formalism interoperability*. The next subsection throws light on how an experimental frame intervenes in the DEVS simulation protocol by causing interrupts, and how it implements dynamic simulation control.

3.4.2 Interrupt Handling

The controller frame is built on top of a root coordinator in DEVSJAVA shown in Figure 3.4 above. We developed interfaces to enable the DEVS engine to take into account the change of experimental frame parameters during the simulation run. It generates interrupts, which are handled by the coordinator in DEVSJAVA. The event from the controller frame is handled by the root coordinator that holds the simulation at that instant, taking care of the simulation state. The event then is channeled through the hierarchical simulator network to the intended model. Once the model has been updated, the root coordinator resumes the simulation by reinitiating the DEVS simulation protocol. Consequently, the model is updated in between the running simulation with other events still being held in different component simulators. Only the intended model is updated, which then participates accordingly as before. How this event (parameter update inside a model) brings change or how the system responds to this change can be seen very well in different visualizers. Examples can be seen in later sections and [Mit06a] described a complete DEVS software project called Generic Network Capable of Planned Expansion (GENETSCOPE) [Gen06].

3.4.3 The Notion of "Simulation Control" Explored

Having laid out the framework to implement the dynamic simulation control, we also explored different methodologies in which the simulation can be controlled. Following are the three ways by which the simulation can be interjected and brought to successful execution.

3.4.3.1 Automated Control

In this methodology, we have stored procedures, basically a predefined event list stored as a file that is being read actively during the running simulation and generates events that sends interrupts to the coordinator.

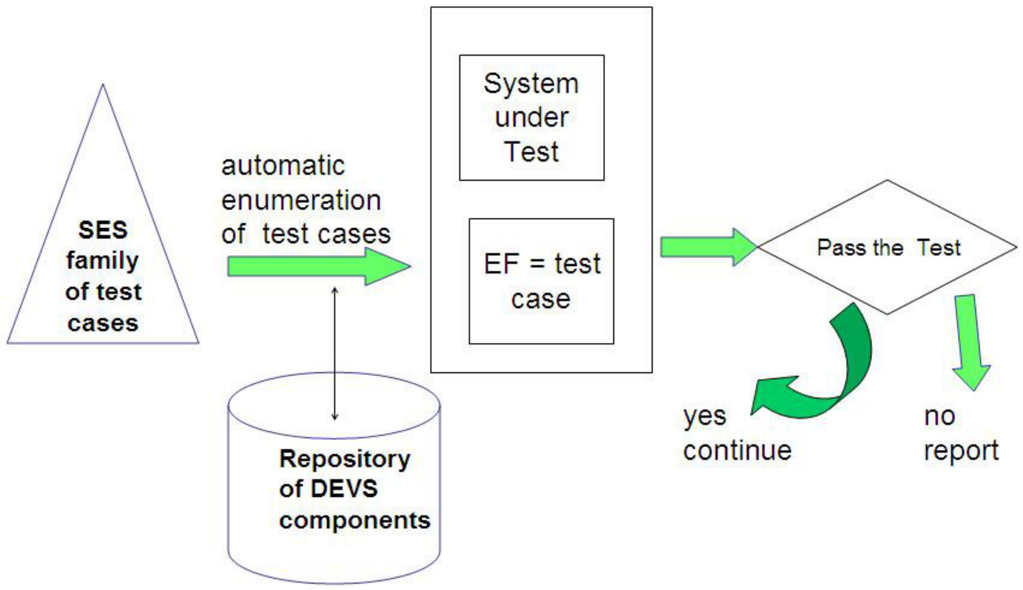


Figure 3.5: Automated test suite execution

This does not require a controller frame that is used to provide real-time interrupts. The experimental frame takes the shape of this file in which different scenarios are preloaded along with simulation parameters. Certainly, execution of a scenario can be considered as one simulation run or a session, but the introduction of a parameter set in the experimental frame that is injected dynamically in the running simulation is of prime interest. This approach has been implemented by Nutaro. This methodology is verily extended toward the following setup shown in Figure 3.5 where the SES family of test cases is implemented as an XML file. The sequence of test is executed in a sequential manner and reported.

3.4.3.2 Manual Reactive Control

In this methodology, the *experimental frame* is operated through a *controller frame* that is designed by the system test designers. The significant parameters and models are identified with reference to the OV-8 document or NR-KPP set and made available in the controller frame [Mit06a]. This methodology provides us with a mechanism to manually interject in the running simulation to introduce modifications. It also provides us with the capability to steer the simulation if the simulation is moving toward a “crash” or if the user wants to see the temporal effects of any parameter update. The capability to steer and study the effects of any single parameter is a powerful capability and is almost nonexistent in current simulators, both in the academic and commercial arenas. There is, however, some software available in the business finance domain that provides this capability. We implemented this capability in one of our active projects. Refer to GENETSCOPE example in Section 8.4. Reactive Mode Testing is a related concept discussed in Section 8.3.3 Testing Status.

3.4.3.3 Hybrid Control

As the name suggests, this methodology takes the best of the above two approaches. This methodology has an automated scenario generation/modification capability as well as reactive control through the *controller frame*. The main purpose of the *controller frame* in this approach is to study the temporal effects and steer the simulation toward optimum performance.

3.4.4 Parameter Control

This subsection presents some ideas on the selection and categorization of parameters. Two classes of parameters that were identified for any system are the tunable parameter set and the result parameter set.

3.4.4.1 Tunable Parameter Set

This set is comprised of the parameters that are to be included in the Experimental frame. This set is termed "tunable" for obvious reasons, as the simulation analysis is conducted to study their effects on the system performance when their values are modified. These parameters are called tunable parameters because these parameters are implemented as a "slider" component in the controller frame with definite bounds. The user can control this slider to tune the system for optimum performance. In the network system terminology, link capacity, router buffer, etc., can be classified as tunable parameters. With reference to the DoDAF and NR-KPP, this makes more sense, as we need to understand the impact of the identified "significant" parameter on the overall system performance [Mit06a].

3.4.4.2 Result Parameter Set

This set is comprised of the aggregated result values that provide the overall system performance estimates. SV-7 provides a place where these documents could be found on a per subsystem basis. However, the holistic result parameters still need to find an appropriate place. There should be a dedicated place in the systems view with respect to the overall system performance. The aggregated parameters in a network system can be thought of as latency, network throughput. This parameter takes leverage from the NR-KPP set that is needed to satisfy the baseline system performance. Its mapping with SV-7 is beyond the scope of the current work.

3.4.5 Synopsis

The above discussion has illustrated how the DEVS simulation framework provides new capabilities in the Experimental frame and how these capabilities are implemented. It also shows that an experimental frame is the place where the user can modify the model and can modulate the simulation according to need. From the basic capability of creating an experimental scenario for the modeled system, we have enhanced it by providing more features like simulation control and parameter tuning. We have also explored various ways simulation control could be performed and how parameters are categorized to find their way in the Experimental frame. Together with the variable structure capability described in section 7, the experimental frame becomes an all-encompassing user interface to a complex hierarchical system model under simulation. It gives the user more power to observe and visualize the simulation by isolation at the parameter level and the component level as well as on the subsystem level.

CHAPTER 4: REQUIREMENT SPECIFICATIONS AND AUTOMATED DEVS MODEL GENERATION

This chapter describes various formats in which the system requirements could be expressed for today's systems and the methodologies leading to generation of DEVS models in an automated manner. The requirements are the most important part of any system development and they are seldom specified in a format that is helpful to the developer at large. Consequently, it is refined throughout the system development lifecycle until the developer as well as the stake-holder settles on a common ground. Testing in such iterative developmental cycle bears the burden of 'meeting' the system specifications. To automate both the model generation and test case generation is a current need of the system design process. Consequently, taking first things first, this section enumerates various formats and implementations in which the requirements could be specified. They are as follows:

- **State-based system specifications:** In this implementation, the system is specified using state-machines with UML [omg] tools such as Rational [rat], Visio [vis] or Enterprise Architect [spa]. Sometimes the DEVS formalized state machine is also available.
- **Rule-based system specifications using restricted natural language processing (NLP):** Natural language such as English can be very ambiguous. To make it more specific, either every aspect must be taken into account for every situation, which results in a voluminous record, or the language itself must be

restricted with chosen keywords. A restricted NLP is provided and model generation is described

- **BPMN/BPEL based system specifications:** Business Process Modeling Notation (BPMN) [bpm] or Business Process Execution Language (BPEL) provide a very high level view of any business process involving sub-processes. It can be looked upon as a super-activity being composed of many activities in a specified sequence and manner. This kind of requirement specification is largely graphical in nature and the information is stored in *.wsdl* and *.bpel* files for BPEL but in proprietary format for BPMN. Methodology is presented on how to extract DEVS model information from such specifications
- **DoDAF-based requirement specifications:** Department of Defense Architecture Framework (DoDAF) [Dod03] is the mandated framework for any future government system architecture specification and it suffers from various deficiencies largely attributed to the fact that M&S is not mandated in it. An enhanced version of DoDAF is proposed and methodology of DEVS model generation is described.

Having analyzed the information set available in different formats, DEVS information is extracted from these sets. To specify a DEVS system, we have the following basic MUST requirements:

- Entities as Objects and their hierarchical organization
- Finite State Machines (FSMs) of atomic models
- Timeouts for each of the phases (States) in atomic models

- Entity interfaces as input and output ports
- External incoming messages at Entity's interface at specified duration in specific State
- External outgoing messages at Entity's interface at specified duration in specific State
- Coupling information derived from hierarchical organization and interface specifications
- Experimental Frame specifications
- As we shall see in each of the ensuing different formats, the required DEVS information is extracted and applied towards automated model generation.

4.1 State-Based System Specifications

UML statecharts are the preferred way of specifying state machines. The process to generate code from UML diagrams is generally human interpretation and depends on the skill and experience of the developer taking into account the help he can get from various stub generation UML tools like Rational, Enterprise Architect etc. However, the UML statecharts are incomplete when it comes to DEVS state machines. DEVS state machine demand more input and specifically the timeouts of each of the specified states to define the complete specifications. The issue of 'timeout' however critical in many mission-critical applications is not addressed adequately in UML literature.

Not going into the rudimentary details about UML statecharts, we would like to propose here a novel way to automate the DEVS state machine specification process. Any state machine can be looked upon as the superposition of two behaviors. The first cycle is the default execution of the machine, wherein it receives no external inputs. The second behavior, which can spawn multiple cycles stems from the actions resulting from reception of various inputs in various states. DEVS categorically separates these two behaviors in its formal δint and δext specification[1].

```
<!ENTITY % variable-info
           "name CDATA #REQUIRED
            type CDATA #REQUIRED">
<!ELEMENT statemachine (deltint, deltext)>
<!ATTLIST statemachine name ID #REQUIRED
                                host CDATA #REQUIRED>

<!ELEMENT transition  (startState?, nextState?, timeout?, outMsg*)>

<!ELEMENT deltint (transitionsInt)>
<!ELEMENT transitionsInt (transition)*>

<!ELEMENT deltext (transitionsExt)>
<!ELEMENT transitionsExt (transitionExt)*>
<!ELEMENT transitionExt (incomingMsg?,transition?)>

<!ELEMENT startState (#PCDATA)>
<!ELEMENT nextState (#PCDATA)>
<!ELEMENT incomingMsg (#PCDATA)>
<!ELEMENT timeout  (#PCDATA)>
<!ELEMENT outMsg (#PCDATA)>
```

Figure 4.1: DEVS state machine Document Type Description (statemachine.dtd)

Consequently, a template based state requirement process is developed where the designer can specify these two behavior cycles. The chosen way to document the state

[1] DEVS atomic specification: M = $< X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta >$
DEVS coupled specification: M = $< X, Y, D, \{M_{ij}\},\{I_j\}, \{Z_{ij}\} >$
Where the symbols have their usual meanings specified in [Zeig]

machines is XML and the constructed artifacts are validated by the DTD. The DEVS state machine DTD is presented in Figure 4.1 above.

The designer can specify the DEVS state machine in a tabular format shown as below:

DEVS Internal State Machine (for default behavior)					
Behavior S.No.	**State (phase)**	**Next State (phase)**	**Timeout**	**Outgoing Message**	
1	A	B	1	C	
2	B	D	10	-	
DEVS External State Machine responding to incoming messages					
Behavior S.No.	**Incoming Message name**	**State (phase)**	**Next State (phase)**	**Timeout**	**Outgoing Message**
1.	X	A	D	5	Y
2.	Z	B	A	1	-

Table 4.1: Tabular structure for State-based specifications

The tabular information presented above in Table 4.1 as a sample is transformed to an XML document validated by the DTD (Figure 4.1 above) and mined for DEVS code generation. The information can also be used to render any graphical output as in conventional UML statechart diagrams.

4.1.1 Sample Example

For illustration purposes, entity name JTAC is specified in the tabular format below (Table 4.2). It is taken from the complete example of Joint Close Air Support (JCAS) provided in detail in Chapter 8.

Entity: JTAC

DEVS Internal State Machine (for default behavior)					
Behavior S.No.	**State (phase)**	**Next State (phase)**	**Timeout**	**Outgoing Message**	
1.	RequestImmediateCAS	WaitForAssignment	0	CASResourceSpec	
2.	WaitForAssignment	Passive	Infinity	-	
3.	ProvideTAC	ContinueExecution	1000	-	
4.	ContinueExecution	Passive	0	CeaseAttack	
5.	WaitForTACRequest	Passive	Infinity	-	
DEVS External State Machine responding to incoming messages					
Behavior S.No.	**Incoming Message name**	**State (phase)**	**Next State (phase)**	**Timeout**	**Outgoing Message**
1.	RequestTAC	WaitForTAC Request	ProvideTAC	10	InitialAttack
2.	YouCanUseUSMC Aircraft	WaitForAssignment	WaitForTACRequest	0	-

Table 4.2: State-based specifications for entity JTAC

The above state description resulted in .xml file validated by the *statemachine.dtd* in Figure 4.1

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE statemachine SYSTEM "statemachine.dtd">

<statemachine name="JTAC" host="LOCAL">
```

```
<deltint>
   <transitionsInt>
      <transition>
         <startState>RequestImmediateCAS</startState>
         <nextState>WaitForAssignment</nextState>
         <timeout>0</timeout>
         <outMsg>CASResourceSpec</outMsg>
      </transition>
      <transition>
         <startState>WaitForAssignment</startState>
         <nextState>Passive</nextState>
         <timeout>Infinity</timeout>
      </transition>
      <transition>
         <startState>ProvideTAC</startState>
         <nextState>ContinueExecution</nextState>
         <timeout>1000</timeout>
      </transition>
      <transition>
         <startState>ContinueExecution</startState>
         <nextState>Passive</nextState>
         <timeout>0</timeout>
         <outMsg>CeaseAttack</outMsg>
      </transition>
      <transition>
         <startState>WaitForTACRequest</startState>
         <nextState>Passive</nextState>
         <timeout>Infinity</timeout>
      </transition>
   </transitionsInt>
</deltint>

<deltext>
   <transitionsExt>
      <transitionExt>
         <incomingMsg>RequestTAC</incomingMsg>
         <transition>
            <startState>WaitForTACRequest</startState>
            <nextState>ProvideTAC</nextState>
            <timeout>10</timeout>
            <outMsg>InitialAttack</outMsg>
         </transition>
      </transitionExt>
      <transitionExt>
         <incomingMsg>YouCanUseUSMCAircraft</incomingMsg>
         <transition>
            <startState>WaitForAssignment</startState>
            <nextState>WaitForTACRequest</nextState>
            <timeout>0</timeout>
         </transition>
      </transitionExt>
   </transitionsExt>
</deltext>
```

```
</statemachine>
```

Figure 4.2: XML transformation of JTAC state machine described in tabular format

The above .xml (Figure 4.2) description is mined using a DOM parser to generate the DEVS atomic model. The generated code is shown in Figure 4.3 below. Each of the rows in table above correspond to the if-else construct in the categorized *deltint()* and *deltext()* functions. For the rows that have Outgoing Message, corresponding constructs are created in the out() function. Other important aspect of the DEVS code generation is the automated creation of DEVS entity interface based on the outgoing and incoming messages. The outgoing message port takes the form of "*out"+Message_name* and the incoming message import takes the form of *"in"+Incoming_message_name*. This relieves burden off the designer in programming the interface specification when developing the state machine of the entity model. The inports and outports are added in the constructor of the said entity and the first state in the default behavior table is the state the entity is initialized with. Further for every state (not equaling 'Passive') corresponding executable stub as 'processing state' function is also provided for the designer to come later on and provide any other logic for the next state. In the figure below, only console messages are printed for logging purposes and tracking the state execution of the encoded entity. The confluent function is largely the external transition function for this particular example.

```
public class JTAC extends ViewableAtomic{

    /** Creates a new instance of JTAC */
```

```
public JTAC() {
    addInport("inRequestTAC");
    addInport("inYouCanUseUSMCAircraft");
    addOutport("outCASResourceSpec");
    addOutport("outCeaseAttack");
    addOutport("outInitialAttack");
}

public void initialize(){
    holdIn("RequestImmediateCAS", 0);
}

public void deltint(){
    if(phaseIs("RequestImmediateCAS")){
        processWaitForAssignment();
        holdIn("WaitForAssignment",0);
    }
    else if(phaseIs("WaitForAssignment")){
        holdIn("Passive",Integer.MAX_VALUE);
        }
    else if(phaseIs("ProvideTAC")){
        processContinueExecution();
        holdIn("ContinueExecution",1000);
    }
    else if(phaseIs("ContinueExecution")){
        holdIn("Passive", Integer.MAX_VALUE);
    }
    else if(phaseIs("WaitForTACRequest")){
        holdIn("Passive",Integer.MAX_VALUE);
    }
}

public void deltext(double e, message x){
    Continue(e);
    for(int i=0; i<x.getLength(); i++){
        if(this.messageOnPort(x,"inRequestTAC",i)){
            if(phaseIs("WaitForTACRequest")){
                processProvideTAC();
                holdIn("ProvideTAC",10);
            }
        }
        if(this.messageOnPort(x,"inYouCanUseUSMCAircraft",i)){
            if(phaseIs("WaitForAssignment")){
                processWaitForTACRequest();
                holdIn("WaitForTACRequest",0);
            }
        }
    }
}

public void deltcon(double e, message x){
    deltext(e,x);
}
```

```
    public message out(){
        message m = new message();

        //deltint output messages
        if(phaseIs("RequestImmediateCAS")){
            m.add(makeContent("outCASResourceSpec",new
entity("CASResourceSpec")));
        }
        else if(phaseIs("ContinueExecution")){
            m.add(makeContent("outCeaseAttack",new
entity("CeaseAttack")));
        }
        ///deltext output messages
        else if(phaseIs("ProvideTAC")){
            m.add(makeContent("outInitialAttack",new
entity("InitialAttack")));
        }

        return m;
    }

    public void processWaitForTACRequest(){
        System.out.println("Processing: WaitForTACRequest");
    }
    public void processProvideTAC(){
        System.out.println("Processing: ProvideTAC");
    }
    public void processContinueExecution(){
        System.out.println("Processing: ContinueExecution");
    }
    public void processWaitForAssignment(){
        System.out.println("Processing: WaitForAssignment");
    }

}
```

Figure 4.3: Generated DEVSJAVA code from valid jtac.xml in Figure 4.2

Similarly, coupled model can be created through similar means. Various frameworks using System Entity Structure (SES) exist that facilitate such automated stub generation. SESM tool developed at ASU is the one in lead that provides GUI based organization and automated SES and XML transformation resulting in DEVSJAVA code. Various methodologies for coupled model creation using SES are not presented here for obvious reasons as this technology is very much in use and mature.

Hence, we see that DEVS state machine can be more easily described in a tabular format rather than going directly to the programming implementation. XML with validating DTD is the preferred way of transformation in an automated manner. Generating the DEVS state machine in such manner and augmenting the code later on for adding further processing constructs based on specific states is very controllable and tractable. Further, such development methodology also encourages 'logic' reuse in specific state-processing functions and separate the execution from the state machine. It also leads to the model-continuity for this component towards SOA where such state-processing functions can become 'services' as discussed in the DEVSML architecture and the proposed universal DEVS atomic DTDs in Chapter 6.

4.2 Message-Based System Specifications with Restricted Natural Language Processing

Any discrete event system communicates internally by way of messages. Developing system requirements in data-flow perspective is of prime importance in this method of requirement specification. English language is used as the preferred means of specifying these interactions, however, bounded by rules that encompass all the possible interactions related to any message type. These rules also limit the way English language is used in terms of removing ambiguous statements. The basic idea is as follows. The entity is considered as a collection of various message streams. It has been observed in complex systems (e.g. GENETSCOPE [Mit06b]) that an entity node can act as receiver and sender simultaneously. This appears logical to consider that a node may be processing more than one messages at a given instant. Consequently, developing a framework where the entity node model can operate with multiple message streams is the objective of this type of requirement specifications.

The rules that provide a binding to this type of requirement specifications are provided below in Figure 4.4. The designer can specify each node's behavior as a sender and a receiver with respect to any specific message type. The message stream is initiated with Rule 1 when the entity (e.g. A below) sends message (e.g of type MessageX) sends to other entity (e.g. B below) at time T.

Rules:

1. A sends *MessageX* to B at time *T*
2. B having received *MessageX* from A, responds with *MessageY* within time *T1*
3. A having send *MessageX*, waits for response until time *T2*
4. B having received MessageX from A, and responding with *MessageY*, A if waiting sends *MessageZ* to B within time *T3*
5. B having received *MessageX* from A, and responding with *MessageY*, can be interrupted with *MessageL* from C
6. B having interrupted with *MessageL* from C, responds with *MessageN* to C within time *T4*
7. B having interrupted with *MessageL* from C, does nothing
8. B forwards *MessageX* to C within time *T*

Figure 4.4: Rules for Restricted NLP based Requirement Specifications

4.2.1 Sample Example:

To illustrate the usage of these rules as requirement specification artifacts let us consider an example in Figure 4.5 that is called 'Simon Says'.

1. The playerGroup consists of Alice, Bruce, and Charles.
2. Simon can send three type of messages to the playerGroup: DoThis(x), SimonSaysDoThis(x), and YourOut
3. A player must respond to DoThis(x) and SimonSaysDoThis(x) with Did(x) or DidNot(x)
4. If player receives DoThis(x) from Simon, and the player responds with Did(x), then Simon sends YourOut to that player.
5. If a player receives SimonSaysDoThis(x) from Simon, and the player responds with DidNot(x), then Simon sends YourOut to that player.
6. When all players have responded, Simon may issue another request.

Figure 4.5: Simon Says in English language

Translating the behavior of Simon in our Rule-based requirements as per Figure 4.4:

1. Simon sends *DoThis(x)* or *SimonSaysDoThis(x)* to Alice at T_1 (Rule 1)
2. Simon sends *DoThis(x)* or *SimonSaysDoThis(x)* to Bruce at T_2 (Rule 1)
3. Simon sends *DoThis(x)* or *SimonSaysDoThis(x)* to Charles at T_3 (Rule 1)
4. Simon having send DoThis(x), waits for response until T2_1 (Rule3)
5. Simon having send SimonSaysDoThis(x), waits for response until T2_2 (Rule 3)

Translating behavior of Alice or Bruce or Charles in our Rule-based requirements:

Message Stream: DoThis(x)

1. Alice having received *DoThis(x)* from Simon, responds with *Did(x)*, within time *T1* (Rule2)
2. Alice having received *DoThis(x)* from Simon, and responding with *Did(x)*, Simon if waiting sends *YourOut(x)* to Alice within time *T3* (Rule 3)

Message Stream: SimonSaysDoThis(x)

1. Alice having received SimonSaysDoThis(x) from Simon, responds with DidNot(x), within time T1 (Rule2)
2. Alice having received SimonSaysDoThis(x) from Simon, responding with DidNot(x), Simon if waiting sends YourOut(x) to Alice within time T3 (Rule 3)

The other statements in Figure 4.5 can be similarly used to provide requirement specification for the coupled DEVS specifications (see statement 1). The use of Statement 2 will be discussed shortly. Statement 3 becomes redundant in our current setup of rule-based requirements as shown above. Statement 6 starts up a new cycle of these message streams. Hence, we see that using only Rule 1, 2 and 3 (Figure 4.4) we can specify the requirements of Simon Says example.

The next step is the automated code generation of Simon or Alice or Bruce etc., which leads us to the transformation of these rule to universal primitives (Table 4.3). Each of these primitives corresponds to a specific state in the DEVS state machine. The basic idea is that that a rule once translated to a specific primitive (and correspondingly to a DEVS state) can be turned on-off depending on the rules usage resulting in a dynamic state machine. The concept of Universal State Machine USM (Figure 4.6) germinates from this idea which contains all of these rule-based primitives.

4.2.2 Transformation of Rules to universal Primitives:

Rule	Name Tag	Primitive Name	Token 1	Token 2	Token 3	Token 4
1	send	*holdSend #1*	MsgX_A_B	T		
2	respond	*receiveSendInterrupt #1*	MsgX_A_B	MsgY_B_A	T1	
3	wait for response	*waitReceive*	MsgX_A_B	T2		
4	acknowledge	*waitHoldSend*	MsgX_A_B	MsgY_B_A	MsgZ_A_B	T3
5	interrupted	*receiveSendInterrupt #2*	MsgX_A_B	MsgY_B_A	IMsgL_C_B	
6	acknowledge interrupt	*receiveSendInterrupt #3*	MsgX_A_B	IMsgL_C_B	AMsgN_B_C	T4
7	ignore interrupt	*receiveSendInterrupt #4*	MsgX_A_B	IMsgL_C_B		
8	forward	*holdSend #2*	MsgX_B_C	T		

Table 4.3: Mapping of Rules 1-8 to universal primitives in Universal State Machine (USM)

Now, assuming that entity A send MessageX to entity B at time T (Rule 1), various other features of such rule-based requirement specifications are as follows:

- Depending on the behavior specified, resulting in activation of Rule 1-8, primitives are switched on-off in the Universal State Machine (USM)
- The message-stream for Message X is encoded in both entities A and B i.e each of the node models for A and B will contain the same code for MessageX processing as the USM takes care of sending and receiving operations for Message X
- The USM generates a stand-alone DEVS state-machine for any Message Stream initiated by message of type Message X
- Each node having multiple message streams keeps track of 'active' message streams in a queue based system as a state variable.

- Each node exists as a coupled model with node controller coordinating with various message streams and tracking operational message streams.

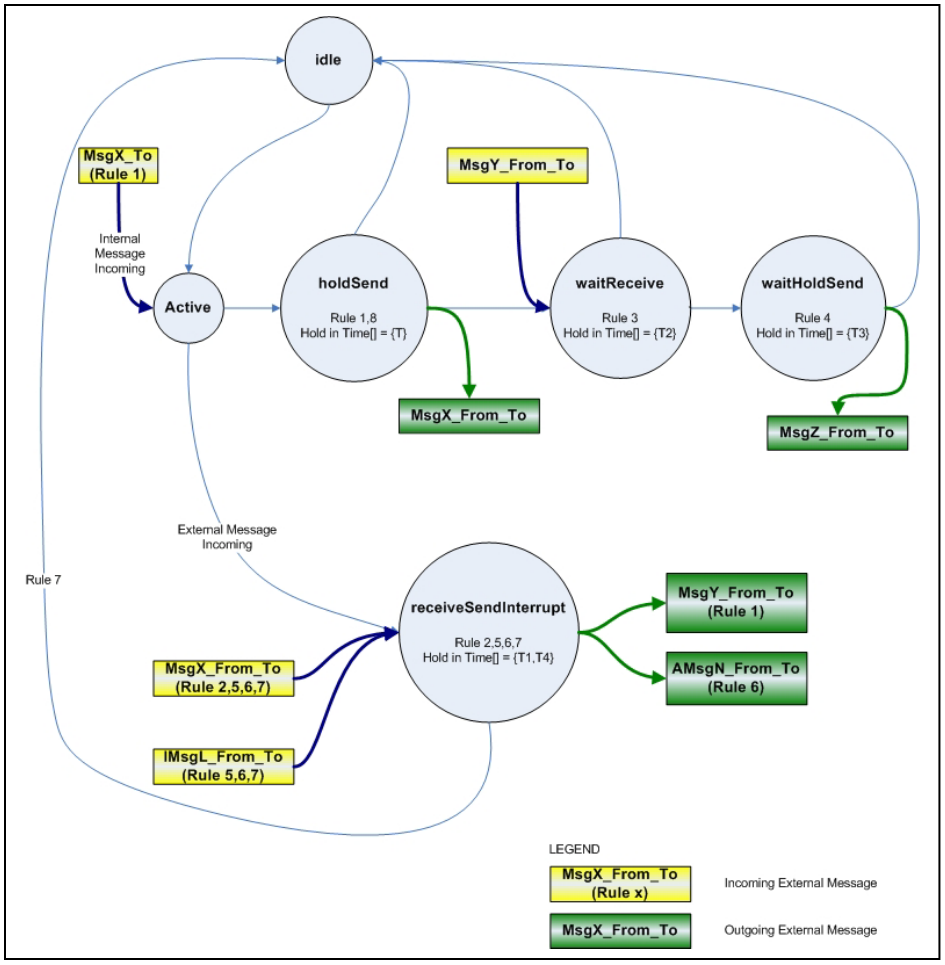


Figure 4.6: Universal State Machine (USM) for Rule-base Requirement Specifications

4.2.3 Design of Entity Node model with multiple message streams:

The entity node model (e.g. A or B in Figure 4.4) is a coupled model consisting of:

- Node controller switch (having state variables for each of message streams)
- Traffic generator that initiates message streams

- Message streams
- Internal coupling relations between the above components
- External coupling relations between node controller and boundary of entity node model

The automated creation of a node model begins with specification of various message streams as laid out in example 'Simon Says' Figure 4.5. Each of messages is given an ID which becomes the identification of the particular message stream for tracking and reporting purposes. Rule 1 in Figure 4.4 is the starting point. Information from Rule 1 and other rules is categorized into two sets of messages:

- Initiator messages
- Participant messages

Initiator messages directly correspond to Rule 1, which activates the message stream and set the node model in sender mode. In terms of USM in Figure 4.6, Message X corresponds to initiator message.

Participant messages correspond to various other messages, such as Message Y, Z, L and N used in Rule 2-7 and others indicated in USM and Table 4.3.

The initiator messages find their way in the Traffic Generator as they contain information about the time at which they will activate the particular message stream.

The node model displaying the coupling information is shown in Figure 4.7 below for the example Simon Says. The formal code and representation is given in the Appendix section. For a detailed example refer Chapter 8.

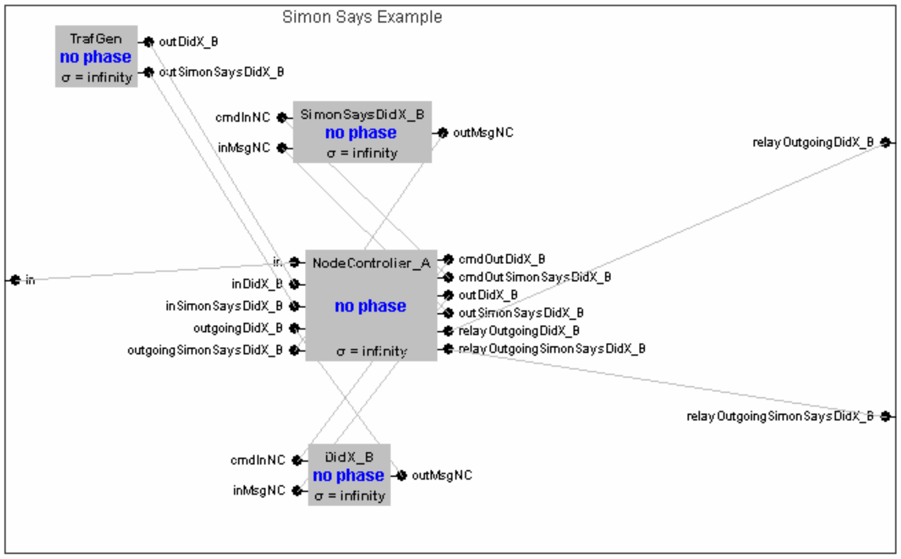


Figure 4.7: Graphical structure of internals of node entity with two message streams as in example of Figure 4.5

The automated code resulted in above coupled diagram is shown in Figure 4.8 below. The constructor only takes information about the various message Ids (a.k.a streams) that are encoded using USM.

```
public SimonSaysExample(String[] msgIds){
        super("SimonSaysNodeA");
        this.msgIds = msgIds;
        msgStreams = new Hashtable();

        for(int i=0; i<msgIds.length; i++){
            ViewableAtomic ms = new ViewableAtomic(msgIds[i]);
            addMsgStreamPorts(ms);
            add(ms);
            msgStreams.put(msgIds[i],ms);
        }
        ViewableAtomic tg = new ViewableAtomic("TrafGen");
        addMsgsToTG(tg, msgIds);
        add(tg);

        ViewableAtomic nc = new ViewableAtomic("NodeController_A");
        nc.addInport("in");
        addMsgsToNC(nc, msgIds);
        add(nc);

        coupleComponents(tg, nc, msgIds);
        coupleNCwithMsgStreams(nc, msgIds);

    }
```

Figure 4.8: Constructor for Node entity of the node diagram in Figure 4.7

The automated code borrows various automated coupling constructs stored in the super class ‘base’ message node as shown in Figure 4.9 below.

```
    public void coupleComponents(ViewableAtomic tg,
                              ViewableAtomic nc, String[] msgIds){
        for(int i=0; i<msgIds.length; i++){
            addNodeCouplingTG_NC(tg, nc, msgIds[i]);
            addSelfNodePortsAndCouplingForMsg(nc, msgIds[i]);
        }
    }

    public void coupleNCwithMsgStreams(ViewableAtomic nc,  String[]
msgIds){
        for(int i=0; i<msgIds.length; i++){
            ViewableAtomic ms =
(ViewableAtomic)msgStreams.get(msgIds[i]);
            addNCtoMsgStreamCoupling(nc, ms, msgIds[i]);
        }
    }

    public void addNCtoMsgStreamCoupling(ViewableAtomic nc,
                                ViewableAtomic ms, String msgId){
        addCoupling(nc, "cmdOut"+msgId, ms, "cmdInNC");
        addCoupling(nc, "out"+msgId, ms, "inMsgNC");
        addCoupling(ms, "outMsgNC", nc, "outgoing"+msgId);
    }

    public void addSelfNodePortsAndCouplingForMsg(ViewableAtomic nc,
String msgId){
        this.addOutport("relayOutgoing"+msgId);
        this.addInport("in");
        addCoupling(nc, "relayOutgoing"+msgId, this,
"relayOutgoing"+msgId);
        addCoupling(this, "in", nc, "in");
    }

    public void addMsgStreamPorts(ViewableAtomic ms){
        ms.addInport("cmdInNC");
        ms.addInport("inMsgNC");
        ms.addOutport("outMsgNC");
    }

    public void addMsgsToTG(ViewableAtomic tg, String[] msgIds){
        for(int i=0; i<msgIds.length; i++){
            addMsgToTG(tg, msgIds[i]);
        }
    }

    public void addMsgsToNC(ViewableAtomic nc, String[] msgIds){
        for(int i=0; i<msgIds.length; i++){
            addMsgToNC(nc, msgIds[i]);
            this.addPortsAtNodeController(nc, msgIds[i]);
        }
    }
    public void addMsgToTG(ViewableAtomic tg, String msgId){
        tg.addOutport("out"+msgId);
    }

    public void addMsgToNC(ViewableAtomic nc, String msgId){
        nc.addInport("in"+msgId);
    }

    public void addNodeCouplingTG_NC(ViewableAtomic tg,
                                    ViewableAtomic nc, String msgId){
        addCoupling(tg, "out"+msgId, nc, "in"+msgId);
    }

    public void addPortsAtNodeController(ViewableAtomic nc, String
msgId){
        nc.addInport("outgoing"+msgId);
        nc.addOutport("relayOutgoing"+msgId);
        nc.addOutport("cmdOut"+msgId);
        nc.addOutport("out"+msgId);
    }
```

Figure 4.9: Various library functions supporting automated node coupling relations

Having constructed a node model with multiple message streams in an automated manner, the next step is construction of coupled scenario model containing various node entity models. The details of coupled scenario construction are not provided as it does not require any research in present state of DEVS advancements. Tools like SESM modeler developed at Arizona State University provide a very solid framework of XML and SES towards creation of coupled models.

For a detailed example for requirement specifications using restricted NLP refer Chapter 8.

4.3 BPEL/BPMN-Based System Requirement Specifications

Business Process Execution Language or BPEL is a business process modeling language that is executable. It is serialized in XML and communicated over a net-centric platform. It is the latest in development of business scenarios where many business stake holders participate towards a common business goal. It is an orchestration language giving a global view of the participating business process communicating over the Web. BPEL's messaging facilities depend on the use of Web Service Description Language (WSDL) 1.1 to describe the outgoing and incoming messages. The BPEL specification [bpel] is described as a process flow that uses various Web Services in a goal-oriented scenario. BPEL4WS provides language for the formal specification of business processes and business interaction protocols. It extends the Web Services interaction model and enables it to support business transactions. This information is stored in file with extension *.bpel*. The interfaces of the constituent Web Services are stored in a WSDL file with extension *.wsdl*.

There is no standard graphical notation for WS-BPEL. Another format known as Business Process Modeling Notation (BPMN), mainly as a graphical front-end is also in use to capture the BPEL process descriptions and many vendors have their proprietary means to portray and design a BPEL process. Interoperability among these different vendors is one of the major problems we faced during this endeavor. Mapping of BPMN to BPEL is problematic and fundamental differences between these two approaches along with vendor issues make it very difficult and in some cases impossible to produce a 'valid' BPEL specification. Even more difficult is the problem of BPMN-to-BPEL roundtrip engineering! A sample BPMN diagram looks like Figure 4.10

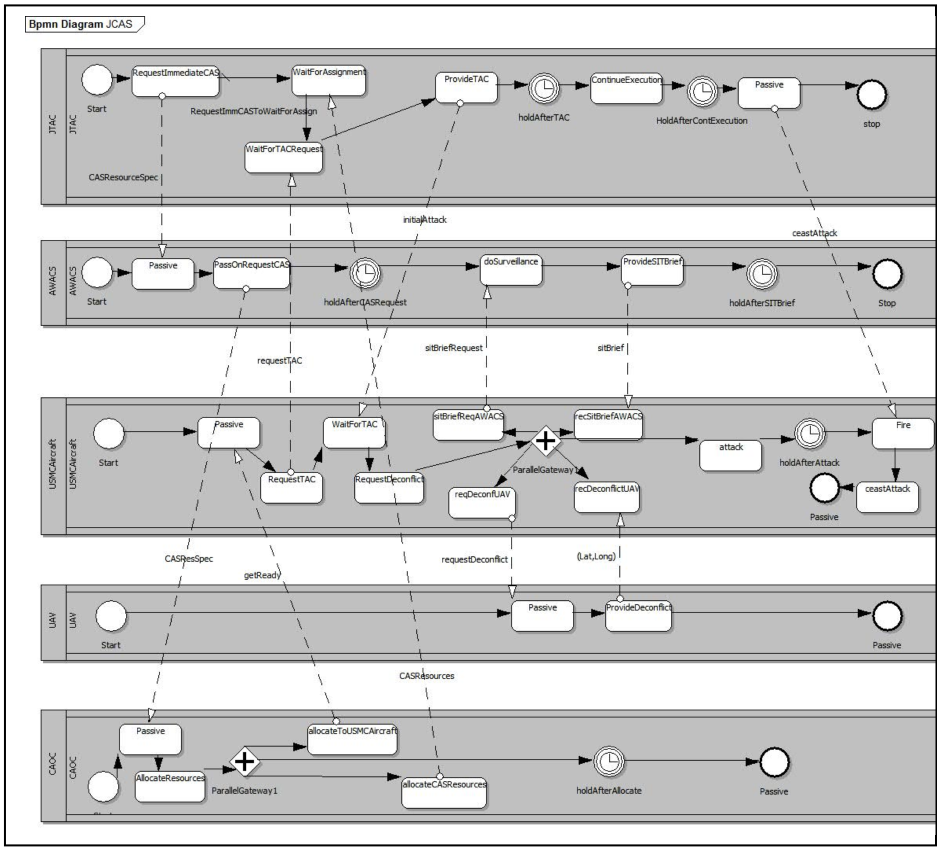


Figure 4.10: Sample BPMN diagram

A typical Web Service when encapsulated in BPEL4WS framework look like Figure 4.11. The composition primitives found in BPEL4WS comes primarily from many years of workflow and business process integration, hence its positioning as a business process composition language. The role of BPEL4WS is to define a new Web service by composing a set of existing services. BPEL4WS is just a language to implement such a composition. The interface of the composite service is described as a collection of WSDL portTypes and the composition (called the process) indicates how the service interface fits into the overall execution of the composition. As can be clearly seen from the figure, there is natural overlapping with DEVS component architecture. DEVS atomic component is also port-interface based hiding within itself the state machine (as a process) for that particular component. In one of our other efforts, we proposed DEVS Service component that builds on top of the DEVS atomic component and can be readily deployed using Model-continuity principle in Web service architecture.

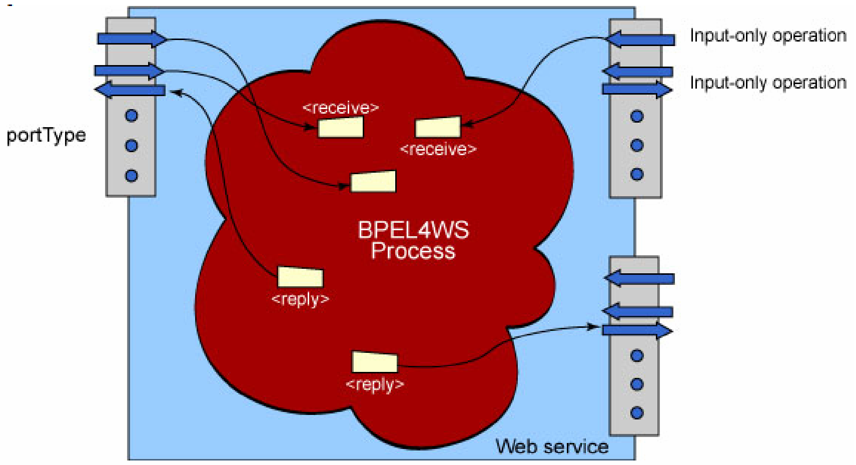


Figure 4.11: View of Web Service implemented as Web Service (courtesy: IBM)

The objective of this section of research deals with the development of DEVS models from any BPEL4WS specification constituting of *.bpel* and *.wsdl* file. As we shall see in a moment that all the information is contained there in the *.bpel* and *.wsdl* files, it only needs to be mined and put into DEVS perspective. The overall process for such transformation is shown in Figure 4.12

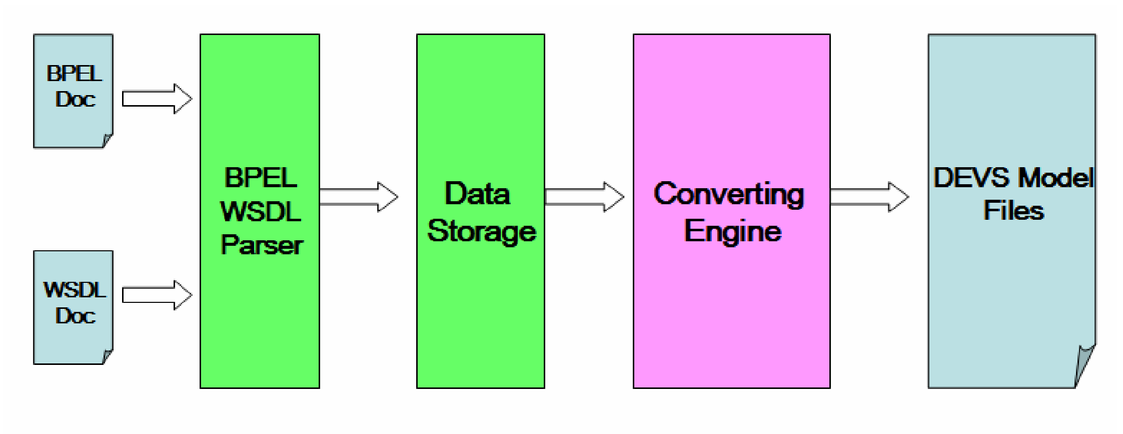


Figure 4.12: Overview of BPEL-to-DEVS process

Not going into the details of actual XML file for *.bpel*, the structure of *.bpel* can be graphically depicted as in Figure 4.13 for clearer understanding. Also shown in the figure is extraction of DEVS elements from various BPEL constructs. As can be seen, the message interchanged between different services, various coupling relations, listing of atomic models and hierarchical organization can be obtained from a *.bpel* file.

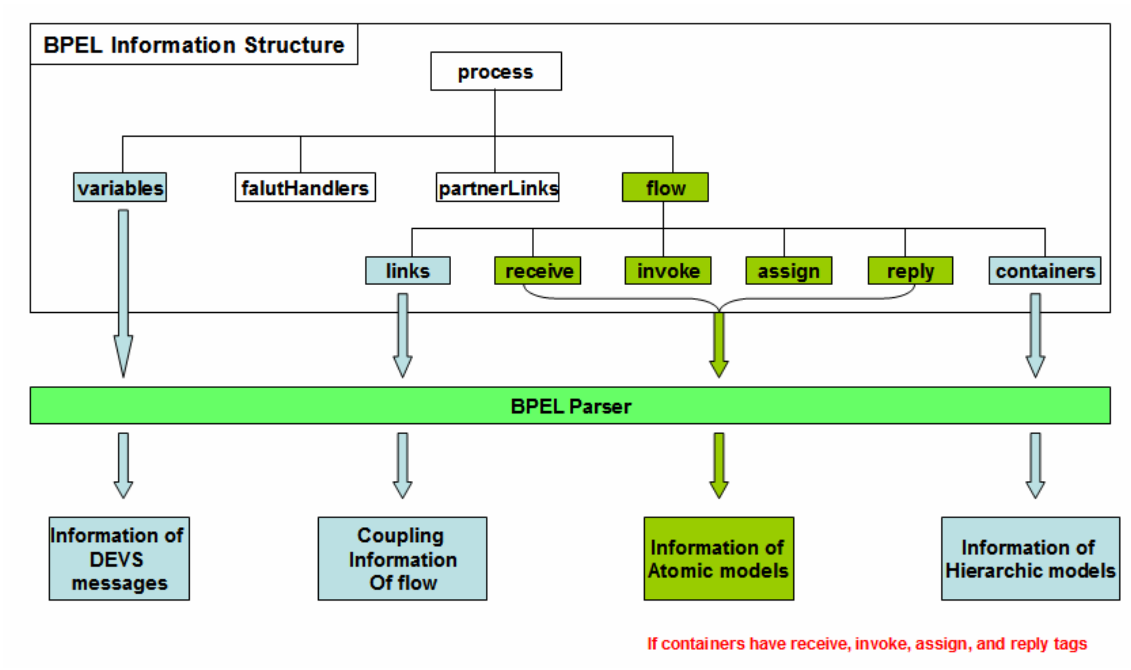

BPEL Information Structure
process
variables
falutHandlers
partnerLinks
flow
links
receive
invoke
assign
reply
containers
BPEL Parser
Information of DEVS messages
Coupling Information Of flow
Information of Atomic models
Information of Hierarchic models
If containers have receive, invoke, assign, and reply tags

Figure 4.13: BPEL-to-DEVS transformation

The second aspect of this development is the information contained about interfaces in the accompanying *.wsdl* file. This file is also specified in valid XML format and is mined for DEVS elements of message types, method names and associated input/output parameters and location of Web services and their port types. The mapping is shown in Figure 4.14 below.

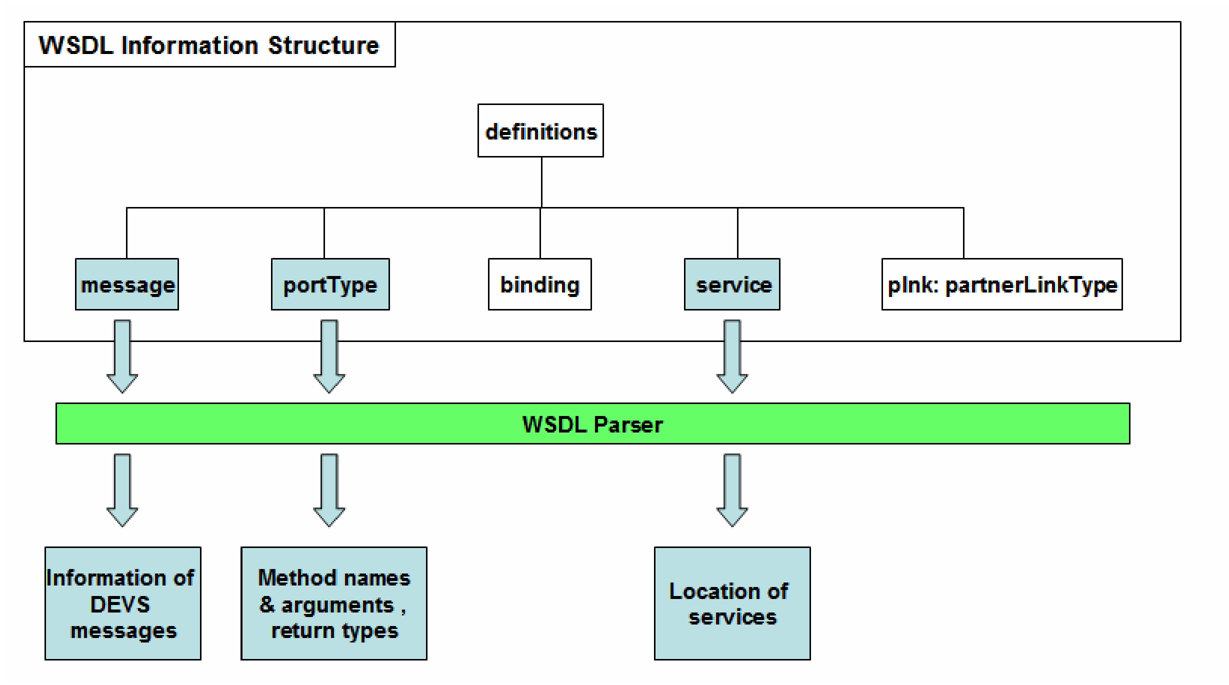
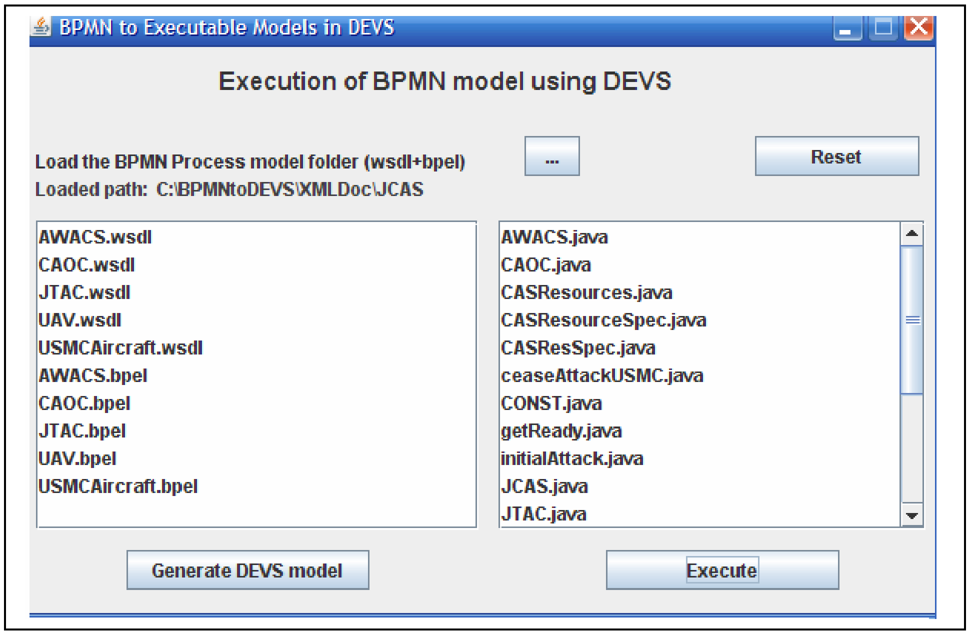

WSDL Information Structure
definitions
message
portType
binding
service
plnk: partnerLinkType
WSDL Parser
Information of DEVS messages
Method names & arguments , return types
Location of services

Figure 4.14: WSDL-to-DEVS transformation

Integration of both of the aspects above led to the development of a tool that takes in a scenario with various .bpel and .wsdl files and transform them into a fully functional DEVS coupled model with operational state machines for atomic models. The contained Web services become the DEVS atomic models and BPEL description produces the coupled models. Figure 4.15 shows the snapshot of the tool that transforms a BPMN scenario (with contained multiple *.bpel* to *.wsdl* file) to an operational DEVS model. A detailed example is presented in Chapter 8.


BPMN to Executable Models in DEVS
Execution of BPMN model using DEVS
...
Reset
Generate DEVS model
Execute

Load the BPMN Process model folder (wsdl+bpel)
Loaded path: C:\BPMNtoDEVS\XMLDoc\JCAS

AWACS.wsdl	AWACS.java
CAOC.wsdl	CAOC.java
JTAC.wsdl	CASResources.java
UAV.wsdl	CASResourceSpec.java
USMCAircraft.wsdl	CASResSpec.java
AWACS.bpel	ceaseAttackUSMC.java
CAOC.bpel	CONST.java
JTAC.bpel	getReady.java
UAV.bpel	initialAttack.java
USMCAircraft.bpel	JCAS.java
	JTAC.java

Figure 4.15: Snapshot of a BPMN-to-DEVS Transformation tool

4.4 Scenario-Based Systems using DoDAF

A recent DoD mandate requires that the DoD Architecture Framework (DoDAF) be adopted to express high level system and operational requirements and architectures [Dod03a]. DoDAF is the basis for the integrated architectures mandated in DOD Instruction 5000.2 [Dod03b] and provides broad levels of specification related to operational, system, and technical views. Integrated architectures are the foundation for interoperability in the joint Capabilities Integration and Development System (JCIDS) prescribed in CJCSI 3170.01D and further described in CJCSI 6212.01D [CJC04, CJC06]. DoDAF and other DoD mandates pose significant challenges to the DoD system and operational architecture development and testing communities since DoDAF specifications must be evaluated to see if they meet requirements and objectives, yet they are not expressed in a form that is amenable to such evaluation. However, DoDAF-compliant system and operational architectures do have the necessary information to construct high-fidelity simulations. Such simulations become, in effect, the executable architectures referred to in the DODAF document. DoDAF is mandated for large procurement projects in the Command and Control domain but its use in relation to M&S is not explicitly mentioned in the documentation [Atk04, Zei05a]. Thus an opportunity has emerged to support the translation of DODAF-compliant architectures into models that are of sufficient fidelity to support architectural evaluation in capable simulation environments. Operational views capture the requirements of the architecture being evaluated and System views provide its technical attributes. Together these views form the basis for semi-automated construction of the needed simulation models.

DoDAF is a framework prescribing high level design artifacts, but leaves open the form in which the views are expressed. A large number of representational languages are candidates for such expression. For example, the Unified Modeling Language, (UML) and Colored Petri Nets (CPN) are widely employed in software development and in systems engineering. Each popular representation has strengths that support specific kinds of objectives and cater to its user community needs. By going to a higher level of abstraction, DoDAF seeks to overcome the plethora of "stove-piped" design models that have emerged. Integration of such legacy models is necessary for two reasons. One is that, as systems, families of systems, and systems-of-systems become more broad and heterogeneous in their capabilities, the problems of integrating design models developed in languages with different syntax and semantics has become a serious bottleneck to progress. The second is that another recent DoD mandate also intended to break down this “stove-piped” culture requires the adoption of the Service Oriented Architecture (SOA) paradigm as supported in the development of Network Centric Enterprise Services (NCES) [DoD05c]. However, anecdotal evidence suggests that a major revision of the DoDAF to support net-centricity is widely considered to be needed. Indeed, under DoD direction, several contractors have begun to design and implement the NCES to support this strategy on Global Information Grid. The result is that system development and testing must align with this mandate – requiring that all systems interoperate in a net-centric environment – a goal that can best be done by having the design languages be subsumed within a more abstract framework that can offer common concepts to relate to. However, as stated before, DoDAF does not provide a formal algorithmically-enabled process to support such integration at higher resolutions. Lacking such processes, DoDAF is inapplicable to the SOA domain and GIG in particular. There have been efforts like [Dan04] that have tried to map DoDAF products to SOA but as it stands out there is no clear-cut methodology to develop an SOA directly from DoDAF, rest aside their testing and evaluation.

4.4.1 DODAF Specifications

The Department of Defense (DoD) Architectural Framework (DoDAF), Version 1.0 (2003), defines a common approach for DoD architecture description development, presentation and integration. The framework enables architecture descriptions to be compared and related across organizational boundaries, including joint and multinational boundaries. DoDAF is an architecture description and it does not define a process to obtain or build the description. The Deskbook [Dod03a] provides one method for development of IT architectures that meet DoDAF requirements, focusing on gathering information and building models required to conduct design and evaluation of architecture. The DoDAF defines three elements for any architecture description, taken from [Dod03a, Zei05a]. These are:

Operational Views (OV)

The OV is a description of the tasks and activities, operational elements, and information exchanges required to accomplish DoD missions. DoD missions include both warfighting missions and business processes. The OV contains graphical and textual products that comprise an identification of the operational nodes[2] and elements, assigned tasks and activities, and information flows required between nodes. It defines the types of information exchanged, the frequency of exchange, which tasks and activities are supported by the information exchanges, and the nature of information exchanges.

System Views (SV)

The SV is a set of graphical and textual products that describes systems and interconnections providing for, or supporting, DoD functions. DoD functions include

[2] Operational Node: A node specified in OV that performs one or more operations. A functional entity that communicates with other functional entity to implement a collective functionality or a capability.

both warfighting and business functions. The SV associates systems resources to the OV. These systems resources support the operational activities and facilitate the exchange of information among operational nodes. Within this view, HOW the functionalities specified in OV will be met is elaborated.

Technical Views (TV)

The Technical view is the minimal set of rules governing the arrangement, interaction, and interdependence of system parts or elements, whose purpose is to ensure that a conformant system satisfies a specified set of requirements. Within this view, the delivery of systems and functionalities is ensured along with their migration strategies towards future standards.

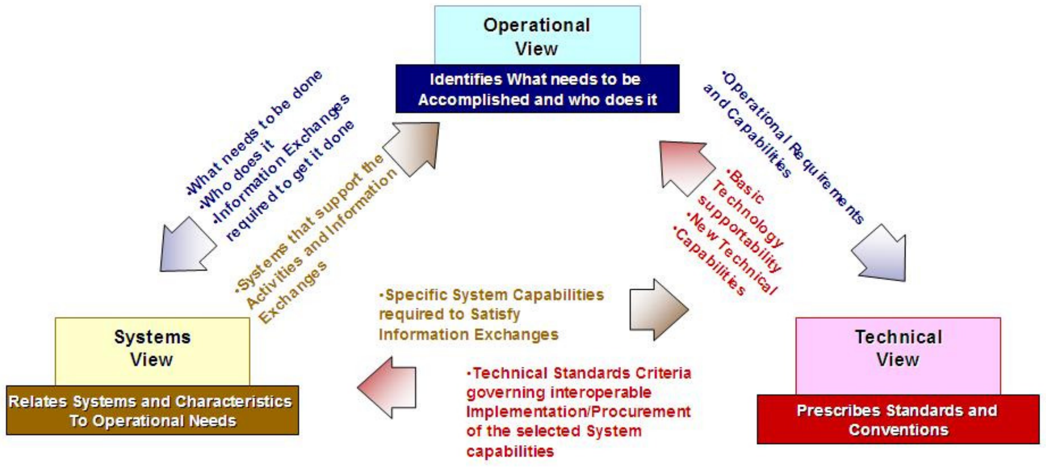

Operational View
Identifies What needs to be Accomplished and who does it
•What needs to be done
•Who does it
•Information Exchanges required to get it done
•Systems that support the Activities and Information Exchanges
•Operational Requirements and Capabilities
•Basic Technology supportability
•New Technical Capabilities
•Specific System Capabilities required to Satisfy Information Exchanges
•Technical Standards Criteria governing interoperable Implementation/Procurement of the selected System capabilities
Systems View
Relates Systems and Characteristics To Operational Needs
Technical View
Prescribes Standards and Conventions

Figure 4.16: Linkages among Views

These views provide three different perspectives for looking at an architecture. The emphasis of DoDAF lies in establishing the relationship between these three elements ensuring entity relationships and supporting analysis (Figure 4.16). The DoDAF approach is essentially data-centric rather than product-centric. The OV, SV and TV are further broken down into specialized views whose brief description can be seen in column 3 in Table 4.4 ahead.

Another way to look at it is through this pyramid (Figure 4.17), which provides the contribution of this research effort, that is, incorporating DEVS M&S is an integral part of design and evaluation cycle based on requirement specifications at the top of the pyramid. The Execution roadmap is as follows:

- Define mission capabilities
- Identify mission threads
- Decompose into Activities and info needs
- Perform M&S based design evaluation,
    - Identification of scenarios (Exp. Frames)
    - Identification of Interfaces for KIP
    - Simulation based on KPP
- Using **model-continuity** identify systems
- Evaluate performance based and do calibration based on simulation
- results of KPP in step 4

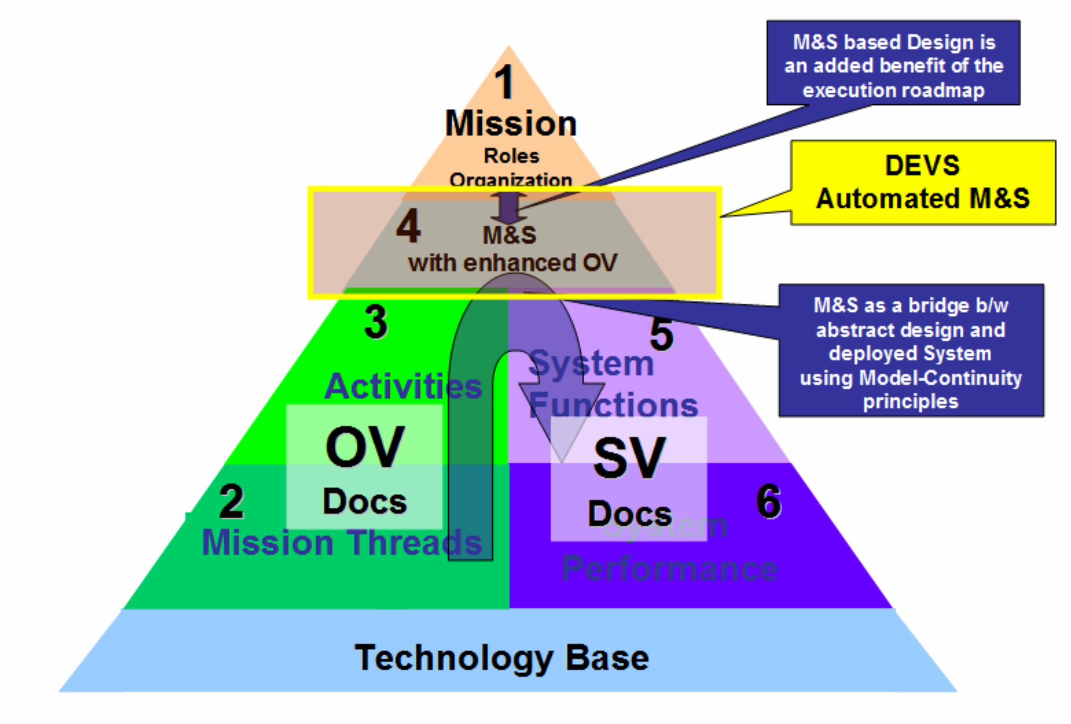


Figure 4.17: DoDAF/DEVS execution roadmap

4.4.2 Motivation for DoDAF-to-DEVS mapping

The DoDAF suffers from following shortcomings:

1. Although there is mention of ‘Executable architectures’ in DoDAF, there is no methodology recommended by DoDAF that would facilitate the development of executable DoDAF models.
2. It has completely overlooked the Model-driven Development approach. Consequently, there is no formal M&S theory that DoDAF mandates.
3. DoDAF fails to address performance issues at OV level
4. DoDAF fails to include measures of effectiveness (MoEs) that can be evaluated at OV stage. If at all any performance measures are considered, they are at System

View level. System parameters and performance is at a totally different resolution than MoEs.

5. There is no mechanism to perform Verification and Validation (V&V) at OV stage
6. It fails to address M&S as potent evaluation and acquisition tool.

We propose a mapping of DoDAF architectures into a computational environment that incorporates dynamical systems theory and a modeling and simulation (M&S) framework. The methodology will support complex information systems specification and evaluation using advanced simulation capabilities. Specifically, the Discrete Event System Specification (DEVS) formalism will provide the basis for the computational environment with the systems theory and M&S attributes necessary for design modeling and evaluation. We will see in the forthcoming sections that the proposed mapping will require augmentation of current DoDAF with more information set that is far from any duplication of the available DoDAF products. We will demonstrate how this information is added and harnessed from the available DoDAF products towards development of an extended DoDAF integrated architecture that is "Executable". This kind augmentation has been attempted earlier by [Lee05] that used CORE® of the Vitech Co. as a tool to develop the executable architecture. They developed 'architectural templates' that elicit information for both the Operational and System views that contained additional information than the usual DoDAF products. In another effort [Ros04] the authors have proposed a new model called Rosen-Parenti model that adds another layer of abstraction to the existing DoDAF, augmenting the model with various user-oriented perspectives.

They further led on to develop the executable architecture with their proposed model and how V&V is applicable in their domain. Their model unearthed the shortcoming of DoDAF that it fails to address the performance issue at OV level, which their model address in one of their perspectives. In our attempt to augment the current DoDAF our focus shall remain to add minimal information that would enable DoDAF to become the executable architecture. There are potential advantages of making DoDAF, a DEVS compliant system.

We seek to employ the DoDAF-to-DEVS mapping to unify multiple model representations by expressing their high-level features within DoDAF and their detailed features as sub-classes of DEVS specifications. DEVS has been shown to be a universal embedding formalism, in the sense of being able to express any sub-class of discrete event systems, such as Petri Nets, Cellular Automata, and Generalized Markov Chains [Zei00]. DEVS has also been employed to express a wide variety of more restricted formalisms, such as state machines, workflow systems, fuzzy logics, and others [Sar01]. Moreover, DEVS environments have a long history of development and are now seeing ever increasing use in the simulation-based design of commercial and military systems [Zei03]. Providing a DoDAF "front end" to a "back end" DEVS environment, will appeal to military information system designers facing the DoDAF and NCES mandates. Such designers will be able to retain their skills with representations familiar to them, while complying with DoDAF abstractions. At the same time they can see the results of their specifications evaluated via simulation-based execution of the model architecture.

Moreover, since all mappings are into subclasses of DEVS, the resulting models can be coupled together and therefore can interoperate at the systems dynamics level. Thus this approach to the synthesis of system design formalisms leverages design and execution methodologies that are already used, or mandated for use, in commercial and military applications.

As a result of recent advances, DEVS can support model continuity through a simulation-based development and testing life-cycle [Hux05]. This means that the mapping of high-level DoDAF specifications into lower-level DEVS formalizations would enable such specifications to be thoroughly tested in virtual simulation environments before being easily and consistently transitions to operate in real environment for further testing and fielding.

4.4.3 From OV-6 UML diagrams to DEVS component behavior specifications

Figure 4.18 below describes the development of DEVS description model from a simple Time-sequencing thread in a Time-sequencing diagram. It must be indicated here that OV diagrams are essentially drawn using UML so we are thereby developing a methodology to transform UML diagrams to DEVS specifications

A simple timing-sequence diagram is considered to illustrate the DEVS Activity component development process and how it fits into the DEVS description of an

Operational node. Consider that a hierarchical activity is being addressed by three Operational nodes and they are exchanging events between sub-activities in order to perform this activity. In the first diagram in Figure 4.18 (leftmost), we can see them interacting with each other. The center part of the figure consists of the thread for one Operational node and is enlarged for better analysis. The sequencing diagram in represented in UML notation and this node has a life-line during the course of which it receives events and sends output messages or events to other nodes.

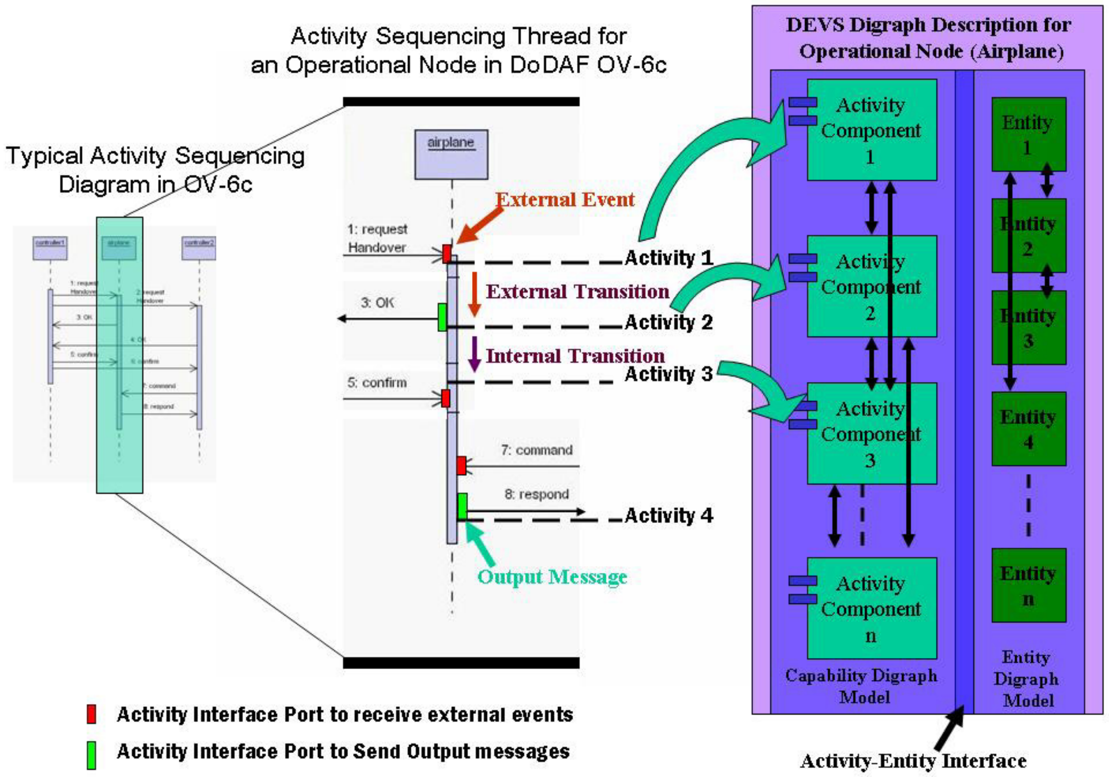


Figure 4.18: Development of DEVS Description model from UML Timing-Sequence Thread

In mapping to DEVS formalism we need to have information about the internal transitions (when no events are received) from one activity to another activity and the External transitions (when an event is received at this node sent by other node). The time-line of the node consists of sequence of activities which the node will undergo in the event of external transition or internal transition. The complete timeline is available in OV-6b, so there is all the more reason to maintain consistency and similar input and output trajectories of sequential activities. Different markings on the thread are self-explanatory. Red boxes indicate the port interfaces where an external event can be received and green boxes indicate the port from which output events can be sent to other nodes. Activity 1 receives an external event and undergoes Activity 2 after generating an output message. Activity 2 undergoes internal transition towards Activity 3 in absence of any external event. This particular thread displays only a subset of activities performed by this node. Since DEVS employs port-based component structure system we identify the input and output ports and assign them to specific activity components at this particular developmental stage. This results in introduction of a new OV document OV-8 that contains the mapping of ports and Activity components. Finally, these activities, if not present in OV-6b, are then introduced in OV-6b for a comprehensive set of activities performed by this Operational node. Another byproduct of this stage is the mapping of Activity components with Entity components that constitute an Operational node. This is specified in a new OV document called as OV-9. This contains information about the Activity-ports, Activity components, Entity components and Entity-ports. Introduction of these new OV documents modifies the overall DoDAF OV specification structure that is illustrated in Figure 4.18.

DoDAF-to-DEVS Elements

The power of UML can not be ignored. UML has matured to great levels and has become integral part of any model based system design. Even the most complex and encompassing DoD Architecture Framework lends itself to UML in description of its various artifacts. However, the bridge to develop executable code from imprecise UML constructs is under research. Executable UML with the aid of Action Semantic approach is one effort that brings to light this important gap of "Model to Code" directly. DEVS with its advanced Model-Continuity process provides this capability readily. Further, the process of developing Executable architectures from object-oriented designs is in place [Wag02] but it has not being explored rigorously for testing software architectures.

As capable as DEVS M&S framework is, it is still not in the mainstream industrial software system design and planning. The ideal progress path now is the development of a mechanism to employ DEVS M&S with UML based developmental methodology. Below is a mapping of UML with the DEVS Elements that provides just the same. The table below is reproduced from our recent work [Mit06a] that involved the artifacts of DoDAF as well. Representation of DoDAF into corresponding UML has been presented earlier by Telelogic [Tel04]. To evaluate a complete example on the implementation of this table refer [Mit06a].

DoDAF Elements			**UML Elements**	**DEVS Elements (generated using XML)**	
	Name	**Description**			
Operational View	OV-1	Top-level Operational View	• Use-case Diagrams	• Activity Component identification • Top level entity structure	DEVS Model Repository
	OV-5	Operational Activity Model	• Use-case • Activity-Sequencing Diagrams • Data-Flow Diagrams	• Activity Component updating • Hierarchical organization of activities • Input-output pairs • Port Identification	
	OV-6	Operational Timing and Sequencing Diagrams	• Timing-Sequencing Diagrams • State-machine Diagrams	• DEVS Atomic Model Creation (Initialize Function, internal and external, transition functions, time advance and output functions) for Activity Components • Entity identification • Acitivy-Entity component mapping	
	OV-2	Operational Node Connectivity	• Composite Structure diagrams	• Coupling Information • Hierarchical component organization	
	OV-8	Activity Component Description	• Composite Structure diagrams • Statecharts	• Activity Component update • Activity port identification and refinement	
	OV-3	Operational Information Matrix		• Input-Output Transaction Pairs • Message formats • Activity Interface & Coupling information	DEVS System-Test Suite

	OV-9	Activity Interface Specifications	• Statecharts • Composite Structure diagrams	• Acitvity-Entity Interface • Entity structure refinement • Activity-Entity port mapping and refinement	🟩
	OV-7	Logical Data Model	• Packages (only for xUML) • Class diagrams	• Entity identification • Hierarchical Structure	🟩
	OV-4	Organizational Relationship Chart	• Class diagrams	• Entity identification • Hierarchical entity structure	DEVS Model Repository
System View	SV-4	System Functional Description	• Use-case Description • Activity Sequencing diagrams	• Hierarchical functional components organization	DEVS Model Repository
	SV-5	System Functional Traceability Matrix (Based on OV-5)		• Coupling Info Refinement	
	SV-10	System State Description and Event Trace (based on OV-6)	• Sequence Diagrams • Statecharts	• DEVS atomic model transition functions refinement	
	SV-6	System Data-Exchange Matrix		• Input-Output pair refinement	
	SV-1	System Interface Description (based on OV-2)	• Composite Structure diagram	• Port assignment Refinement • Entity refinement	
	SV-2	System Communication Description	• Deployment Diagrams	• Coupling Info Refinement (hierarchical management)	
	SV-7	System Performance Parameters Matrix		• Experimental Frame	DEVS System-Test Suite

	SV-3	System-Systems Matrix		• Hierarchical model organization • Entity refinement	DEVS Model Repository
	SV-11	Physical Schema	Class diagrams	• Hierarchical Model organization	
Technical View	TV-1	Current Standards	• Timing Response	• Basic DEVS model for COTS component	
	TV-2	Future Standards		• Improved DEVS model for desired Functionality	

Table 4.4: Mapping of DoDAF with UML and DEVS M&S Elements

4.4.4 Representing DoDAF within the System Entity Structure: Multiple Aspects

The System Entity Structure (SES) is a high level ontology framework targeted to modeling, simulation, systems design and engineering. Its expressive power, both in strength and limitation, derive from that domain of discourse. An SES is a formal structure governed by a small number of axioms that provide clarity and rigor to its models. The structure supports hierarchical and modular compositions allowing large complex structures to be built in stepwise fashion from smaller, simpler ones. Tools have been developed to transform SESs back and forth to XML allowing many operations to be specified in either SES directly or in its XML guise. The axioms and functionally based semantics of the SES promote pragmatic design and are easily understandable by data modelers. Together with the availability of appropriate tool support, this makes development of XML Schema transparent to the modeler. Finally, SES structures are compact relative to equivalent Schema and automatically generate associated executable simulation models.

Figure 4.19 shows the various DoD AF views map into the SES framework. Operational and System perspectives are considered two different decompositions of the system under consideration. They are represented by corresponding nodes called aspects labeled by the names, OperationalView and SystemView, respectively.. The OperationalView aspect has entities labeled opNodes (operational nodes) and activities. The various operational views of DoD AF (other than OV-4) are easily interpreted as describing the entities and their interactions. Likewise, the SystemView aspect has entities labeled functions with DoD AF views that are associated with the functions and their interactions. The one exception is SV-5 which is a relation between the functions of the SystemView and the activities of the OperationalView. This view describes how the activities are implemented via executable functions supplied by the system. To accommodate OV-4 we have added another aspect, the OrganizationalAspect, which represents the decomposition of the system into the roles played by participating personnel.

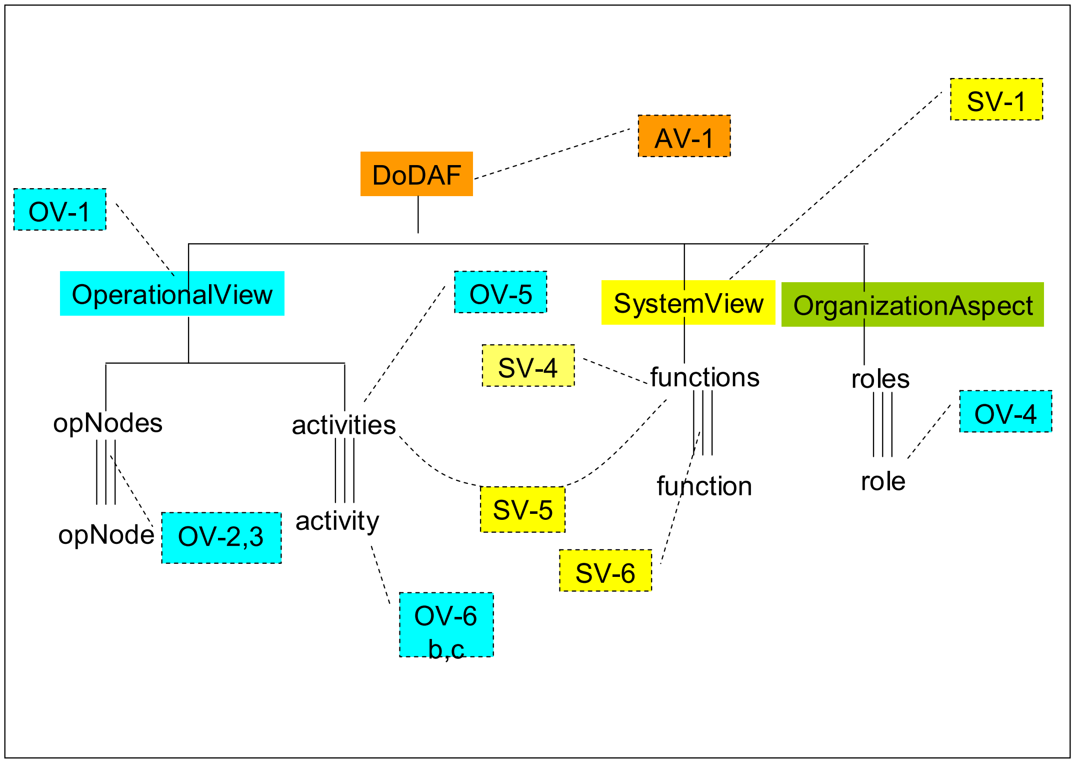


Figure 4.19: Representing DoD AF within the SES framework

4.4.5 Deriving testable behaviors from DoDAF specification

So far the SES has been shown to provide a means of pigeon-holing the various DoD AF views. The power of this representation however lies in the support it provides for deriving system behaviors that can be transferred in semi-automated fashion to executable test federations. The System View is further refined by explicitly adding messages as entities to it. For simplicity, components represent both the functions and their decomposition into services. The coupling associated with the components aspect specifies how messages are routed among the components. This information is what is required to automatically map the System View to a simulation model, that is, in this case, a test federation. To obtain such information, we develop a process for deriving it from specifications associated with the Operational View and mappings between the Operational View elements and their realizations in the System View. If an opNode is engaged in an activity which requires a certain information exchange and the opNode is mapped to a component that executes the function implementing the activity, then this component must be observed to receive and send messages associated with that information exchange.

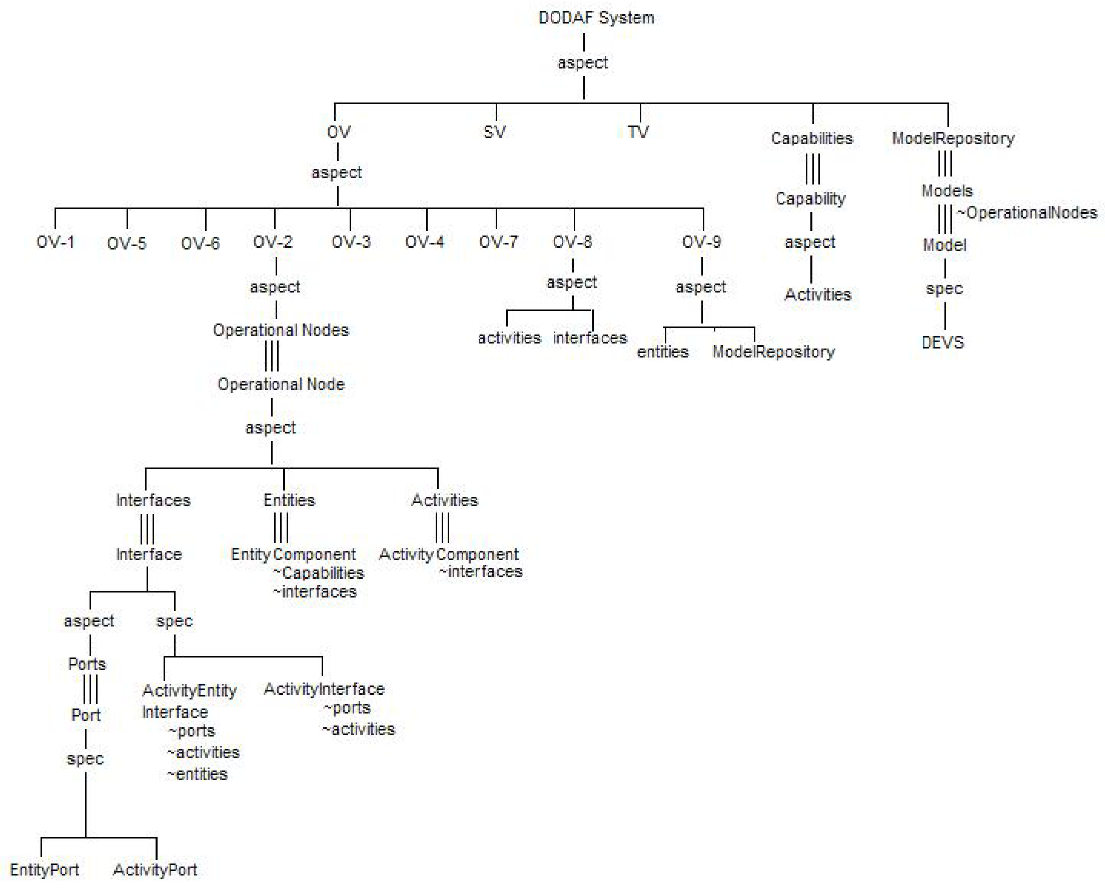


Figure 4.20: SES for enhanced DoDAF with a focus on OV

Information technology–based systems of the future will be increasingly complex with participants across the globe communicating through disparate channels. Interoperability is very much in question. Scalability and fault-tolerance issues have to be addressed. Capabilities have to be satisfied and reliability has to be ensured. Any large system that DoDAF specification documents intend to build has to realize these important facets of architecture design. Modeling and simulation with its model-continuity principles is fast becoming an accepted method of evaluating design principles ensuring accountability to various components within the system. DoDAF has completely overlooked M&S as a possible means to evaluate design, capabilities, and planned expansion of current architectures. There is no provision for testing the constructed system, either in OV or in SV. The ability to configure systems for optimum performance is not allowed in the current DoDAF specification document.

We have introduced two new operational views, OV-8 and OV-9 that add features to enable M&S of the system under design. More details can be found in [Mit06a]. We have also demonstrated how these new documents will be created from the existing Operational Views. We aim to provide structure to the OV process by shifting the perspective from describing functionality as an activity to an Activity-component with definite interfaces to other Activity components as well as identified entities within an Operational node. To what extent an Operational node is decomposable is a subject requiring further research. We have developed a testing process for defined capabilities (that were defined during the conceptual design process in OV-5) and ways in which various rules and doctrines (in OV-6a) can be evaluated for interoperability with different capabilities. By purview of the information contained in OV-9 we have introduced the *model repository* as an important aspect of DoDAF system specification that enhances the DoDAF by making way for M&S activity. Figure 4.20 shows the *system entity structure* (SES) snapshot of the enhanced DoDAF with focus toward the Operational Views. Table 4.4 provides the mapping of various DoDAF OV products into DEVS modeling constructs. UML is chosen as the preferred way of DoDAF representation. First the UML element is mapped with the DoDAF product document and then the same UML element is mapped to the DEVS element(s).

Their representation included SV products as well. In the Table above we have also incorporated the two new OV products i.e. OV-8 and OV-9. Since UML is essentially an Object-oriented methodology, work has been attempted in the area of transforming UML models to CPN executable architectures [Wag02]. Our work is similar in nature, where UML elements are transformed to DEVS elements. The last column links the DEVS elements to Figure 3 and 4 by categorizing them into Model Repository and Semi-automated test-suite elements.

DEVS modeled systems are inherently Object-oriented and DoDAF at the OV stage does not have full expressiveness to be transformed to an executable model. In one of our other systems engineering approaches using System Entity Structure (SES), we developed a hierarchical perspective representation that would enable DEVS to step into at various levels of resolutions. The three main perspectives are

1. Component-based,
2. Capability-based, and
3. Rule-Based.

DEVS Bifurcated model continuity-based system requires all three perspectives to be available in order for the system-model be deployable. As you can see in the Table 4.5 below, the current DoDAF if enhanced with the new OV documents, does make the DoDAF a DEVS compliant system.

Artifact	**SES Elements**	**Current DoDAF**	**Enhanced DoDAF**	**Can DEVS model be created ?**
Tree 1 (Component Perspective)	Entities	OV-2 (Operational nodes) SV-4 (Systems identification)		Too early!
	Hierarchical entity construction	OV-2, OV-7,	OV-9 (no mechanism to provide information of hierarchical formation in current OVs)	YES (only the skeleton with well-formed Coupled models)
	Specified entity-based constraints	SV-7	OV-9 (Hierarchical node descriptions help localize contraints at OV design phase)	NO (information missing to develop behavior models)
Tree 2 (Capability Perspective)	Capabilities	OV1,5,6b, SV-4		NO (no Activity-components defined)
	Hierarchical Activities	OV-6, 6b,c, SV-5		NO (no Activity-components defined)
	Activity-based parameters	ABSENT	OV-8 Activity as Activity-Components definitions based on OV-5,6b) Documenting procedure has place-holders for Parameters	YES (DEVS Capability skeleton can be created with hierarchical Activity composition with defined interfaces)

			and constraints identification) See [44].	
	Activity-based IE	OV-5,6b	OV-8 (may be redundant here)	YES
	Activity-based ROE	OV-6a	OV-8	YES
Tree 3 (Rule perspective)	Rule Hierarchy	OV-6a		YES (ATC-Gen project [9, 40])
	Rule-Activity mapping	ABSENT	OV-8 (the whole purpose of OV-8 is realized here)	YES with full behavior for (Capability Testing)
	Rule-Entity mapping	ABSENT (partially in OV-6a)	OV-9 (the whole purpose of OV-9 is realized here)	YES with full behavior for (System Testing)

Table 4.5: Summarizing the contribution of OV-8, 9 to DEVS M&S

We have also introduced two new Operational Views OV-8 and OV-9 to address the additional information that is needed to make the DoDAF M&S compatible. We have also demonstrated the process to create OV-8 and OV-9 from the existing Operational Views [Mit06a]. OV-8 contains the information about the Activity Component structure and how different Activities are interfaced with each other using the specified logical interfaces. OV-9 contains information about the constituent components inside an Operational Node and its corresponding DEVS model structure along with their mapping to the Activity components in OV-8. Together OV-8 and OV-9 provide a means to correlate Activity Components with accountable entities in an Operational node using logical interfaces. It is after the transformation of OV-8 and OV-9 into DEVS models that rules assigned to specific Activity or Entity components makes OV-8,9 server their complete purpose. Automation using XML and simulation-tuning are important concepts that can be well executed and performed under current DEVS technology. Composing simulations that are hierarchically stable and realizable is a step forward in evaluation of multi-resolutional architectures. Issues like personnel management and task assignment at proper resolution of architectural execution are worth exploring further in future work. Capability to objectify parameters and visualize them with respect to end goal in mind is critical for success. Current DEVS technology is well equipped to accomplish such a capability.

4.5 Synopsis

Referring back to the basic Figure 1.1, the content of this chapter gives way to Figure 4.21 wherein technology and theory is developed to transform various methods of requirement specifications into detailed DEVS operational models. The generalized Bifurcated Model-Continuity based process is now transformed to DEVS-Based Bifurcated Model-Continuity process with requirements specified in various formats.

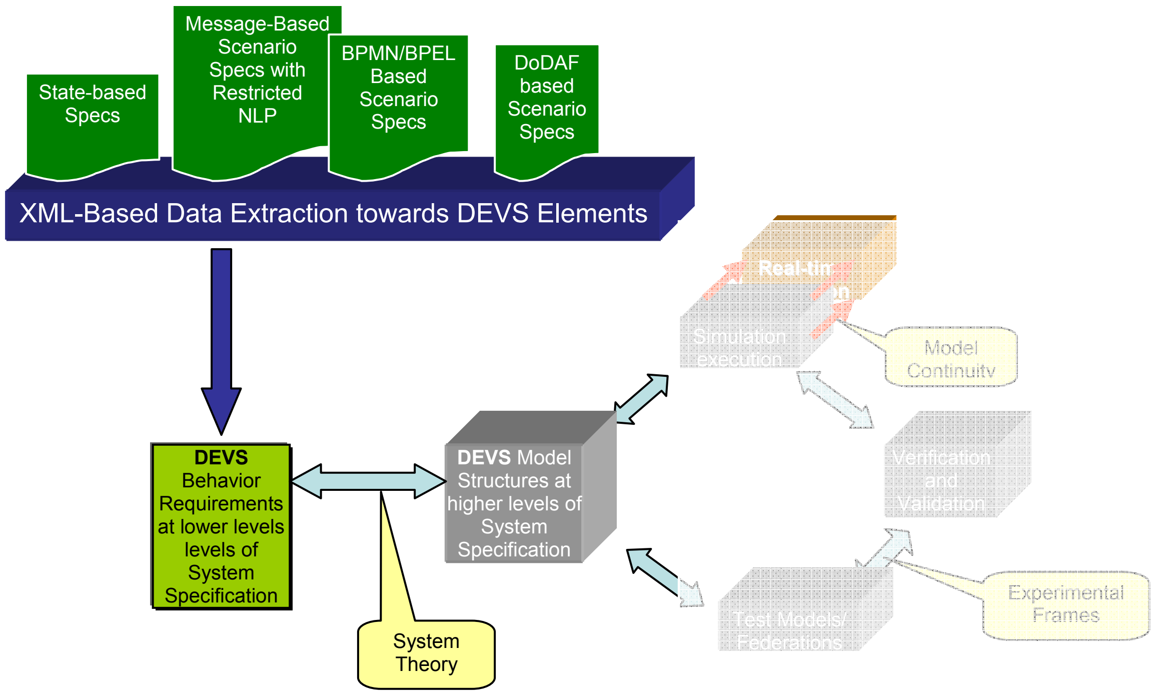

State-based Specs
Message-Based Scenario Specs with Restricted NLP
BPMN/BPEL Based Scenario Specs
DoDAF based Scenario Specs
XML-Based Data Extraction towards DEVS Elements
Real-time execution
Simulation execution
Model Continuity
DEVS Behavior Requirements at lower levels levels of System Specification
DEVS Model Structures at higher levels of System Specification
Verification and Validation
Test Models/ Federations
Experimental Frames
System Theory

Figure 4.21: DEVS Model generation from various types of Requirement Specifications

CHAPTER 5: AUTOMATED MODEL-BASED TEST CASE GENERATION

As detailed in recent DoD reports [Nap97, Nap02], when modeling and simulation is properly used, it provides assistance to formulate system capabilities, compares the cost/benefit ratios of various alternative designs and evaluates their projected effectiveness. In this paper, we discuss an automated testing framework based on Discrete Event System Specification (DEVS) modeling and simulation formalism, Extensible Markup Language (XML), and System Entity Structure (SES), being introduced at DoD's Joint Interoperability Test Command (JITC) for interoperability testing. This framework supports the separation of experimentation, models, and simulators. The experimental frames are developed to support reusable models and simulators based on the DEVS formalism and dynamic system theory. The hierarchical structures of the models are represented by SES and written in XML format to promote extensibility and interoperability. In order to support the separation of models and simulators in the software development, the Model/Simulator/View/Controller design pattern provides the framework to support model execution and multiple network simulation protocols.

The automated testing framework introduced in this paper is a part of the Automated Test Case Generator (ATC-Gen) research project funded by JITC to support the mission of standards compliance and certification. With the simulation-based acquisition initiative, the test requirements in the simulated environments become challenges. These challenges include how to automate and define the scope, the extent, and the methodology to update conformance testing. The automated testing framework is developed based on three concepts: SES, DEVS, and XML. SES can represent a family of hierarchical DEVS models, and serves as a means of organizing the configuration of a model to be designed, which is extracted from a pruning process. Pruning reduces the number of probable models to meet the system requirement. In the automated testing framework, the minimal testable I/O pair and the test model are represented by Pruned Entity Structures (PES). The test models obtained via PES are in executable form. XML uses elements to break up the test model into hierarchical form, and it can be used to represent the SES hierarchical structure. PES is directly mapped into XML, and the three SES modes become XML elements. XML-PES offers simplicity, extensibility, and interoperability. The test models are represented in XML-PES, which can be transformed into DEVS C++ source code.

5.1 Automated Test Case Generator: Concept

ATC-Gen is composed of several stages that are developed in conjunction with DEVS formalism. It applies DEVS to the formalization of Military Standard (MIL-STD) 6016C. The MIL-STD is written in natural language, and can be formalized into the system theory framework by putting a set of requirements in the natural language. By combining system theory and DEVS, the formalization can be transformed into an executable simulation model, and the model can be implemented for testing. By using software tools and modeling packages, the test model can be derived and generated from the natural language. Then, these test models are transformed into an executable format and deployed by the Test Driver to perform testing on the SUT. The processes described above become automated testing. Figure 5.1 illustrates the four stages of the ATC-Gen development.

The first stage is Rule Capturing, which captures and formalizes the MIL-STD 6016C in XML format. The military standard is written in the form of natural language, but do not support the systematic study of large-scale intelligent system. By translating the MIL-STD to a constrained form of natural language that is used in describing system behavior, analysis will be easier. Natural language statements, such as "IF, THEN" used in knowledge-based expert system and artificial intelligence will be suitable to describe the system behavior. The disadvantage of the natural language statement is that it is incapable of describing the time behavior of the system. It can be overcome by using a finite state machine which will be described in stage 3. Capturing requires analysts to read and interpret the standard. Formalizing requires the analysts to identity ambiguous requirements and extracts the state variables and rules. The rules are written in the "If, Then" format as Figure 5.2, and these rules are not associated with time.

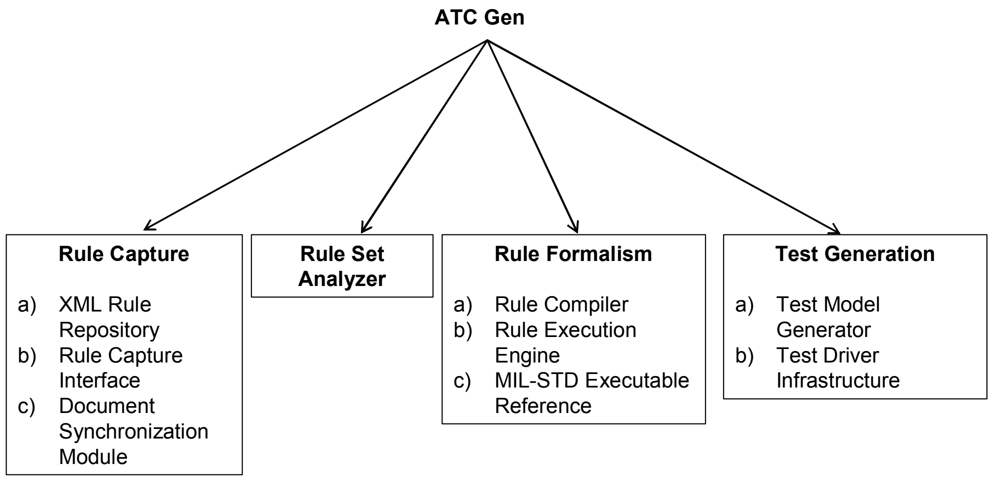


Figure 5.1: ATC-Gen Development

If X is true,

Then do action Y later

Figure 5.2: IF-THEN rule format

To illustrate this, let us consider a simple system consisting of a vending machine and a customer. The vending machine is in idle state if there is no customer present. In addition, the vending machine does not dispense any item if the customer does not put correct amount of money. It dispenses an item if the correct amount of money is inserted into the machine. This simple system can be described by three statements without any time reference:

- If the vending machine is idle, there is no customer.
- If the customer doesn't insert the correct amount of money, no item will be dispensed.
- If the customer inserts enough money, an item will be dispensed from the machine.

There are two state variables in the above example: money and item. Money represents the amount of money required to purchase the item, and item represents the product that the customer wishes to get from the machine. The “IF, THEN” statements can be written into the XML format. Tags are created to enhance the structure and identify the relationship in the document, and they are the legal building blocks of the XML document. Each statement in Figure 5.3 is considered as a rule. Each rule is composed of conditions and actions. Conditions and actions can have state variables. The combination of all rules in an example is a rule set. Based on these guidelines, the vending machine example is translated to the XML document. A XML Document Type Definition (DTD) or schema must be created to validate and provide the correct syntax to the XML document.

Stage 2 consists of the Rule Set Analyzer. It employs the Dependency Analyzer (DA) to determine useful relationships among rules. The DA is a DEVS tool and provides a visual display of dependencies, allowing selection of test sequences by the test engineer. The DA uses DTDs specially written for the project to validate the syntax of the XML files. As mentioned briefly above, the DTD ensures the correctness of the XML files before further processing. Once the syntax is validated, all the rule sets in the XML files will be stored in memory. The DA will determine, manipulate and reorganize all the rules and variables, allowing potential dependencies to surface if shared state variables are identified between pairs of rules. Finally, all the rules and variables will be stored in a single new XML file, which will be used when creating test sequences in the next stage.

```
<RuleSet>
        <name>Vending machine example</name>
        <rule name="1">
                <condition txt="If vending machine is idle">
                        <var name="money" varType="currency"/>
                </condition>
                <action txt="no action"/>
        </rule>
        <rule name="2">
                <condition txt="If customer inserts insufficient money">
                        <var name="money" varType="currency"/>
                </condition>
                <action txt="no item is dispensed">
                        <var name="item" varType="String"/>
                </action>
        </rule>
        <rule name="3">
                <condition txt="If customer inserts enough money">
                        <var name="money" varType="currency"/>
                </condition>
                <action txt="dispense item that is chosen by the customer">
                        <var name="item" varType="String"/>
                </action>
        </rule>
</RuleSet>
```

Figure 5.3: XML RuleSet

Stage 3 is Rule Formalization, which consists of selecting and formulating the test sequences; test models are generated from these sequences. The test engineer formulates test sequences in accordance with the structure of the testing requirements, and converts them into executable simulation models. The DA is executed in order to restore the XML files and the rules created at the end of stage 2, producing a file containing all the possible paths through the simulation and the information required to build a visual representation of the rule connections. By invoking the GUI, it displays the rules by level and shows the sequence of rule firing, providing a visual organization of the rules and their interrelationships and allowing the test engineer to examine the paths that are created between rules in order to finds any potential errors. Although the DA shows all the possible paths, an identification of all possible paths is impractical owing to the fact that not all paths are useful. The test engineer manually examines all feasible paths and creates a test case according to the specification and requirement. The test case is the description of the desired SUT behavior in the minimal testable input/output representation. Based on the minimal table I/O pairs, the test model generates the DEVS test model in C++.

Stage 4 is Test Generation, which consists of generating DEVS C++ test models and executing the test models against a real hardware/software system using the Test Driver. The Test Model Generator generates C++ DEVS model in two steps. First, it converts the test cases to XML test models. Second, the XML test models are converted into C++ DEVS model. The Test Driver is an experimental frame which is capable of executing the test model behavior and interacts with and connects to the System Under Test (SUT) via a High-Level Architecture (HLA) or Simple J interface. The Test Driver performs SUT conformance testing by inducing the testable behavior expressed in the models into the SUT and checking the responses for accuracy.

5.2 Automated Testing Methodology

The automated testing approach combines the systems theory, Modeling and Simulation framework, and model-continuity concepts, and applies the Bifurcated Model-Continuity-based Life-cycle Process [Zei05] to the Link 16 conformance testing. In this section, the two processes of the ATC-Gen stage 4 are discussed – Test Model Generator and Test Driver. The overview is provided in Figure 5.4

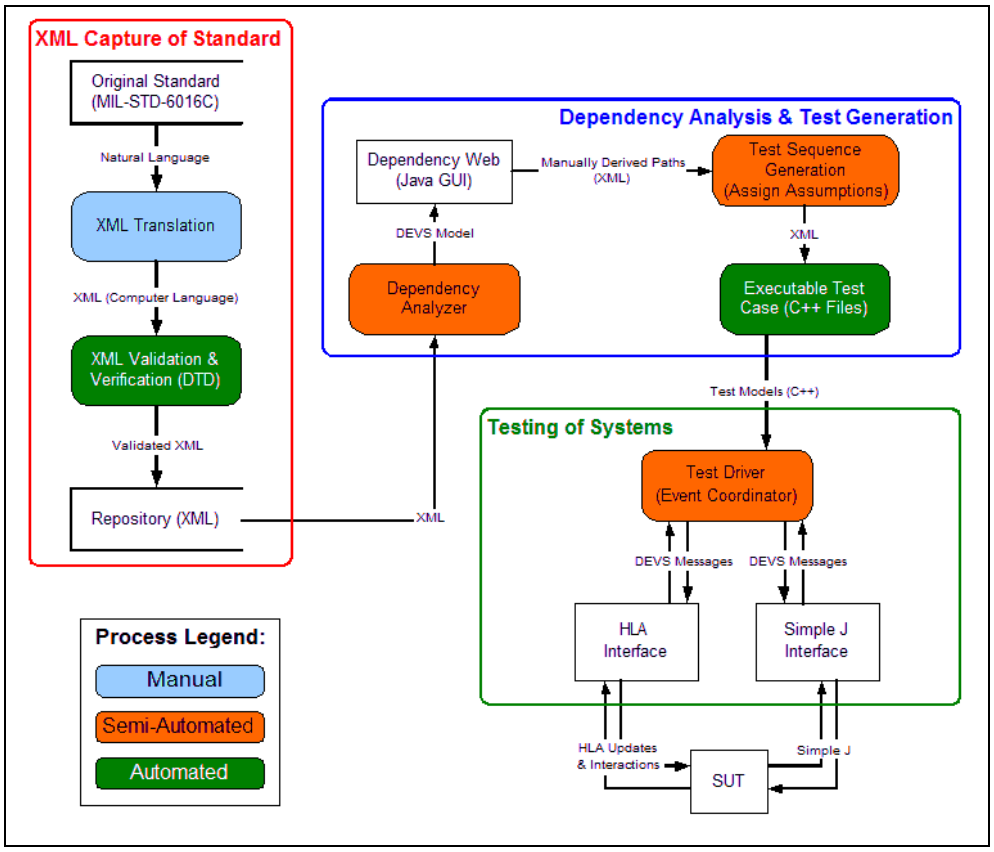

XML Capture of Standard
Original Standard (MIL-STD-6016C)
Natural Language
XML Translation
XML (Computer Language)
XML Validation & Verification (DTD)
Validated XML
Repository (XML)
XML
Dependency Analysis & Test Generation
Dependency Web (Java GUI)
Manually Derived Paths (XML)
Test Sequence Generation (Assign Assumptions)
DEVS Model
XML
Dependency Analyzer
Executable Test Case (C++ Files)
Test Models (C++)
Testing of Systems
Test Driver (Event Coordinator)
DEVS Messages
DEVS Messages
HLA Interface
Simple J Interface
HLA Updates & Interactions
Simple J
SUT
Process Legend:
Manual
Semi-Automated
Automated

Figure 5.4: Overview of ATC-Gen Tool Development

5.2.1 Test Model Generator

The objective of the Test Model Generator is to a create DEVS test models based on minimal testable I/O pairs. In this research effort, we are performing a reachable states study and not generating a complete system behavior. The test scenario is defined in the form of inputs and outputs according to the MIL-STD 6016C definition. The collection of I/O function is infinite in principle because there are numerous states to start from and the inputs can be extended indefinitely. For practice purposes, we restrict our testing focus to messages, and assuming they are the only automatable observables available for testing. These tests are performed against the military hardware/software systems to study its conformity to the MIL-STD.

The DEVS test models are in the form of an experimental frame and allow the Test Driver to perform experiments against the System Under Test. The test engineer analyzes the requirements and creates the test scenarios which describe the behaviors of the SUT based on the MIL-STD 6016C. The requirements are written in minimal testable input/output representation, and the test models are created by applying the model mirroring concept that reverse the minimal testable I/O pairs. Both the minimal testable file and test models are written in XML format and represented by SES, allowing for the transformation between the two XML files. The inputs/output pairs are now represented by three *primitive* atomic models: *holdSend*, *waitReceive*, and *waitNotReceive*. Since the input/output are in sequential order, only one atomic model is active each time, and the rest of the atomic models are passive. In order to try out these test models against the real system, they are converted to software programming source code. This allows quick incorporation of the test models into the Test Driver. Figure 5.5 below illustrates the process of automated test model generation.

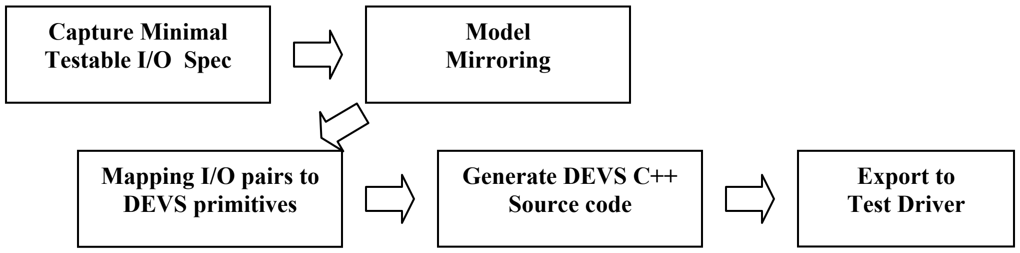

Capture Minimal Testable I/O Spec
Model Mirroring
Mapping I/O pairs to DEVS primitives
Generate DEVS C++ Source code
Export to Test Driver

Figure 5.5: Test Model Generator

5.2.2 Test Driver

The ATC-Gen Test Driver (TD) is an experimental frame designed to perform interoperability testing on TADIL-J systems. The objective of the Test Driver is to execute the DEVS test models generated by the Test Model Generator (TMG). TD emulates a tactical TADIL-J system by providing simulated TADIL-J messages over the simulated tactical communication network, and accepts TADIL-J messages from the SUT to determine the condition of the test model, and is implemented via component-based design using the enhanced Model/Simulator/View/Controller (MSVC) design pattern. The DEVS model is generated by TMG. The TD simulator is a thread derived from the controller that schedules and receives Link-16 messages. The viewer extracts outputs from the simulator, and converts the outputs into a specific middleware format.

Enhanced MSVC Design Pattern

Jim Nutaro [Nut05] demonstrated that the simulator was tuned to the behavior of certain network simulation protocols, and the controller could be rapidly modified to support other protocols. For example, the simulator is associated with HLA time management through the controller in order to pace the execution. The same simulator can be reused by implementing a new controller supporting other network simulation protocols to pace the execution using the wall clock. In this methodology, one controller is associated with one simulator due to the difficulties inherent in handling multiple control strategies and the differing characteristics of the middleware. The simulator is a child thread derived from the controller thread that contains the parameters to influence the simulator. Although Nutaro did not consider using the controller to manage the model operation, his work led to the enhanced MSVC framework, where a new controller is implemented to control the model as well as the simulator.

Figure 5.6 below provides a graphical representation of the enhanced MSVC paradigm [Mit06b]. The functions of the model, simulator, and view are the same as the original MSVC design. A basic controller is implemented to receive the messages via middleware. The specialized controllers are derived from the basic controller to handle message routing to either the simulator or the model. For example, as shown in Figure 5.6, Simple J controller handles the inputs from Simple protocol and controls the DEVS simulator. HLA Controller receives inputs from HLA middleware and controls the model's operations, such as start and stop operations.

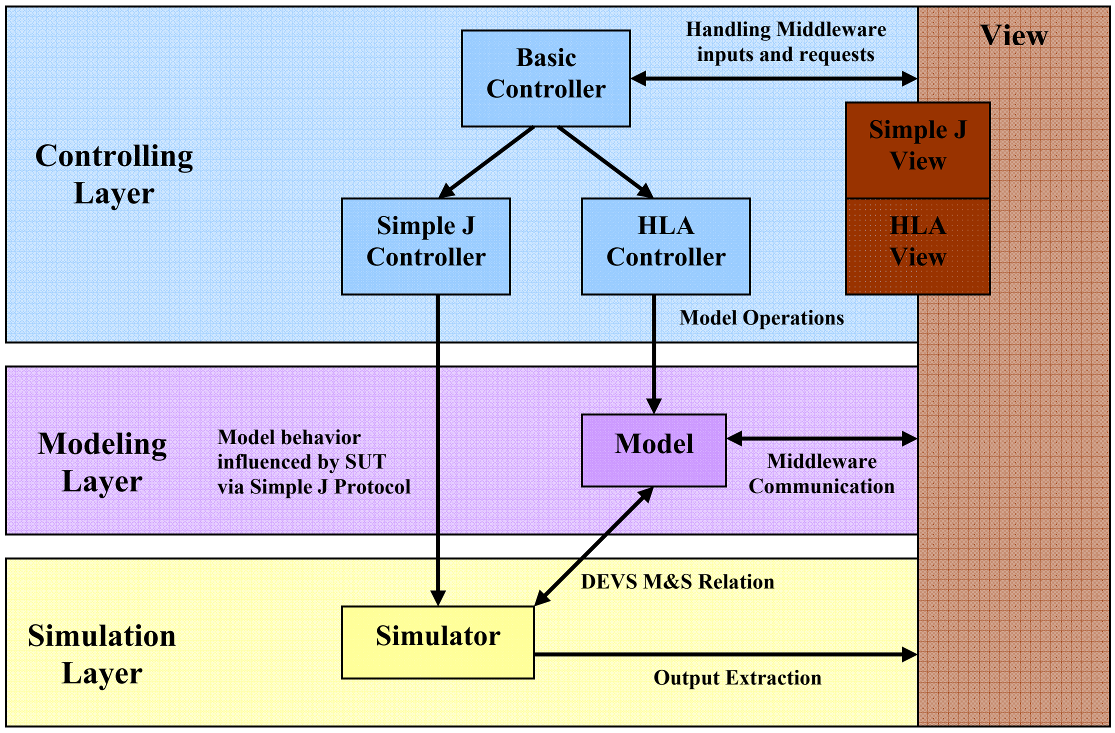


Figure 5.6: Enhanced MSVC paradigm with multiple controllers

It is common for simulation software to support multiple network simulation protocols. In a distributed testing environment, there are combinations of test components, such as simulation software, gateways, and hardware. Each of these components is associated with different network simulation protocols or middleware. A test control manager is often used to control the basic operations of all the test components or hardware, sending operation commands to control the component via a particular middleware. For example, the test control manager synchronizes the time and start/stop of all the test components via HLA, and each component is considered as a federate in an HLA federation.

The Test Driver is implemented based on the enhanced MSVC pattern design. It supports HLA middleware and the Simple J network protocol. The test model is provided by the TMG, and the model behaviors are generated by three atomic models. The view is capable of extracting outputs from the simulator, and provides inputs the basic controller. Model operations are controlled by the HLA controller via HLA middleware, and the simulator is controlled by a Simple J controller.

5.3 Synopsis

A new automated testing approach has been successfully developed using System Entity Structure, the Extensible Markup Language, the Discrete Event System Specification, and the Model/Simulator/View/Controller design pattern. The hierarchical structures of the SUT scenarios and Test models are represented by SES and written in XML format. XML DTDs are developed based on the SES to verify the correctness of the XML files. The processes of automated testing approach are defined as follows:

The SUT scenario is constructed by the test engineer based on the system and the test requirement using the Minimal Testable Input/Output concept.

DEVS test models are developed using the model mirroring by reversing the minimal testable pairs of the SUT.

DEVS programming source codes are generated based on the test models.

The DEVS source codes are implemented into the Test Driver.

Test Driver executes the models and experiments against a real or simulated system.

The automated testing approach is developed to perform conformance testing on the military TADIL-J systems. This approach combines the system theory, the DEVS modeling and simulation framework, and the model continuity concepts to formulate and develop DEVS models. It promotes the separations of models and simulator, which allows model reuse and develops models independently of the simulation engine. The Test models are developed using the system specifications and DEVS framework by collecting the input/output pairs with the initial states and describing the I/O behaviors in DEVS. The simulators are well-defined for reusability and implemented according to the system behavior.

MSVC design pattern used in the Test Driver provides a model for building distribution simulation for the automated testing. MSVC promotes the component-based design and the reusability of the simulation software. By applying this design pattern in conjunction with DEVS modeling and simulation framework, Test models and the simulators are developed separately, and we can attach any network simulation protocols to the simulation. The models are expressed in the DEVS formalism, and the simulators are associated with ADEVS simulation engine to execute the models. The well defined semantics of the DEVS modeling and simulation formalism allows the simulator to be encapsulated and reused. The Test models developed under the automated testing guidelines are able to be executed by the Test Driver.

The automated testing approach was used to verify the conformance of the Integrated Architecture Behavior Model (IABM) to the MIL-STD 6016C, and the results of the test scenarios were validated using the Simple J network packet monitoring tool. Mo re details about the complete research can be seen at [Mak06]. The SUT/Test model method was introduced in this thesis to verify the correctness of the DEVS models. The transmissions and the receipts of the Simple J messages were captured by the packet monitoring tool. The system analyst interpreted and verified the messages, and determined whether these messages were the intended behavior of the Test Driver.

CHAPTER 6: NET-CENTRIC MODEL EXECUTION USING SERVICE ORIENTED ARCHITECTURE

This chapter presents a novel framework known as DEVSML that is built on XML middleware. It provides the capability to develop a portable integration coupled description with complete behavior in XML format that can be simulated either centrally, remotely or in distributed manner. Section 6.1 describes DEVSML that expresses DEVS model with full behavioral representation and provides a novel way to collaborate and share models over the web using Web Services technology. Along with the standardization of DEVS DTDs, a vice-versa DEVSML transformation to DEVS JAVA code is the prime objective of DEVSML. Client and Server side designs of DEVSML framework are described. Section 6.2 builds on DEVSML framework and proposes SOADEVS that provides the simulation engine to execute DEVS models over the web using Simulation Service in SOA. It also describes the underlying design of SOADEVS framework.

6.1 DEVSML: Automating DEVS Execution over SOA Towards Transparent Simulators

DEVS formalism [Zei00] exists in many implementations, primarily in DEVS/C++ and DEVSJAVA [ACI06]. Extensions of these implementations are available as DEVS/HLA [Sar00], DEVS/CORBA [Cho01], cell-DEVS [Wai01], and DEVS/RMI [Zha05]. Since DEVS is inherently based on object oriented methodology, C++ and Java are the chosen programming languages. Almost all of the extensions capitalize on the underlying object orientation provide by these two programming languages. The models are coded either in C++ or Java. DEVS formalism categorically separates the model, the Simulator and the Experimental frame. However, one of the major problems in this kind of mutually exclusively system is that the formalism implementation is itself limited by the underlying programming language. In other words, the model and the simulator exist in the same programming language. Consequently, legacy models as well as models that are available in one implementation are hard to translate from one language to another even though both the implementations are object oriented. Other constraints like libraries inherent in C++ and Java are another source of bottleneck that prevents such interoperability.

The motivation for this work stems from this need of model interoperability between the disparate simulator implementations and provides a means to make the simulator transparent to model execution. We propose DEVS Modeling Language (DEVSML) that is built on eXtensible Markup Language (XML) [Xml] as the preferred means to provide such transparent simulator implementation. The present work has been done with Java and efforts are ongoing in the direction to provide C++ implementation of the concept. This work is built on the JAVAML research done by Vladimir for DEVS Meta Language [Jan06]. While his work aims to provide a stand-alone XML schema for DEVS formalism that can be used by any of programming implementations, research is still ongoing to specify the logic behavior in atomic models. The present work aims to extend his approach and provide complete behavioral support in DEVSML by implementing the proposed universal Atomic and Coupled DTDs. We look forward toward standardization of these DTDs so that models across the web can participate in Dynamic Modeling & Simulation over Net-centric web services.

We have implemented our proposed DTDs in web service architecture; specifically a Service Oriented Architecture (SOA) [Sun] and paper will illustrate the Server as well as Client designs. We also propose modifications in the DEVS formalism as well that will make a DEVS model to be a DEVS Service model that can be readily deployed using Model-continuity principles [Hux03].

6.1.1 Overview of DEVSML

DEVSML is a novel way of writing DEVS models in XML language. This DEVSML is built on JAVAML, which is infact, XML implementation of JAVA. The current development effort of DEVSML takes its power from the underlying JAVAML [Bad05] that is needed to specify the 'behavior' logic of atomic and coupled models. The DEVSML models are transformable back'n forth to java and to DEVSML. It is an attempt to provide interoperability between various models and create dynamic scenarios. The key concept is shown in the Figure 6.1.

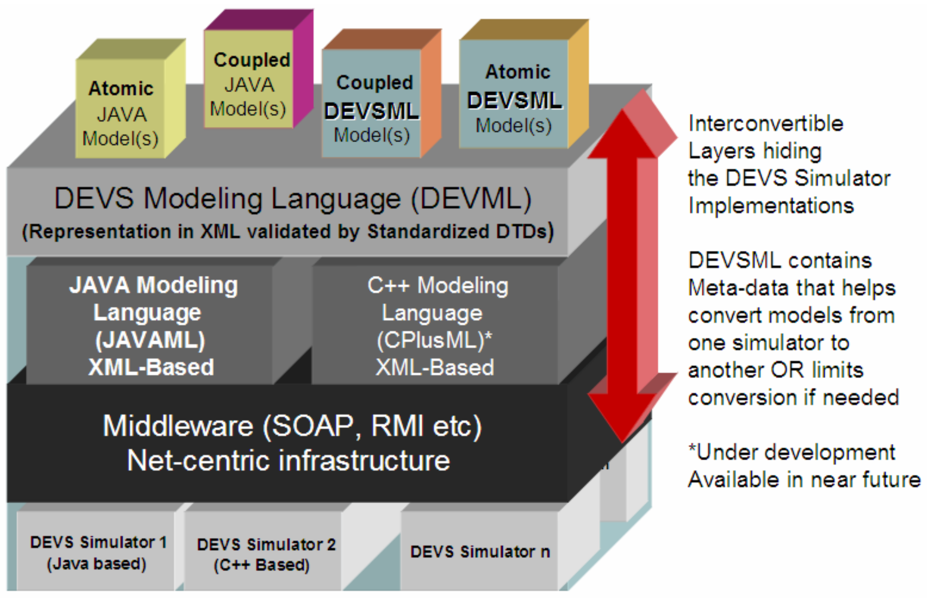

Atomic JAVA Model(s)
Coupled JAVA Model(s)
Coupled DEVSML Model(s)
Atomic DEVSML Model(s)
DEVS Modeling Language (DEVML)
(Representation in XML validated by Standardized DTDs)
JAVA Modeling Language (JAVAML) XML-Based
C++ Modeling Language (CPlusML)* XML-Based
Middleware (SOAP, RMI etc)
Net-centric infrastructure
DEVS Simulator 1 (Java based)
DEVS Simulator 2 (C++ Based)
DEVS Simulator n
Interconvertible Layers hiding the DEVS Simulator Implementations
DEVSML contains Meta-data that helps convert models from one simulator to another OR limits conversion if needed
*Under development Available in near future

Figure 6.1: DEVS Transparency and Net-centric model interoperability using DEVSML

The layered architecture of the said capability is shown in Figure 6.1. At the top is the application layer that contains model in DEVS/JAVA or DEVSML. The second layer is the DEVSML layer itself that provides seamless integration, composition and dynamic scenario construction resulting in portable models in DEVSML that are complete in every respect. These DEVSML models can be ported to any remote location using the net-centric infrastructure and be executed at any remote location. Another major advantage of such capability is total simulator 'transparency'. The simulation engine is totally transparent to model execution over the net-centric infrastructure. The DEVSML model description files in XML contains meta-data information about its compliance with various simulation 'builds' or versions to provide true interoperability between various simulator engine implementations. This has been achieved for at least two independent simulation engines as they have an underlying DEVS protocol to adhere to. This has been made possible with the implementation of a single atomic DTD and a single coupled DTD that validates the DEVSML descriptions generated from these two implementations. Such run-time interoperability provides great advantage when models from different repositories are used to compose bigger coupled models using DEVSML seamless integration capabilities.

Figure 6.2 provides a basic flow chart of operations that can be done with DEVSML framework. The designer can start with either the JAVA code for atomic/coupled model or the DEVSML code for atomic/coupled model. In either of the case, the process has to lead to DEVSML representation of the model. The DEVSML description that is essentially XML is then validated by the standardized DTDs (shown in next section), can now participate in model composition (blue box). The composed coupled model as well as DEVSML atomic model can verily be stored in the Library for reuse. The composed integrated model, that is complete in every respect, as it contains behavior as well, as ready for simulation. The DEVSML model is then sent to various remote locations or specifically Server, wrapped in SOAP message to the destination host (Server in our case). Based on the information contained in the DEVSML model description, corresponding simulator is called for to instantiate the model and executes the simulation with the designated simulator.

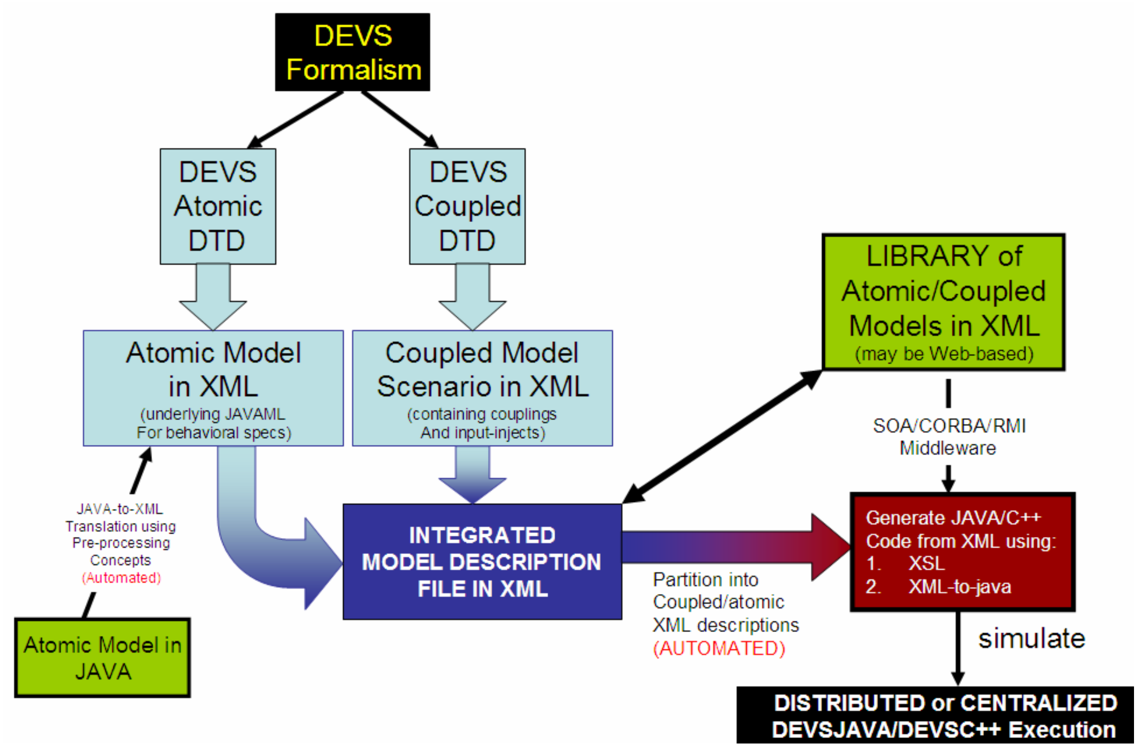

DEVS Formalism
DEVS Atomic DTD
DEVS Coupled DTD
Atomic Model in XML
(underlying JAVAML For behavioral specs)
Coupled Model Scenario in XML
(containing couplings And input-injects)
LIBRARY of Atomic/Coupled Models in XML
(may be Web-based)
SOA/CORBA/RMI Middleware
JAVA-to-XML Translation using Pre-processing Concepts
(Automated)
Atomic Model in JAVA
INTEGRATED MODEL DESCRIPTION FILE IN XML
Partition into Coupled/atomic XML descriptions
(AUTOMATED)
Generate JAVA/C++ Code from XML using:
1. XSL
2. XML-to-java
simulate
DISTRIBUTED or CENTRALIZED DEVSJAVA/DEVSC++ Execution

Figure 6.2: Flow chart of basic operations leading to model composability using DEVSML

Web Services and Interoperability using XML

Service oriented Architecture (SOA) framework is a framework consisting of various W3C standards, in which various computational components are made available as 'services' interacting in an automated manner towards achieving machine-to-machine interoperable interaction over the network. The interface is specified using Web Service Description language (WSDL) [WSD] that contains information about ports, message types, port types, and other relating information for binding two interactions. It is essentially a client server framework, wherein client request a 'service' using SOAP message that is transmitted via HTTP in XML format. A Web service is published by any commercial vendor at a specific URL to be consumed/requested by another commercial application on the Internet. It is designed specifically for machine-to-machine interaction. Both the client and the server encapsulate their message in a SOAP wrapper.

JavaML

JavaML [Bad05] is an XML-Based source code representation for Java programs. The JAVAML Document Type Definition (DTD) specifies various elements of a valid JavaML document. It is well-suited to be used as canonical representation of Java source code for tools. It comes with an XSLT-based back-converter that translates a JavaML document back into java source code. More details about JavaML can be found at [Bad05].

6.1.2 DEVS DTDs and their Standardization

This section provides details about the modified DEVS formalism for the atomic model to make it 'service enabled' in the process of software engineering. The motivation comes from the fact that testing of Web Services as in 'system test suite' is still in infancy and DEVS based testing is still in progress. With a slight modification in the DEVS formalism for atomic model we plan to achieve the following:

Transform any existing DEVS atomic as a container that is capable of publishing services

Promote testing of web service components by making them DEVS enable so that a DEVS wrapper would encapsulate a Service as a 'component'

Transition from a DEVS Service component directly to a web service component after removal of wrapper and deploy it using model-continuity principles.

Figure 6.3 provides a graphical view of an abstract component that inherits the basic functionality of DEVS atomic model. The extended DEVS formalism is specified as below:

$$\mathbf{SM} = \langle \mathbf{X}, \mathbf{S}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \delta_{\mathbf{conf}}, \lambda, \mathbf{ta}, \mathbf{V} \rangle$$

where,

V is the set of Service methods that are represented by this atomic model.

The other symbols have their usual meaning as described in standard notations in [Zei00]. As can be seen in Figure 6.4, we express the DEVS atomic model in XML format. We have structured the atomic component's behavior on the line of Service component. Any Service component provides 'services', which means that, it is implemented as a *method* in the underlying OOP language. We express the new proposed atomic **SM** formalism with a collection set of these services as **V**. We collect these *methods* and store their names in the collection **V** with the intent of producing a WSDL that makes these operational methods 'visible'. This manner of making methods available through WSDL provides two advantages:

The DEVS model could become the actual Service using model-continuity concepts

Each Service, assuming there is only one method that is made visible, is provided a state-machine for its behavior testing in off-line mode.

The approach is under research and will be reported in near future.

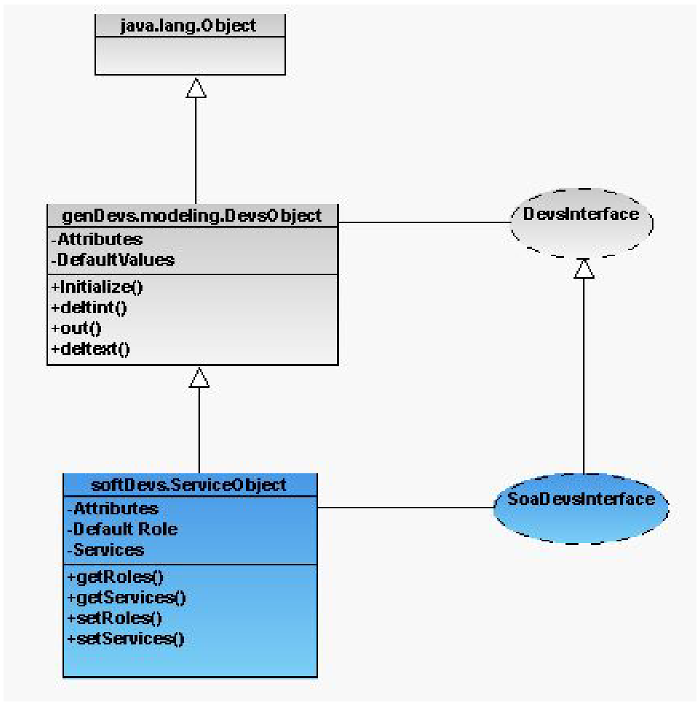


Figure 6.3: an SOA object capable of DEVS modeling

The XML representation of this abstract component is shown in Figure 6.4. The idea here is that a DEVS atomic model contains the behavior of a component that has defined interfaces. The *devsObject* is the wrapper that takes care of $\delta_{\mathbf{int}}$, $\delta_{\mathbf{ext}}$, $\delta_{\mathbf{conf}}$ interactions, while the *serviceObject* presents the services, or methods that are either used individually or in nested manner to implement a published service. Making this change in the DEVS formalism does not change DEVS original formalism. It just introduces a container that contains the name of the methods that could be published as a service. In complex models, it is a common practice to break the use-case into smaller manageable use-cases for implementation purposes. Similarly, implementing complex behaviors and complicated state machines [Mit06a] require the functionality to be organized into methods that are called in the DEVS $\delta_{\mathbf{int}}$, $\delta_{\mathbf{ext}}$, $\delta_{\mathbf{conf}}$ functions. The set **V** keeps an account of such methods that can be made available for service publications. The XML structure of a *serviceObject* is implemented based on the UML diagram in Figure 6.3. What is required here is the addition of code for 'services' tag. Once implemented on SOA, the code with respect to the 'services' tag can be exchanged through a SOAP message and a DEVS model is made ready for simulation.

```
<?xml version="1.0" encoding="UTF-8"?>
<xml-body>
<model>
  <atomic>
     <name>Hello</name>
     <params>  </params>
  <construct>
     <args> </args>
     <ports>
         <inports>
           <inport>in</inport>
           </inports>
         <outports>
           <outport>out</outport>
         </outports>
     </ports>
  </construct>

   <initialize>
  </initialize>
    . .
  <services>
     <function>
        <access> public </access>
           <return> int      </return>
               <inport> in </inport>
               <outport> out </outport>
           <fname> decrement() </fname>
           <logic>    </logic>
     </function>

  </services>
  </atomic>
</model>
</xml-body>
```

Figure 6.4: Automated XML snippet for a DEVS atomic model.

Figure 6.5 shows the DEVSML DTD for extended DEVS formalism that contains the 'services' container. Similarly, Figure 6.6 contains the DTD for DEVS coupled (digraph) model. The coupled model is a hierarchical model that takes into account of the contained atomic or coupled models. Also notice the attribute '**simulator (devsjava|xdevs)**' in the ATTLIST tag for atomic as well as coupled element. This is the meta-data that is stored with every model that is used by server to assign the appropriate simulator for this model. Components within a coupled model could be managed by different simulators. The attribute simulator in the nodes *coupledRef* and *atomicRef* (see Figure 6.6) defines the simulator to use. This attribute is generated when de whole model is integrated in one DEVSML file. Of course, the simulator must comply with the DEVS simulation protocol. The authors call for standardization of both of these DTDs [Mit07e].

```
<!-- DEVS ATOMIC MODEL -->
<!ENTITY % variable-info
      "name CDATA #REQUIRED
       type CDATA #REQUIRED">
<!ELEMENT atomic
(inputs,outputs,states,ta,deltint,deltext,deltcon,lambda,services?,java
-specific?)>
<!ATTLIST atomic
      name ID #REQUIRED
      simulator (devsjava|xdevs) #REQUIRED
      host CDATA #REQUIRED>
<!ELEMENT inputs (port*)>
<!ELEMENT port EMPTY>
<!ATTLIST port
      name CDATA #REQUIRED>
<!ELEMENT states (state*)>
<!ELEMENT state EMPTY>
<!ATTLIST state
      %variable-info;>
<!ELEMENT outputs (port*)>
<!ELEMENT ta (block?)>
<!ELEMENT deltint (block?)>
<!ELEMENT deltext (block?)>
<!ELEMENT deltcon (block?)>
<!ELEMENT lambda (block?)>
<!ELEMENT services (service*)>
<!ELEMENT service (method)>
<!ATTLIST service
      name ID #REQUIRED
      port CDATA #REQUIRED>
<!ELEMENT java-specific (package-decl,import*,constructor*,method*)>
```

```
<!ELEMENT import EMPTY>
```

Figure 6.5: DEVS atomic DTD

```
<!--DEVS COUPLED MODEL-->
<!ENTITY % connection-info
      "component_from CDATA #REQUIRED
       port_from      CDATA #REQUIRED
       component_to   CDATA #REQUIRED
       port_to        CDATA #REQUIRED">
<!ELEMENT devs (scenario,models)>
<!ELEMENT scenario (coupled)>
<!ELEMENT coupled
(inputs,outputs,components,internal_connections,external_input_connecti
ons,external_output_connections,java-source-program)>
<!ATTLIST coupled
      name ID #REQUIRED
      model CDATA #REQUIRED
      simulator (devsjava|xdevs) #REQUIRED
      host CDATA #REQUIRED>
<!ELEMENT inputs (port*)>
<!ELEMENT port EMPTY>
<!ATTLIST port
      name CDATA #REQUIRED>
<!ELEMENT outputs (port*)>
<!ELEMENT components (coupledRef|atomicRef)*>
<!ELEMENT coupledRef (components?)>
<!ATTLIST coupledRef
      name  CDATA #REQUIRED
      model CDATA #REQUIRED
      simulator (devsjava|xdevs) #IMPLIED
      host CDATA #REQUIRED>
<!ELEMENT atomicRef EMPTY>
<!ATTLIST atomicRef
      name  CDATA #REQUIRED
      model CDATA #REQUIRED
      simulator (devsjava|xdevs) #IMPLIED
      host CDATA #REQUIRED>
<!ELEMENT internal_connections (connection*)>
<!ELEMENT external_input_connections (connection*)>
<!ELEMENT external_output_connections (connection*)>
<!ELEMENT connection EMPTY>
<!ATTLIST connection
      %connection-info;>

<!ELEMENT models (model*)>
<!ELEMENT model (java-source-program)>
<!ATTLIST model
      name ID #REQUIRED>
```

Figure 6.6: DEVS coupled DTD

6.1.3 Web Services Architecture for DEVSML

Figure 6.7 shows the designed Web Architecture. At server's end, there are N simulators registered, the WSDL files containing the Web services offered and an Applet for generation and simulation of DEVSML models that uses these Web services. At the client's end, it is possible to use the Applet or an own client program [DML], which makes use of the Web services (in Figure 6.7: CLAPP, Client Application).

Registering a simulator means to enable it so that it can be used according to the defined DEVSML DTDs. This involves the definition of two additional classes that implement the interfaces *InterfaceXmlAtomic* and *InterfaceXmlCoupled* (see Figure 6.7). These classes must generate XML *elements* that define the structure of the specific simulator models according to the atomic and coupled DTDs, These *elements* are inputs, outputs, etc. Efforts are ongoing to develop a template for the user community to register their simulators via a new process in order to make the registration process easier.

Once the simulator is registered, the Web services are available for this simulator. The registry is recommended, since the clients can use any simulator registered at the server.

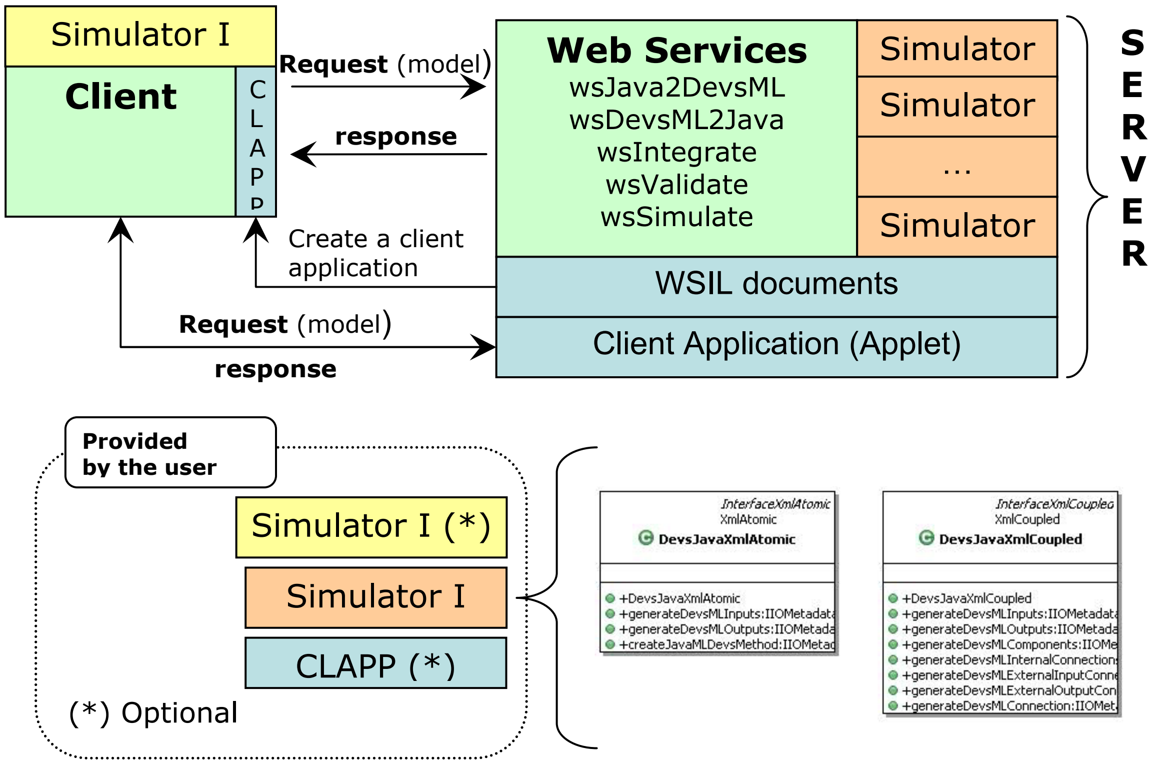


Figure 6.7: Web service Architecture for DEVSML Implementation

The most important Web services offered in our current architecture are:

1. Convert Java models to DEVSML.
2. Convert DEVSML models to Java.
3. Integrate coupled and atomic DEVSML models towards a portable 'Composite' Coupled DEVSML file that can be simulated at any server.
4. Validate an existing DEVSML model.
5. Simulate a Composite Coupled file at the server

Figure 6.8 shows part of the UML diagram of the Applet developed. *xdevs* and *devsjava* classes are directly generated from the Web services since we have these two simulators registered. The rest of the diagram provides the functionality of the Applet. Providing complete details is outside the scope of this article and will be report in our forthcoming publication dedicated to Server and Client designs. Demonstration of these web services is available at [DML] that are hosted at ACIMS www.acims.arizona.edu.

Systems M&S based on DEVS theory [Zei00] and web-based collaborative modeling leading to composite coupled models based on DEVSML has been attempted for the Java programming language. In order to solve the same problem for other programming languages such as C++, C#, ADA, etc., we can choose among different alternatives:

Using JNI: In this case, it is necessary to adapt each simulator to JNI. Therefore, the models must be rewritten into Java. The reason behind this conversion is due to the fact that we need behavior representation in XML. We do have cppML, that is C++ Modeling Language in XML but we want only one behavioral representation in XML. Our preferred way of doing it is through JavaML as Java is better positioned to address the Web Services domain. Using another XML representation more versatile for the behavior of the model. In this case it is possible to use XML definitions defined to represent any object oriented programming language, such as o:XML [OXML] or OOPML [OPML]. This is again a work in progress.

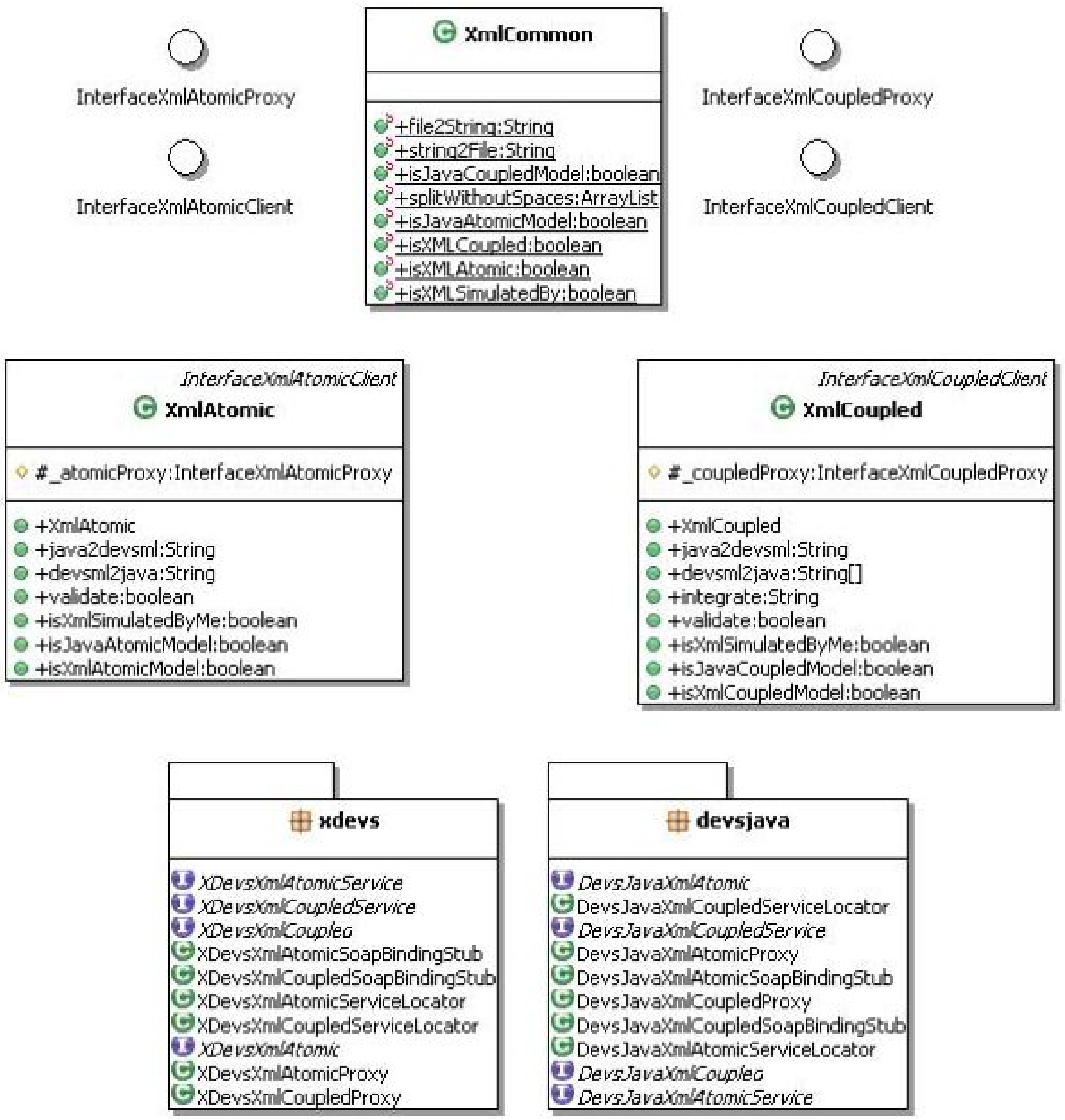

InterfaceXmlAtomicProxy
InterfaceXmlAtomicClient
XmlCommon
+file2String:String
+string2File:String
+isJavaCoupledModel:boolean
+splitWithoutSpaces:ArrayList
+isJavaAtomicModel:boolean
+isXMLCoupled:boolean
+isXMLAtomic:boolean
+isXMLSimulatedBy:boolean
InterfaceXmlCoupledProxy
InterfaceXmlCoupledClient
InterfaceXmlAtomicClient
XmlAtomic
#_atomicProxy:InterfaceXmlAtomicProxy
+XmlAtomic
+java2devsml:String
+devsml2java:String
+validate:boolean
+isXmlSimulatedByMe:boolean
+isJavaAtomicModel:boolean
+isXmlAtomicModel:boolean
InterfaceXmlCoupledClient
XmlCoupled
#_coupledProxy:InterfaceXmlCoupledProxy
+XmlCoupled
+java2devsml:String
+devsml2java:String[]
+integrate:String
+validate:boolean
+isXmlSimulatedByMe:boolean
+isJavaCoupledModel:boolean
+isXmlCoupledModel:boolean
xdevs
XDevsXmlAtomicService
XDevsXmlCoupledService
XDevsXmlCoupled
XDevsXmlAtomicSoapBindingStub
XDevsXmlCoupledSoapBindingStub
XDevsXmlAtomicServiceLocator
XDevsXmlCoupledServiceLocator
XDevsXmlAtomic
XDevsXmlAtomicProxy
XDevsXmlCoupledProxy
devsjava
DevsJavaXmlAtomic
DevsJavaXmlCoupledServiceLocator
DevsJavaXmlCoupledService
DevsJavaXmlAtomicProxy
DevsJavaXmlAtomicSoapBindingStub
DevsJavaXmlCoupledProxy
DevsJavaXmlCoupledSoapBindingStub
DevsJavaXmlAtomicServiceLocator
DevsJavaXmlCoupled
DevsJavaXmlAtomicService

Figure 6.8: Client side implementation using interfaces.

The disadvantage of using one solution or other resides in the interoperability between different simulators executing the same model. Proving interoperability between simulators is what true transparency is. If all the simulators are running under JNI, then adapters must been made in order to change information among them. The current

DEVSML architecture with only one universal underlying atomic DTD and coupled DTD is the first step towards interoperable simulators. Defining a distributed coordinator between these simulators is the second step. If o:XML or OOPML are used, then it is not necessary to define JNI simulators or rewrite models in Java, but what is needed a mechanism to interoperate between different DEVS simulators. How to communicate a simulator written in C++ with a simulator written in Java? Perhaps the solution resides in the definition of standards for the format of the data at the syntactic level.

6.2 SOADEVS: Remote Execution of DEVS using Simulation Service

This section aims to develop and evaluate distributed simulation using the web service technology. After the development of World Wide Web, many efforts in the distributed simulation field have been made for modeling, executing simulation and creating model libraries that can be assembled and executed over WWW. By means of XML and web services technology these efforts have entered upon a new phase.

A prototype simulation framework has been implemented using web services technology. The central point resides in executing the simulator as a web service. The development of this kind of frameworks will help to solve large-scale problems and guarantees interoperability among different networked systems and specifically DEVS-validated models.

Discrete event system specification (DEVS) is one of the most suitable formalisms for the representation of real world systems. Simulating a model involves the implementation of a behavioral model and running it in the simulator. A simulator is defined as a piece of program that executes the model. Our aim is to make the simulation process totally transparent in the model-design cycle. By such capability, the modeler need not focus on the simulator compatibility or any platform issues as in earlier developments like DEVS/C++, DEVSJAVA, DEVS/RMI, DEVS/CORBA and other. Implementing simulation platform as a 'Simulation Service platform' the designer will be able to execute the model over Internet through web services, using SOA as the communication protocol. In a first approximation, our framework is able to execute DEVSJAVA models, but the reader will see that the web services have been developed using the adapter pattern, so the framework is extensible to other simulation platforms.

6.2.1 WWW and Distributed Simulation

Web-based simulation requires the convergence of simulation methodology and WWW technology (mainly web service technology). The fundamental concept of web services is to integrate software application as services. Web services allow the applications to communicate with other applications using open standards. We are offering DEVS-based simulators as a web service, and they must have these standard technologies: communication protocol (Simple Object Access Protocol, SOAP), service description (Web Service Description Language, WSDL), and service discovery (Universal Description Discovery and Integration, UDDI).

Figure 6.9 shows the framework of the proposed distributed simulation using SOA. The complete setup requires more than one server that is capable of running DEVS Simulation Service. The capability to run the simulation service is provided by the server side design of DEVS Simulation protocol supported by the latest DEVSJAVA Version 3.1.

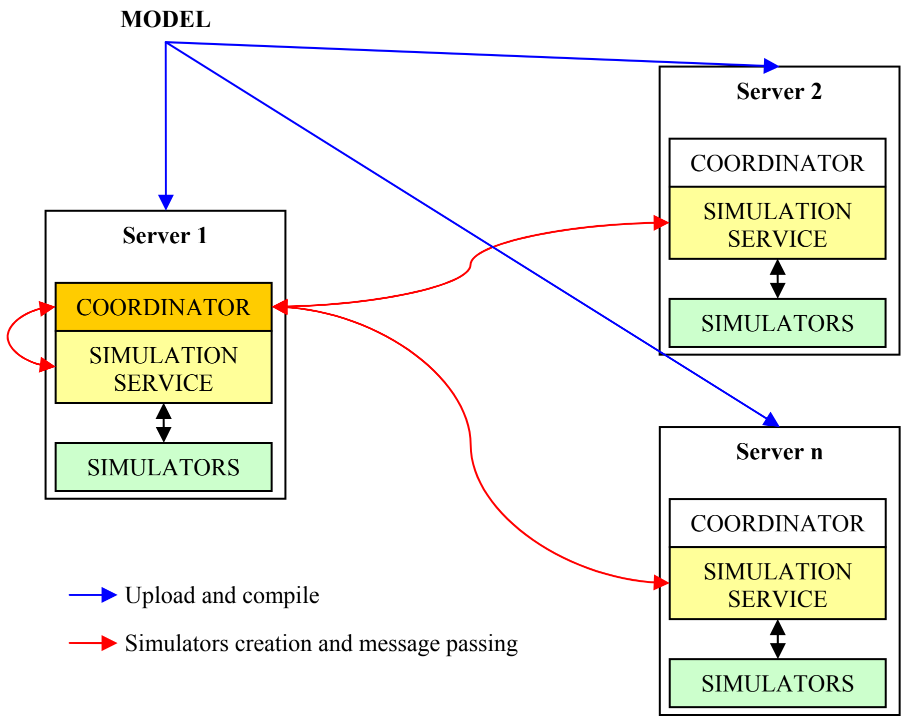


Figure 6.9: DEVS/SOA distributed architecture.

The Simulation Service framework is two layered framework. The top-layer is the user coordination layer that oversees the lower layer. The lower layer is the true simulation service layer that executes the DEVS simulation protocol as a Service. The lower layer is transparent to the modeler and only the top-level is provided to the user. The top-level has three main services:

- Upload DEVS model service.

- Compile DEVS model service.
- Simulate DEVS model service.

The top-level Service layer is presented in the WSDL below:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://devsml"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://devsml" xmlns:intf="http://devsml"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.3
Built on Oct 05, 2005 (05:23:37 EDT)-->
 <wsdl:types>
  <schema elementFormDefault="qualified"
targetNamespace="http://devsml"
xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="upload">
    <complexType>
     <sequence>
      <element name="packageName" type="xsd:string"/>
      <element name="arrayOfFileContents" type="xsd:base64Binary"/>
      <element name="arrayOfFileNames" type="xsd:base64Binary"/>
      <element maxOccurs="unbounded" name="ips" type="xsd:string"/>
      <element name="start" type="xsd:int"/>
     </sequence>
    </complexType>
   </element>
   <element name="uploadResponse">
    <complexType>
     <sequence>
      <element name="uploadReturn" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="simulate">
    <complexType>
     <sequence>
      <element name="clientId" type="xsd:string"/>
      <element name="mainClass" type="xsd:string"/>
      <element maxOccurs="unbounded" name="ips" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="simulateResponse">
    <complexType>
     <sequence>
      <element name="simulateReturn" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="compile">
    <complexType>
     <sequence>
      <element name="packageName" type="xsd:string"/>
      <element name="arrayOfFileNames" type="xsd:base64Binary"/>
      <element maxOccurs="unbounded" name="ips" type="xsd:string"/>
      <element name="start" type="xsd:int"/>
     </sequence>
    </complexType>
   </element>
   <element name="compileResponse">
    <complexType>
     <sequence>
      <element name="compileReturn" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
  </schema>
 </wsdl:types>
   <wsdl:message name="simulateResponse">
      <wsdl:part element="impl:simulateResponse" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="uploadRequest">
      <wsdl:part element="impl:upload" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="compileRequest">
      <wsdl:part element="impl:compile" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="uploadResponse">
      <wsdl:part element="impl:uploadResponse" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="simulateRequest">
      <wsdl:part element="impl:simulate" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="compileResponse">
      <wsdl:part element="impl:compileResponse" name="parameters"/>
   </wsdl:message>
   <wsdl:portType name="MainServices">
      <wsdl:operation name="upload">
         <wsdl:input message="impl:uploadRequest"
name="uploadRequest"/>
        <wsdl:output message="impl:uploadResponse"
name="uploadResponse"/>
      </wsdl:operation>
      <wsdl:operation name="simulate">
         <wsdl:input message="impl:simulateRequest"
name="simulateRequest"/>
         <wsdl:output message="impl:simulateResponse"
name="simulateResponse"/>
      </wsdl:operation>
      <wsdl:operation name="compile">
```

```
         <wsdl:input message="impl:compileRequest"
name="compileRequest"/>
         <wsdl:output message="impl:compileResponse"
name="compileResponse"/>
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="MainServicesSoapBinding"
type="impl:MainServices">
      <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="upload">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="uploadRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="uploadResponse">
           <wsdlsoap:body use="literal"/>
        </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="simulate">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="simulateRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="simulateResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="compile">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="compileRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="compileResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="MainServicesService">
      <wsdl:port binding="impl:MainServicesSoapBinding"
name="MainServices">
         <wsdlsoap:address
location="http://localhost:8080/DevsMLSimpleServer/services/MainService
s"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

The client is provided a list of servers hosting DEVS Service. He selects some servers to distribute the simulation of his model. Then, the model is uploaded and compiled in all the servers. The main server selected creates a coordinator that creates simulators in the server where the coordinator resides and/or over the other servers selected.

6.2.2 Abstraction of a Coupled model as an Atomic model with DEVS State Machine

One of the significant development steps we undertook in this effort is the masking of coupled model as an atomic model. What this implies is that we have an abstraction mechanism by which a coupled model can be treated as a black box and can be executed like an atomic model. In other words, a coupled model now has a state machine similar to that of any atomic model. In contrast to the DEVS hierarchical modeling, where a coupled model is merely a container and has corresponding coupled-simulators (Figure 6.10), now it is considered an atomic model with lowest level atomic simulator (Figure 6.11). This has been accomplished by implementing an adapter as shown in Figure 6.11 below. The adapter *Digraph2Atomic* takes each coupled component of the model and uses it as an atomic model. We will discuss this point later.

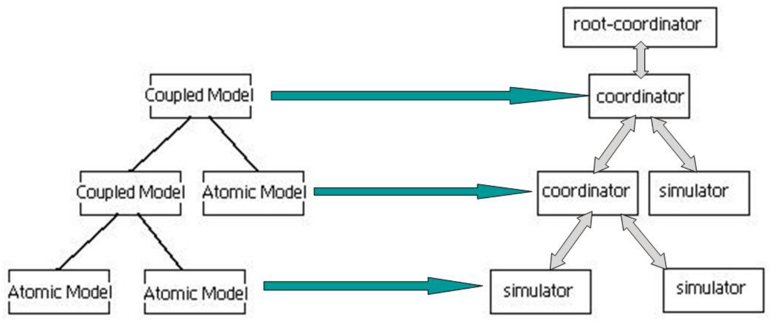


Figure 6.10: Hierarchical simulator assignment for a hierarchical model

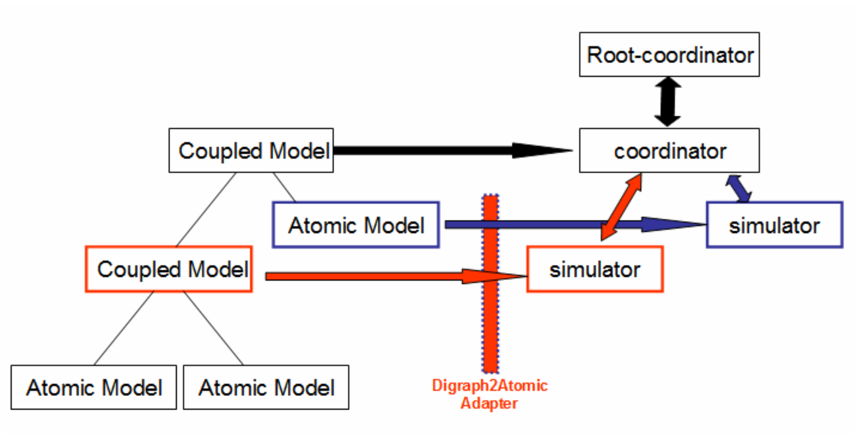


Figure 6.11: Hierarchical simulator assignment with Digraph2Atomic adapter

The implementation of the adapter is shown below:

```
public class Digraph2Atomic extends atomic {
      private CoordinatorInterface coord;

      public Digraph2Atomic(digraph model) {
            super(model.getName());

            couprel couplings = model.getCouprel();
            Iterator itr = couplings.iterator();
            while(itr.hasNext()) {
                  Pair relation = (Pair)itr.next();
                  Pair from = (Pair)relation.getKey();
                  String fromComponentName = (String)from.getKey();
                  String fromPortName = (String)from.getValue();
                  Pair to = (Pair)relation.getValue();
                  String toComponentName = (String)to.getKey();
                  String toPortName = (String)to.getValue();
                  if(fromComponentName.equals(model.getName()))
```

```
                        this.addInport(fromPortName);
                  else if(toComponentName.equals(model.getName()))
                        this.addOutport(toPortName);
            }
            coord = new coordinator(model);
      }

      public void initialize() {
            coord.initialize();
      }
      public void deltext(double e, message x) {
            coord.simInject(e, x);
      }
      public void deltint() {
            coord.wrapDeltfunc(coord.getTN());
      }
      public message out() {
            coord.computeInputOutput(coord.getTN());
            return (message)coord.getOutput();

      public double ta() {
            return coord.getTN() - coord.getTL();
      }

}
```

The number of simulators created depends on the number of components of the model at the top-level and the number of servers selected by the user. If the model contains 10 top-level components (including the contained digraphs) and the user select 5 servers, then 2 simulators are created in each server. After the whole simulation process, each simulation service sends a report back to the user containing information related to IP addresses and simulator assignment.

6.2.3 Message Serialization

The issue of message passing and models upload is done through serialization and SOA technologies. Figure 6.12 illustrates the message serialization process. When a component makes an external transition or executes the output function, the message received or emitted is serialized and then sent to the coordinator through the simulation service. The coordinator stores the location of each simulation service, so he is able to request all the messages after each iteration.

All the communication between the coordinator and simulation services is done through SOA protocol. The serialization is done through Java serialization utilities. Currently, a real time version is under development. In this version each simulator knows each simulation service at its end (from coupling information). So the communication can be solved by passing messages from simulation services to simulation services directly, without using the coordinator.

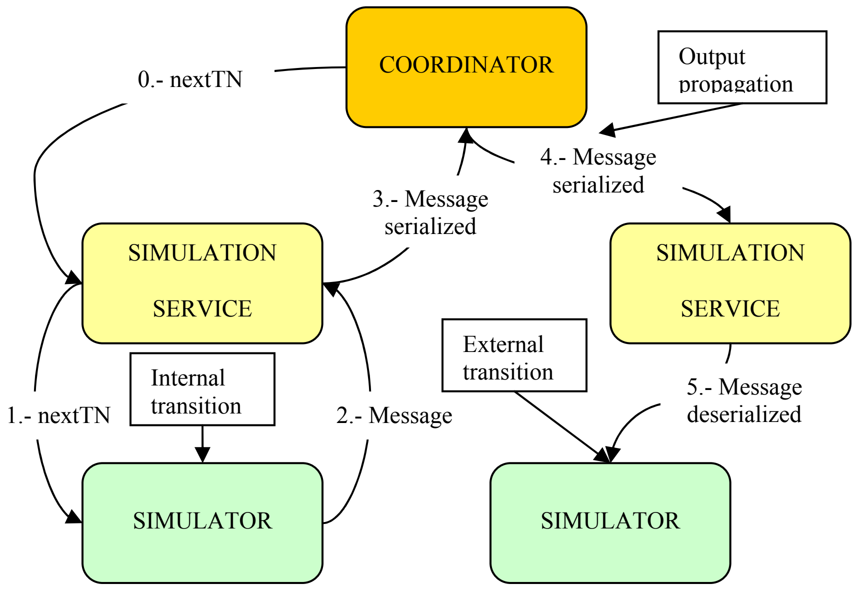


Figure 6.12: Communication among services

Summarizing from a user's perspective, the simulation process is done through three steps:

1. Write a DEVS model (currently DEVSJAVA is only supported).
2. Have a list of DEVS servers (through UDDI, for example). Since we are testing the application, these services have not been published using UDDI by now. Select N number of servers from the list available.
3. Run the simulation (upload, compile and simulate) and wait for the results.

Figure 6.13 shows these steps in graphical format.

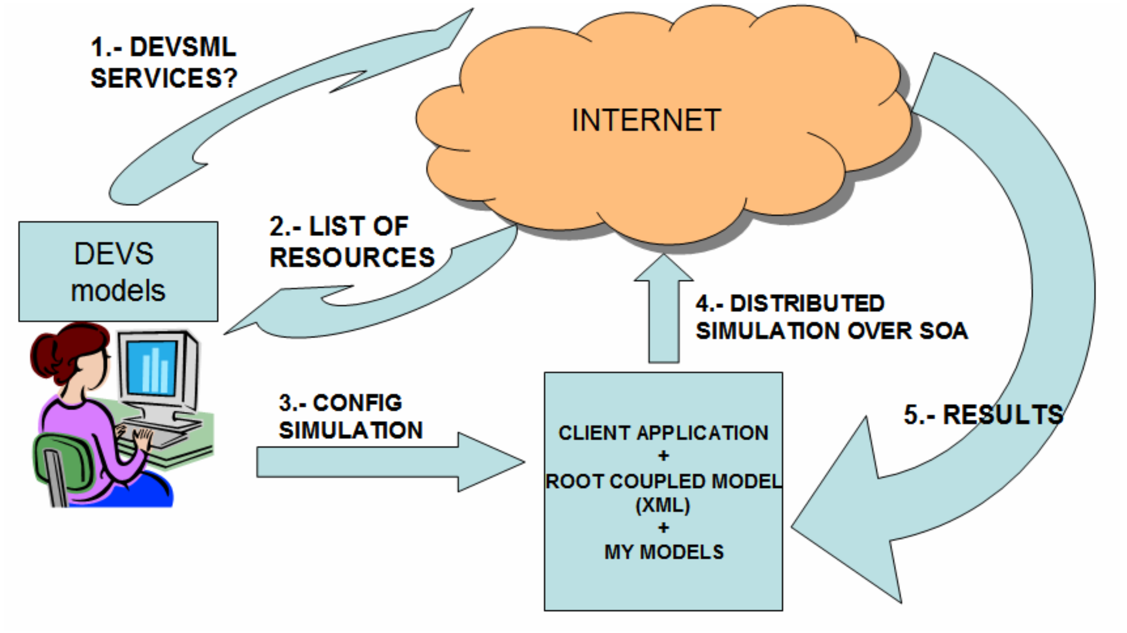

1.- DEVSML SERVICES?
INTERNET
DEVS models
2.- LIST OF RESOURCES
4.- DISTRIBUTED SIMULATION OVER SOA
3.- CONFIG SIMULATION
CLIENT APPLICATION + ROOT COUPLED MODEL (XML) + MY MODELS
5.- RESULTS

Figure 6.13: Execution of DEVS SOA-Based M&S

6.2.4 Details about the server architecture

The global design of the whole architecture at server's end is as follows, as shown in Figure 6.14.

api + devsml + adapter + modeling: This constitutes the DEVS modeling library. Once a DEVS model is received by the servers, the DEVS model is rebuilt using an adapter pattern. By now, only DEVSJAVA models are allowed. But, since this framework follows an adapter pattern, other Java-based models will be allowed in future. The **api** package contains only the interfaces. The **devsml** package contains Entity class. This is the starting class for all the modeling classes and allows serialization and deserialization. The **adapter** package contains the *Digraph2Atomic* class in Figure 6.15. This class is used to transform coupled components to atomic components. Thanks to this adapter we only have to design simulator services, so coordinator services are not necessary. The **modeling** package contains Atomic and Message classes shown in Figure 6.16. Atomic and Message follow an adapter pattern. Atomic encapsulates a DEVS atomic model and Message encapsulates a DEVS message.

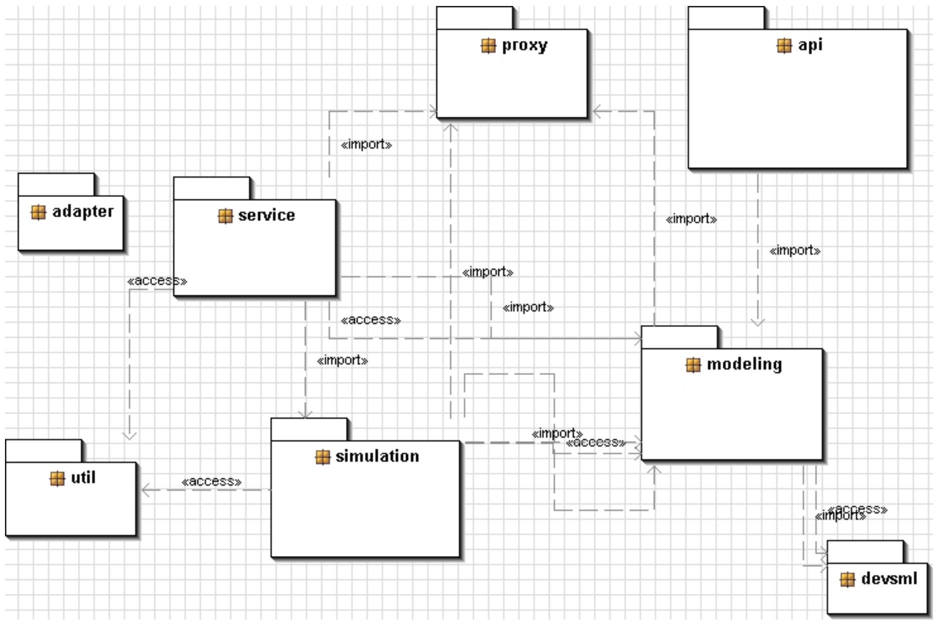


Figure 6.14: Server’s package structure for DEVS SOA

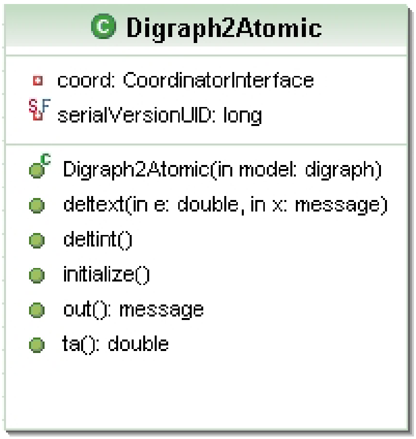

Digraph2Atomic
coord: CoordinatorInterface
serialVersionUID: long
Digraph2Atomic(in model: digraph)
deltext(in e: double, in x: message)
deltint()
initialize()
out(): message
ta(): double

Figure 6.15: Adapter package containing Digraph to Atomic adapters

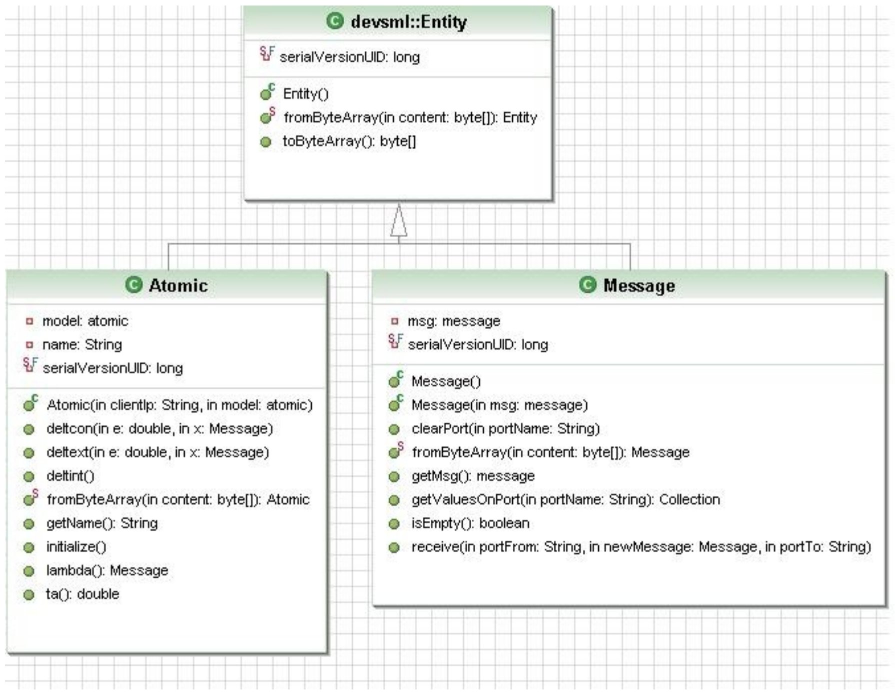

devsml::Entity
serialVersionUID: long
Entity()
fromByteArray(in content: byte[]): Entity
toByteArray(): byte[]
Atomic
model: atomic
name: String
serialVersionUID: long
Atomic(in clientIp: String, in model: atomic)
deltcon(in e: double, in x: Message)
deltext(in e: double, in x: Message)
deltint()
fromByteArray(in content: byte[]): Atomic
getName(): String
initialize()
lambda(): Message
ta(): double
Message
msg: message
serialVersionUID: long
Message()
Message(in msg: message)
clearPort(in portName: String)
fromByteArray(in content: byte[]): Message
getMsg(): message
getValuesOnPort(in portName: String): Collection
isEmpty(): boolean
receive(in portFrom: String, in newMessage: Message, in portTo: String)

Figure 6.16: devsml Modeling package for DEVS SOA

The **simulation** package contains simulators and coordinators, that is, Simulator and Coordinator classes as shown in Figure 6.17. The main difference with other simulators platforms starts here. The Coordinator is executed at the first server selected by the user. This coordinator is called through a *MainService* class publish as a Web service. The

Coordinator receives the user IP, the name of the root coupled model, and a list of IPs. The list of IPs is used to invoke simulation services in other remote servers. In this way, the components of the model are shared among N servers, where N is the length of that list. Also the Coordinator stores the user IP, the DEVSJAVA model, the last event time, the next event time, and a map of simulation services in use. This list is used to propagate and receive messages through the coupling protocol stored in the root coupled model.

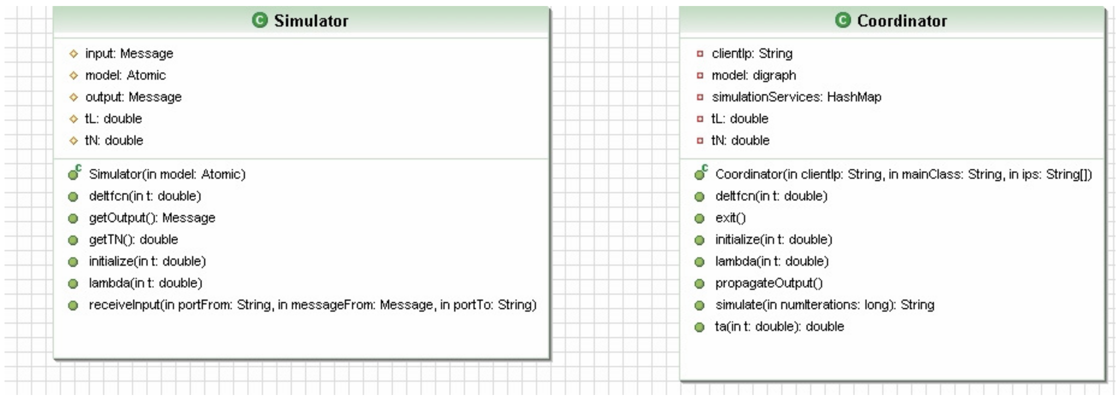


Figure 6.17: simulation package in DEVS SOA

The **service** package contains the services offered. It contains *MainService*, and *Simulation* classes as shown in Figure 6.18. *MainService* is designed to allow upload, compile and start the simulation process creating the coordinator. Simulation services are used to store the simulators used and to establish a communication between the DEVS simulators stored at this server and the coordinator housed in (maybe) other server. One server could be executing more than one simulator. It depends on the number of components that the root coupled model contains and the number of server selected by the user. This is the reason because there is not a unique relation between simulation service and simulator. The assignment of simulators corresponding to the models at the top-level is done through round-robin mechanism that takes care of model-simulator number mismatch. Functionality can be provided through which the user can direct any specific model to any particular IP server.

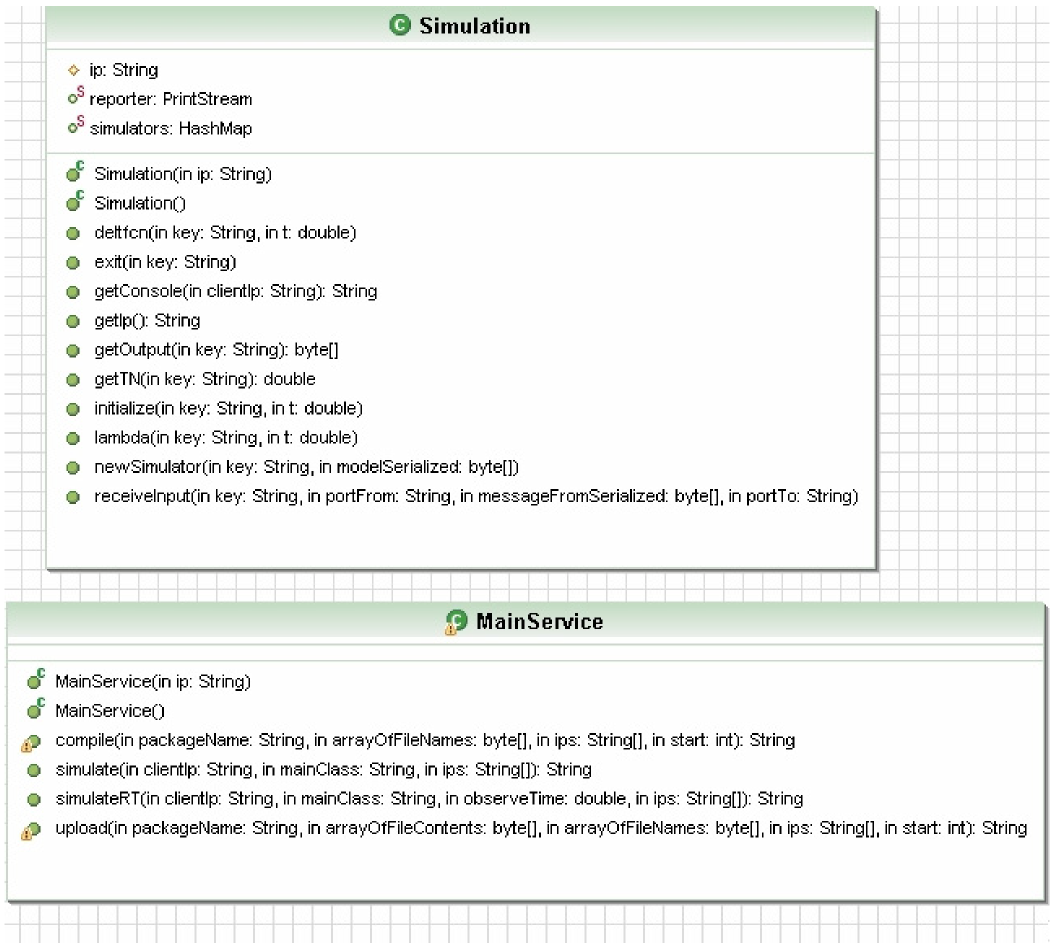

Simulation
ip: String
reporter: PrintStream
simulators: HashMap
Simulation(in ip: String)
Simulation()
deltfcn(in key: String, in t: double)
exit(in key: String)
getConsole(in clientIp: String): String
getIp(): String
getOutput(in key: String): byte[]
getTN(in key: String): double
initialize(in key: String, in t: double)
lambda(in key: String, in t: double)
newSimulator(in key: String, in modelSerialized: byte[])
receiveInput(in key: String, in portFrom: String, in messageFromSerialized: byte[], in portTo: String)
MainService
MainService(in ip: String)
MainService()
compile(in packageName: String, in arrayOfFileNames: byte[], in ips: String[], in start: int): String
simulate(in clientIp: String, in mainClass: String, in ips: String[]): String
simulateRT(in clientIp: String, in mainClass: String, in observeTime: double, in ips: String[]): String
upload(in packageName: String, in arrayOfFileContents: byte[], in arrayOfFileNames: byte[], in ips: String[], in start: int): String

Figure 6.18: Service package in DEVS SOA

The **proxy** package contains the proxies of the services as in Figure 6.19. All these classes are automatically generated from the WSDL files. The user only needs the *MainService* proxy. The server needs this service and other *Simulation* services.

MainService adds like a coordinator for all the lower-level services through interfaces. It assigns and initializes the coordinator which starts other simulators, after distributing the simulators at respective IPs. Once the simulators are active, the *MainService* waits for them to complete the execution to receive the logs and simulation outputs. This is because the main server needs simulation services from other servers to disperse the root coupled model through its components.

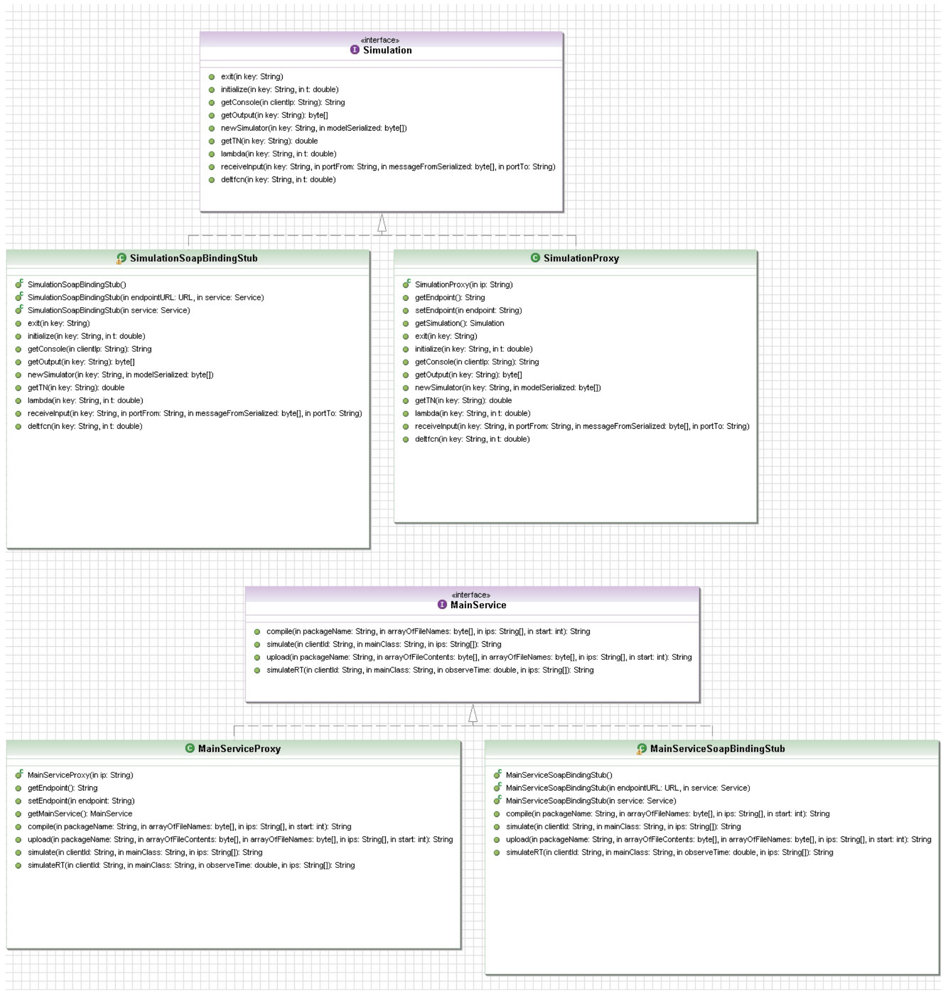
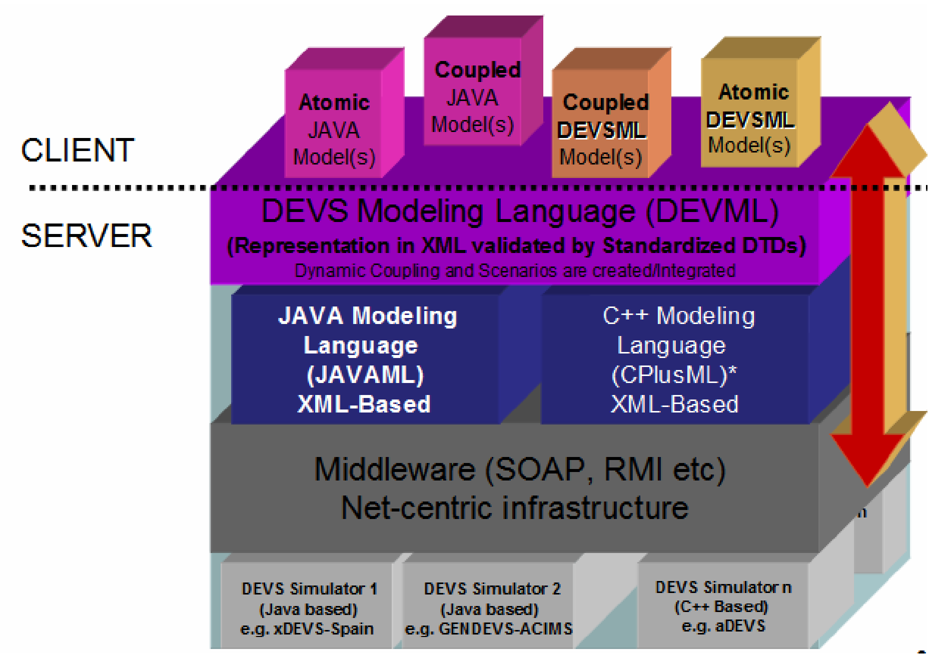

«interface»
Simulation
exit(in key: String)
initialize(in key: String, in t: double)
getConsole(in clientIp: String): String
getOutput(in key: String): byte[]
newSimulator(in key: String, in modelSerialized: byte[])
getTN(in key: String): double
lambda(in key: String, in t: double)
receiveInput(in key: String, in portFrom: String, in messageFromSerialized: byte[], in portTo: String)
deltfcn(in key: String, in t: double)
SimulationSoapBindingStub
SimulationSoapBindingStub()
SimulationSoapBindingStub(in endpointURL: URL, in service: Service)
SimulationSoapBindingStub(in service: Service)
exit(in key: String)
initialize(in key: String, in t: double)
getConsole(in clientIp: String): String
getOutput(in key: String): byte[]
newSimulator(in key: String, in modelSerialized: byte[])
getTN(in key: String): double
lambda(in key: String, in t: double)
receiveInput(in key: String, in portFrom: String, in messageFromSerialized: byte[], in portTo: String)
deltfcn(in key: String, in t: double)
SimulationProxy
SimulationProxy(in ip: String)
getEndpoint(): String
setEndpoint(in endpoint: String)
getSimulation(): Simulation
exit(in key: String)
initialize(in key: String, in t: double)
getConsole(in clientIp: String): String
getOutput(in key: String): byte[]
newSimulator(in key: String, in modelSerialized: byte[])
getTN(in key: String): double
lambda(in key: String, in t: double)
receiveInput(in key: String, in portFrom: String, in messageFromSerialized: byte[], in portTo: String)
deltfcn(in key: String, in t: double)
«interface»
MainService
compile(in packageName: String, in arrayOfFileNames: byte[], in ips: String[], in start: int): String
simulate(in clientId: String, in mainClass: String, in ips: String[]): String
upload(in packageName: String, in arrayOfFileContents: byte[], in arrayOfFileNames: byte[], in ips: String[], in start: int): String
simulateRT(in clientId: String, in mainClass: String, in observeTime: double, in ips: String[]): String
MainServiceProxy
MainServiceProxy(in ip: String)
getEndpoint(): String
setEndpoint(in endpoint: String)
getMainService(): MainService
compile(in packageName: String, in arrayOfFileNames: byte[], in ips: String[], in start: int): String
upload(in packageName: String, in arrayOfFileContents: byte[], in arrayOfFileNames: byte[], in ips: String[], in start: int): String
simulate(in clientId: String, in mainClass: String, in ips: String[]): String
simulateRT(in clientId: String, in mainClass: String, in observeTime: double, in ips: String[]): String
MainServiceSoapBindingStub
MainServiceSoapBindingStub()
MainServiceSoapBindingStub(in endpointURL: URL, in service: Service)
MainServiceSoapBindingStub(in service: Service)
compile(in packageName: String, in arrayOfFileNames: byte[], in ips: String[], in start: int): String
simulate(in clientId: String, in mainClass: String, in ips: String[]): String
upload(in packageName: String, in arrayOfFileContents: byte[], in arrayOfFileNames: byte[], in ips: String[], in start: int): String
simulateRT(in clientId: String, in mainClass: String, in observeTime: double, in ips: String[]): String

Figure 6.19: Proxy package in DEVS SOA

6.2.5 DEVSML and SOADEVS

In Section 6.1 we introduced DEVSML as a means to develop net-centric collaborative models resulting in a composite XML portable file that can be executed by the validated DEVS simulator. In this section we will illustrate how the DEVSML architecture aides the distributed execution over net-centric platform thereby offering simulator transparency using Simulation Services.

The DEVSML architecture is now divided in Client and Servers functionalities as shown below in Figure 6.20. The client provides model in DEVSJAVA or DEVSML, wherein they are transformable into each other and the Server end takes care of executing the simulation in a distributed manner using SOADEVS architecture.



CLIENT
SERVER
Atomic JAVA Model(s)
Coupled JAVA Model(s)
Coupled DEVSML Model(s)
Atomic DEVSML Model(s)
DEVS Modeling Language (DEVML)
(Representation in XML validated by Standardized DTDs)
Dynamic Coupling and Scenarios are created/Integrated
JAVA Modeling Language (JAVAML) XML-Based
C++ Modeling Language (CPlusML)* XML-Based
Middleware (SOAP, RMI etc)
Net-centric infrastructure
DEVS Simulator 1 (Java based) e.g. xDEVS-Spain
DEVS Simulator 2 (Java based) e.g. GENDEVS-ACIMS
DEVS Simulator n (C++ Based) e.g. aDEVS

Figure 6.20: DEVSML implementation over SOADEVS

Looking it in another perspective, the integration of DEVSML and SOADEVS is performed with the layout as shown below in Figure 6.21. The manner in which DEVSJAVA models could be attained or developed by client can be manifold. More details can be seen in Chapter 4. Once the client has DEVSJAVA models, DEVSML server can be used to integrate the client's model with model available at some other place on the web to get an enhanced integrated DEVSML file that can reproduce DEVSJAVA model in .java format. The SOADEVS enabled server can either take this integrated DEVSML file directly or can ask user to provide the top-level coupled manner, as described in earlier sections.

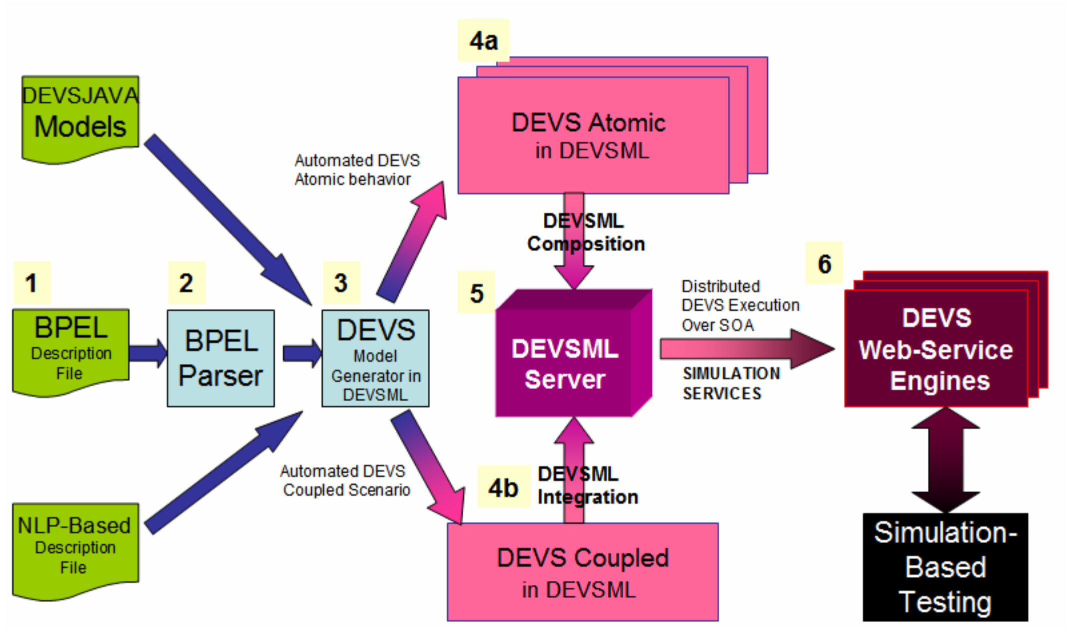


Figure 6.21: DEVSML and SOADEVS integrated

CHAPTER 7: DEVS UNIFIED PROCESS: PUTTING IT ALL TOGETHER

In an editorial [Car05], Carstairs asserts an acute need for a new testing paradigm that could provide answers to several challenges described in a three-tier structure. The lowest level, containing the individual systems or programs, does not present a problem. The second tier, consisting of systems of systems in which interoperability is critical, has not been addressed in a systematic manner. The third tier, the enterprise level, where joint and coalition operations are conducted, is even more problematic. Although current test and evaluation (T&E) systems are approaching adequacy for tier-two challenges, they are not sufficiently well integrated with defined architectures focusing on interoperability to meet those of tier three. To address mission thread testing at the second and third tiers, Carstairs advocates a collaborative distributed environment (CDE), which is a federation of new and existing facilities from commercial, military, and not-for-profit organizations. In such an environment, modeling and simulation (M&S) technologies can be exploited to support model-continuity [Hux04] and model-driven design (MDD) development [Weg02], making test and evaluation an integral part of the design and operations life-cycle.

The development of such a distributed testing environment would have to comply with recent Department of Defense (DoD) mandates requiring that the DoD Architectural Framework (DoDAF) be adopted to express high-level system and operational requirements and architectures [Dod03a, Dod03b, CJC04, CJC06]. Unfortunately, DoDAF and DoD net-centric [Atk04] mandates pose significant challenges to testing and evaluation since DoDAF specifications must be evaluated to see if they meet requirements and objectives, yet they are not expressed in a form that is amenable to such evaluation.

Combining the systems theory, M&S framework and model-continuity concepts leads naturally to a formulation of a Bifurcated Model-Continuity based Life-cycle process as illustrated in Figure 7.1 (reproduced again from Chapter 1). The process can be applied to development of systems using model-based design principles from scratch or as a process of reverse engineering in which requirements have already been developed in an informal manner. As we shall see ahead in next chapter, the said process is used in both manners. The depicted process is a universal process and is applicable in multiple domains. The objective of this research effort is to incorporate DEVS as the binding factor at all phases of this universal process.

This chapter describes the refined bifurcated Model-Continuity process and how various elements like automated DEVS model generation (Chapter 4), automated test-model generation (Chapter 5) and net-centric simulation over SOA (Chapter 6) are put together in the process, resulting in DEVS Unified Process.

Bifurcated Model-Continuity Based Life-Cycle Methodology

The process has the following characteristics:

- **Behavior Requirements at lower levels of System Specification**: The hierarchy of system specification as laid out in [Zeig] offers well-characterized levels at which requirements for system behavior can be stated. The process is essentially iterative and leads to increasingly rigorous formulation resulting from the formalization in subsequent phases.
- **Model Structures at higher levels of System Specification:** The formalized behavior requirements are then transformed to the chosen model implementations e.g. DEVS based transformation in C++, Java, C# and others.
- **Simulation Execution:** The model base which may be stored in Model Repository is fed to the simulation engine. It is important to state the fact that separating the Model from the underlying Simulator is necessary to allow independent development of each. Many legacy systems have both the Model and the Simulator tightly coupled to each other which restrict their evolution. DEVS categorically separates the Model from the Simulator for the same simple reason.
- **Real-time Execution:** The simulation can be made executable in real-time mode and in conjunction with Model-Continuity principles, the model itself becomes the deployed code
- **Test Models/Federations:** Branching in the lower-path of the Bifurcated process, the formalized models give way to test models which can be developed at the atomic level or at the coupled level where they become federations. It also leads

to the development of experiments and test cases required to test the system specifications. DEVS categorically aids the development of Experimental Frames at this step of development of test-suite.

- **Verification and Validation:** The simulation provides the basis for correct implementation of the system specifications over a wide range of execution platforms and the test suite provides basis for testing such implementations in a suitable test infrastructure. Both of these phases of systems engineering come together in the Verification and Validation (V&V) phase.

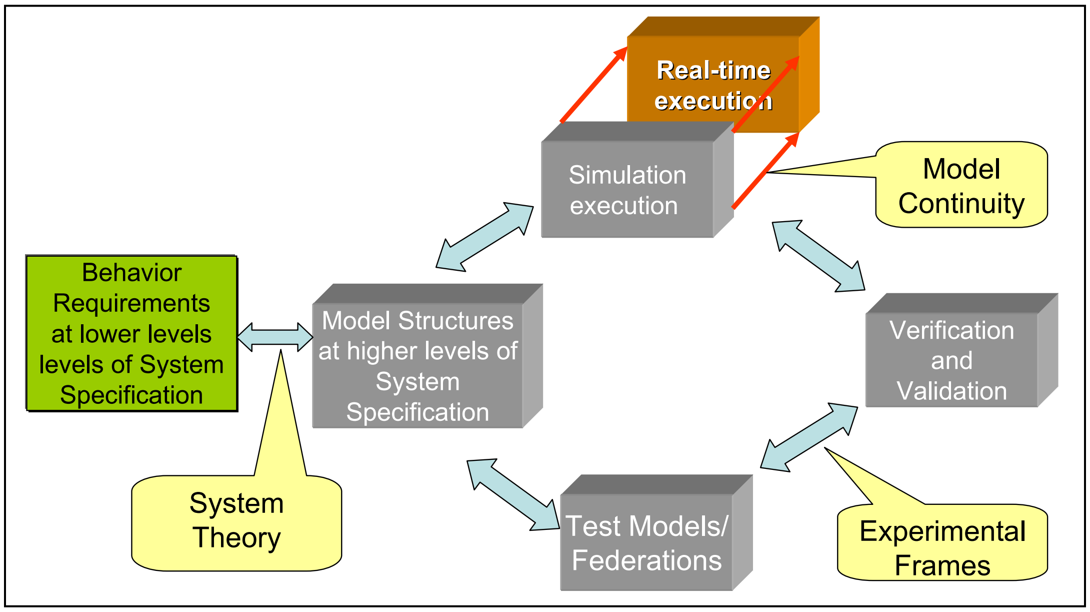

Real-time execution
Simulation execution
Model Continuity
Behavior Requirements at lower levels levels of System Specification
Model Structures at higher levels of System Specification
Verification and Validation
System Theory
Test Models/ Federations
Experimental Frames

Figure 7.1: Bifurcated Model-Continuity based System Life-cycle Process

7.1 Automated DEVS Model Generation and DEVSML

This section provides an overview on various technologies developed during this research effort in empowering DEVS to provide the complete solution for Bifurcated Life-cycle process. Considerable amount of effort has been spent in analyzing various forms of requirement specifications, viz, state-based, Natural Language based, Rule-based, BPMN/BPEL-based and DoDAF-based, and the automated processes which each one should employ to deliver DEVS hierarchical models and DEVS state machines. Chapter 4 provides an overview of these automated processes. Simulation execution today is more than just model execution on a single machine. With Grid applications and collaborative computing a norm in industry as well as in scientific community, this research effort also developed a net-centric platform using XML as middleware resulting in an infrastructure aiding distributed collaboration and model reuse. It led to the development of DEVS Modeling Language (DEVSML) and its net-centric execution using Service-Oriented Architecture called as SOADEVS. Both the DEVSML and SOADEVS provide novel approaches to integrate, collaborate and remotely execute models on SOA and are described in Chapter 6. The third area, in Chapter 5, which required development of automated procedures is the area of test-case generation leading to test-models. Using XML as the system specifications in rule-based format, a tool known as Automated Test Case Generator (ATC-Gen) was developed which facilitated the automated development of test models.

The integration of DEVSML and SOADEVS is performed with the layout as shown below in Figure 7.2. The manner in which DEVSJAVA models could be attained or developed by client can be manifold. As described in Chapter 4, it can come from state-based approach, BPEL-based or DoDAF-based or through NLP-based requirements. It is fed to the DEVSML client which coordinates with the DEVSML server farm. Once the client has DEVSJAVA models, DEVSML server can be used to integrate the client's model with model available at some other place on the web to get an enhanced integrated DEVSML file that can reproduce DEVSJAVA model in .java format. The SOADEVS enabled server can either take this integrated DEVSML file directly or can ask user to provide the top-level coupled manner, as described in earlier sections. Figure 7.2 provides an overview of the DEVSML input and its output to SOADEVS simulation framework with leads to the simulation-based systems testing.

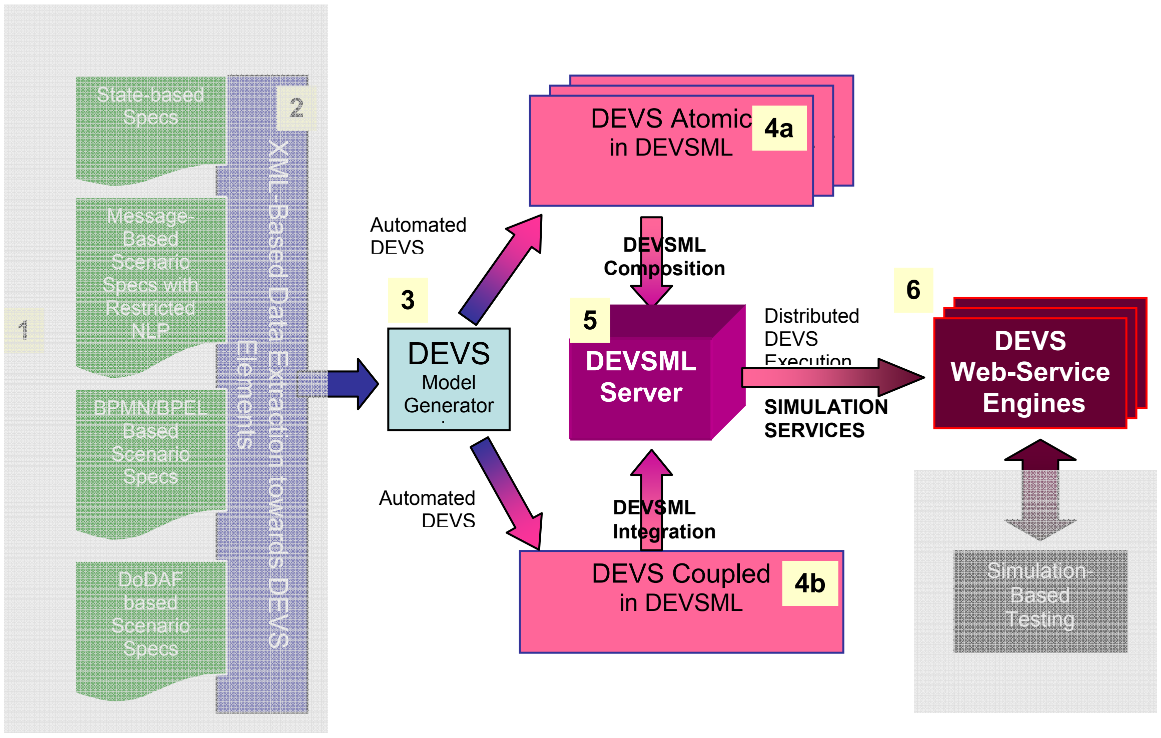


Figure 7.2: Netcentric collaboration and execution using DEVSML and SOADEVS

7.2 DEVSML Collaborative Development

This section provides information about the client application that communicates with the server resting at both ACIMS center and at Spain (redundancy purposes). The application is made available as an applet [DML] or as a .exe application that is capable of communicating to the server at client's end.

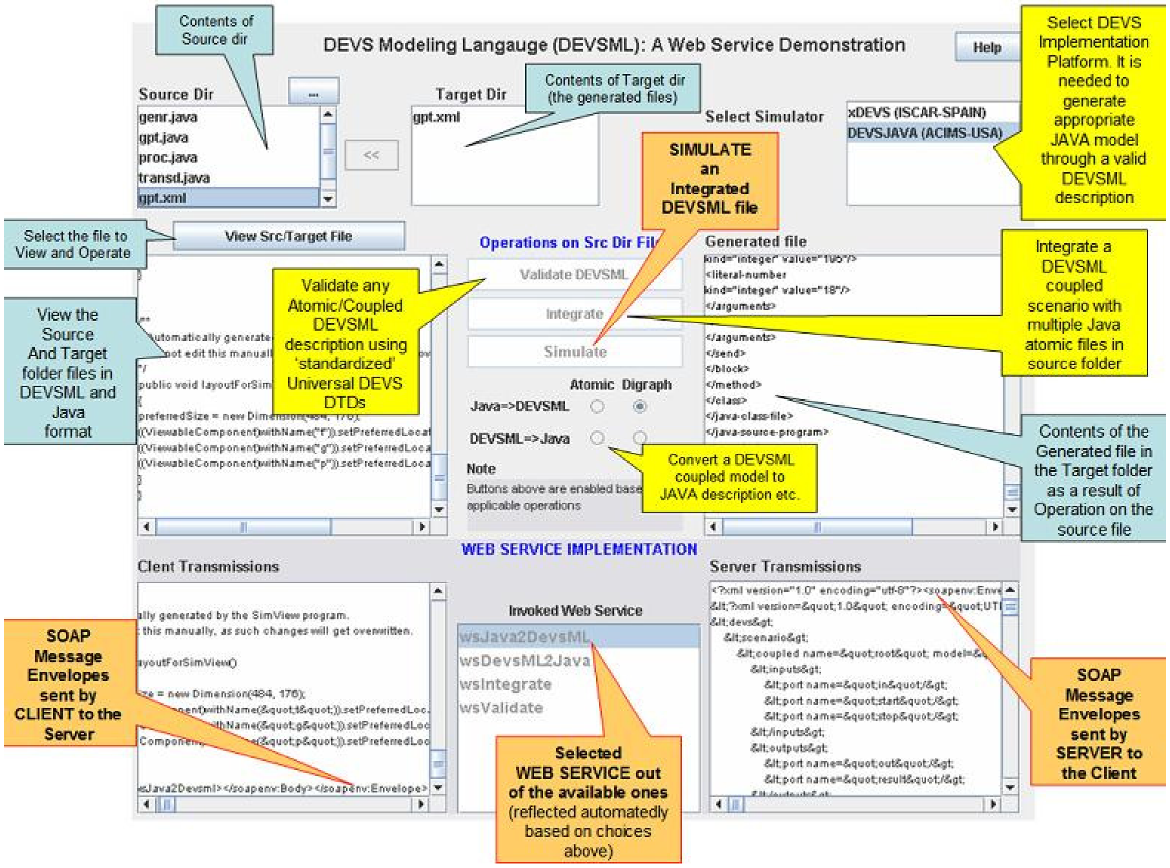
The following snapshot in Figure 7.3 shows the java application Ver. 2.0 that demonstrates the following:

1. Contains two simulator operability i.e xDEVS (Spain) [xDEVS] and GenDEVS (ACIMS-USA) [ACI06] demonstrating validation of DEVSML atomic and coupled models with same Atomic and Coupled DTD
2. Converts any atomic/coupled model from their JAVA implementation to DEVSML transformation and vice-versa
3. Validates any DEVSML model description
4. Integrates any coupled DEVSML description into a composite DEVSML coupled model ready to be simulated with corresponding simulator
5. Generation of JAVA code library from a composite DEVSML coupled model.
6. Out of ten web services in operation, five Web Services that are publicly offered are:
    a. Convert Java model to DEVSML
    b. Convert DEVSML to java code
    c. Validate the existing DEVSML model

d. Integrate coupled and atomic DEVSML models towards a portable 'Composite' Coupled DEVSML file that is Simulatable at any remote server

e. Simulates the Composite Coupled file and sends console messages at Server to Client window giving evidence of simulation running.

7. Server rests at ACIMS lab that provides these Services
8. User can select his own Source and Target directories
9. User can choose his chosen implementation i.e. java code and Simulator compatibility. The Server application checks for compatibility as well


Contents of Source dir
DEVS Modeling Langauge (DEVSML): A Web Service Demonstration
Help
Select DEVS Implementation Platform. It is needed to generate appropriate JAVA model through a valid DEVSML description
Source Dir
...
genr.java
gpt.java
proc.java
transd.java
gpt.xml
<<
Target Dir
gpt.xml
Contents of Target dir (the generated files)
Select Simulator
xDEVS (ISCAR-SPAIN)
DEVSJAVA (ACIMS-USA)
SIMULATE an Integrated DEVSML file
Select the file to View and Operate
View Src/Target File
Operations on Src Dir Fil
Generated file
View the Source And Target folder files in DEVSML and Java format
Validate any Atomic/Coupled DEVSML description using 'standardized' Universal DEVS DTDs
Validate DEVSML
Integrate
Simulate
Integrate a DEVSML coupled scenario with multiple Java atomic files in source folder
Atomic Digraph
Java=>DEVSML
DEVSML=>Java
Note
Buttons above are enabled base
applicable operations
Convert a DEVSML coupled model to JAVA description etc.
Contents of the Generated file in the Target folder as a result of Operation on the source file
WEB SERVICE IMPLEMENTATION
Clent Transmissions
Server Transmissions
Invoked Web Service
wsJava2DevsML
wsDevsML2Java
wsIntegrate
wsValidate
SOAP Message Envelopes sent by CLIENT to the Server
Selected WEB SERVICE out of the available ones (reflected automatically based on choices above)
SOAP Message Envelopes sent by SERVER to the Client

```
}

/**
 utomatically generate
 not edit this manuall
*/
public void layoutForSim
{
preferredSize = new Dimension(484, 176);
((ViewableComponent)withName("f")).setPreferredLocat
((ViewableComponent)withName("g")).setPreferredLoca
((ViewableComponent)withName("p")).setPreferredLoca
}
}
```

```
kind="integer" value="195"/>
<literal-number
kind="integer" value="18"/>
</arguments>

</arguments>
</send>
</block>
</method>
</class>
</java-class-file>
</java-source-program>
```

```
ally generated by the SimView program.
 this manually, as such changes will get overwritten.

ayoutForSimView()

ze = new Dimension(484, 176);
 nent)withName(&quot;t&quot;)).setPreferredLoc
 thName(&quot;g&quot;)).setPreferredLoc
Componen  (&quot;p&quot;)).setPreferredLoc

sJava2Devsml></soapenv:Body></soapenv:Envelope>
```

```
<?xml version="1.0" encoding="utf-8"?><soapenv:Enve
&lt;?xml version=&quot;1.0&quot; encoding=&quot;UT
&lt;devs&gt;
  &lt;scenario&gt;
    &lt;coupled name=&quot;root&quot; model=&
      &lt;inputs&gt;
        &lt;port name=&quot;in&quot;/&gt;
        &lt;port name=&quot;start&quot;/&gt;
        &lt;port name=&quot;stop&quot;/&gt;
      &lt;/inputs&gt;
      &lt;outputs&gt;
        &lt;port name=&quot;out&quot;/&gt;
        &lt;port name=&quot;result&quot;/&gt;
```

Figure 7.3: Client application snapshot implemented as an applet.

7.3 Automated Test-case Generation from DEVS models

Assuming that the DEVS model is easily specified using State-based approach as described in Section 4.1, the automated test-model generation is constructed at Level 1 of Input/Output behavior (see Table 3.2) taking DEVS component as a black-box. The test-model is called the Observer model and its state-machine is defined by the testee component's state-machine. The component model being tested is called Testee and the component model doing the testing is called Tester.

The prime objective of this Tester is to verify that the Input/Output pair co-exist according to the timeout specification as defined in Testee's configuration. The Tester's state machine then becomes a very simple state-machine with both the input message of Testee and output message of Testee as the external input messages for Tester. When the Testee receives the prescribed input message, it is also sent to the Tester's input port and starts a timer, the value of which is user-specified in the Tester's state machine specifications. When the Testee generates the corresponding output message, it is sent as an external input to the Tester, which if received before the timeout value, passivates the Tester. In case of Tester not receiving Testee's output message, it generates and ErrorReport message that is written in the logs as a failure of I/O transaction pair of the Testee model. The example can be seen in Chapter 8, Section 1. The other aspect of Tester-Testee configuration is their mutual coupling and the coupling of Tester with that of components coupled to the Testee's input port. The Tester is a mirror model of Testee with same inports and some additional inports that correspond to the outports of Testee. Hence, the only coupling between Testee and Tester is this additional coupling set. Rest all the couplings are same for Tester. Consequently, the Tester model coupling can be also generated automatedly. For a detailed example, refer Chapter 8, Section 1.

7.4 SOADEVS: Net-centric Execution using Simulation Service

This sub-section provides the client application to execute DEVS model over an SOA framework using Simulation as a Service as described in Chapter 6. From multifarious modes of DEVS model generation, the next step is the simulation of these models. The SOADEVS client takes the DEVS models package and through the dedicated servers hosting simulation services, it performs the following operations:

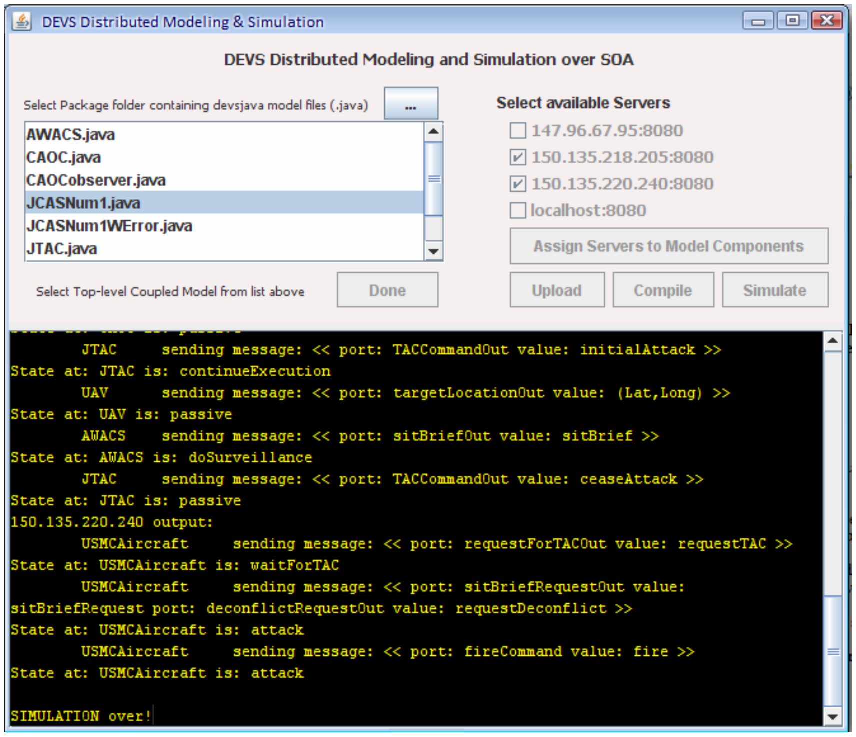
1. Upload the models to specific IP locations
2. Run-time compile at respective sites
3. Simulate the coupled-model
4. Receive the simulation output at client's end

The SOADEVS client as shown in Figure 7.5 below operates in the following sequential manner:

1. The user selects the DEVS package folder at his machine
2. The top-level coupled model is selected as shown in Figure 7.5
3. Various available servers are selected. Any number of available servers can be selected. Figure 7.6 shows how Servers are allocated on per-model basis. The user

can specifically assign specific IP to specific models at the top-level coupled domain. The localhost (Figure 7.5) is chosen using debugging sessions.

4. The user then uploads the model by clicking the Upload button. The models are partitioned in a round-robin mechanism and distributed among various chosen servers
5. The user then compiles the models by clicking the Compile button at server's end
6. Finally, Simulate button is pressed to execute the simulation using Simulation service hosted by these services.
7. Once the simulation is over, the console output window displays the aggregated simulation logs from various servers at the client's end.

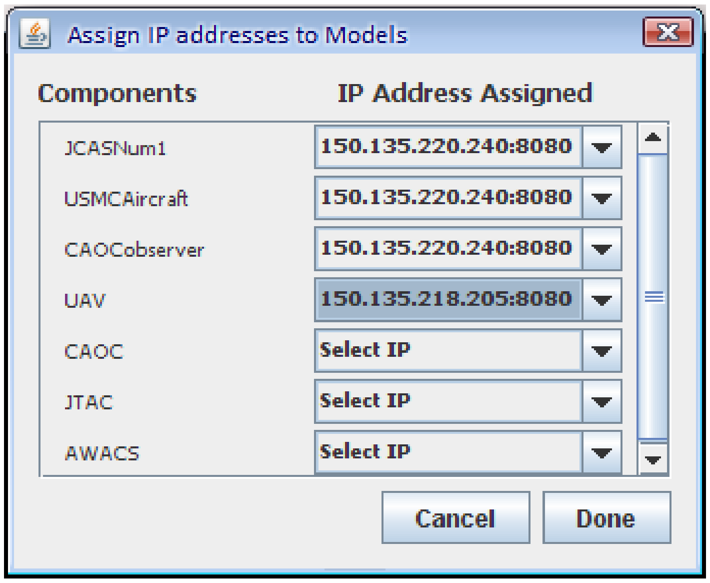

DEVS Distributed Modeling & Simulation
DEVS Distributed Modeling and Simulation over SOA
Select Package folder containing devsjava model files (.java)
...
Select available Servers
Assign Servers to Model Components
Select Top-level Coupled Model from list above
Done
Upload
Compile
Simulate

AWACS.java
CAOC.java
CAOCobserver.java
JCASNum1.java
JCASNum1WError.java
JTAC.java

[ ] 147.96.67.95:8080
[✓] 150.135.218.205:8080
[✓] 150.135.220.240:8080
[ ] localhost:8080

```
        JTAC     sending message: << port: TACCommandOut value: initialAttack >>
State at: JTAC is: continueExecution
        UAV      sending message: << port: targetLocationOut value: (Lat,Long) >>
State at: UAV is: passive
        AWACS    sending message: << port: sitBriefOut value: sitBrief >>
State at: AWACS is: doSurveillance
        JTAC     sending message: << port: TACCommandOut value: ceaseAttack >>
State at: JTAC is: passive
150.135.220.240 output:
        USMCAircraft     sending message: << port: requestForTACOut value: requestTAC >>
State at: USMCAircraft is: waitForTAC
        USMCAircraft     sending message: << port: sitBriefRequestOut value:
sitBriefRequest port: deconflictRequestOut value: requestDeconflict >>
State at: USMCAircraft is: attack
        USMCAircraft     sending message: << port: fireCommand value: fire >>
State at: USMCAircraft is: attack

SIMULATION over!
```

Figure 7.5: GUI snapshot of SOADEVS client hosting distributed simulation



Components	IP Address Assigned
JCASNum1	150.135.220.240:8080
USMCAircraft	150.135.220.240:8080
CAOCobserver	150.135.220.240:8080
UAV	150.135.218.205:8080
CAOC	Select IP
JTAC	Select IP
AWACS	Select IP

Figure 7.6: Server Assignment to Models

7.5 The Complete Process

This chapter has described various elements of DEVS Unified Process. Chapter 4 dealt with the automated generation of DEVS models from various modes of requirement specification. Chapter 5 dealt with Automated Test case generation directly from the requirements or from DEVS models. Chapter 6 dealt with the net-centric execution of DEVS models using DEVSML and SOADEVS clients. The basic Bifurcated Model Continuity-based Life-cycle process for systems engineering in Figure 7.1 in light of the developments in DEVS area is summarized in Figure 7.7 below. The grey boxes show the original process and the colored boxes show the extensions that were developed to make it a DEVS compliant process. A sample demo movie is available at [Dun07].

With the developed DEVS Unified Process we now have the capability to:

1. Transform various forms of requirement specifications to DEVS models in an automated manner.
2. Transform any DEVS model to a Platform Independent Model (PIM) using DEVSML for model and library reuse and sharing leading to collaborated development
3. Simulate any valid DEVSML using the SOADEVS architecture exploiting the transparent simulator paradigm for model interoperability execution (for models implemented in disparate languages e.g. Java and C++)
4. Transform any DEVSML model to a Service component in SOA

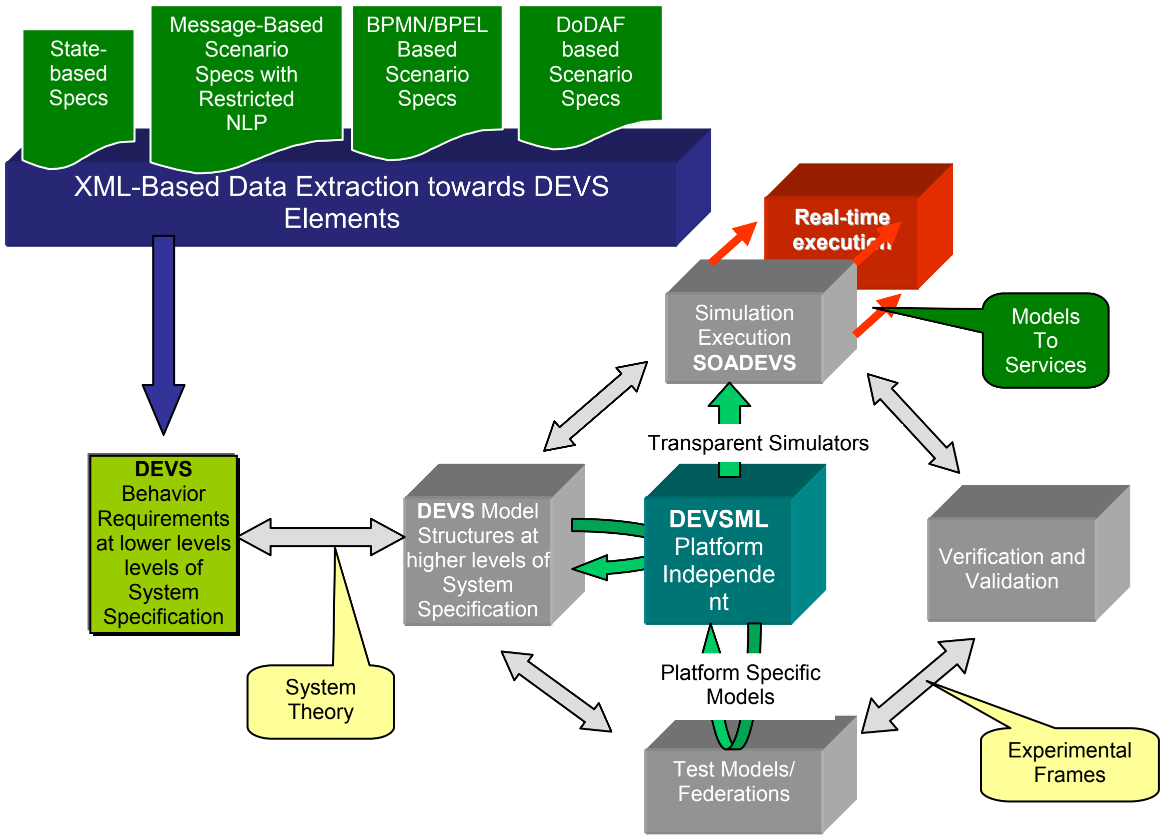

State-based Specs
Message-Based Scenario Specs with Restricted NLP
BPMN/BPEL Based Scenario Specs
DoDAF based Scenario Specs
XML-Based Data Extraction towards DEVS Elements
Real-time execution
Simulation Execution SOADEVS
Models To Services
Transparent Simulators
DEVS Behavior Requirements at lower levels levels of System Specification
DEVS Model Structures at higher levels of System Specification
DEVSML Platform Independent
Verification and Validation
Platform Specific Models
System Theory
Test Models/ Federations
Experimental Frames

Figure 7.7: The Complete DEVS Unified Process

CHAPTER 8: PROJECTS FROM WHICH DUNIP EVOLVED

This chapter contains many case studies that came about as DUNIP was defined and developed. Many of the projects are currently active at Joint Interoperability Test Command (JITC) and others are at concept validation stage towards a deliverable end. Each of the project either uses the complete DUNIP process or a subset of it. As we shall see on a case by case basis, DEVS emerge as a powerful M&S framework contributing to the roundtrip systems software engineering process. With the proposed DEVS Based Bifurcated Model-continuity Life-cycle process, systems theory with DEVS implementation finds its way to the next generation net-centric application development and testing.

This chapter describes the following case studies:

1. Joint Close Air Support (JCAS) model
2. DoDAF-based Activity scenario
3. Link-16 Automated Test Case Generator (ATC-Gen project at JITC)
4. Generic Network for Systems Capable of Planned Expansion (GENETSCOPE project at JITC)

Each of the projects has been developed independently and ATC-Gen and GENETSCOPE are team projects. All of the projects stand-alone and applies DUNIP

(Figure 7.7 in full or in-part. Table 8.1 below provides an overview of the DUNIP elements used in each of the projects. All of the DUNIP elements have been applied at least once in one of the projects. However, presently, there is not an available live case study that implements all the aspects of DUNIP elements as DUNIP was not defined prior to the design of these active projects

Project / DUNIP Elements		**JCAS model**	**DoDAF-based Activity Scenario**	**ATC-Gen Project**	**GENETSCOPE Project**
Requirement Specification Formats		X			X
	State-based Specs	X			
	Message-based Specs with restricted NLP	X			
	BPMN/BPEL based Specs	X			
	DoDAF-Based Scenario Specs		X		X
XML-based Data Extraction		X	X	X	
DEVS Model Structure at lower levels of Specification		X	X	X	
DEVS model structure at higher levels of System specification			X		X
DEVSML Platform Independent Models		X			
Test Model Development		X		X	
Verification and Validation using Experimental Frames			X	X	X
SOADEVS net-centric Simulation		X			

Table 8.1: Overview of DUNIP application in available case-studies

The JCAS system requirements come in many formats and it served as a base example to test many of the DUNIP earlier processes for requirements-to-DEVS transformation. It was specified using the state-based approach, BPEL-based approach and restricted natural language approach. This case study describes all three of the approaches leading to an executable DEVS model with identical simulation results. Finally, the executable model is executed over a net-centric platform using DEVSML and SOADEVS architecture.

DODAF-based Activity scenario is specified in UML based Activity specification and it illustrates the process carried to transform various DoDAF documents into DEVS requirement specifications. Population of the new proposed Operational View document OV-8 and OV-9 is described and how DEVS models could be generated from these two documents is illustrated. Complete example is presented in [Mit06a].

The ATC-Gen project at JITC is the project dealing with automated Link-16 testing environment and the design of ATC-Gen tool. A sample experiment is described and results are provided. Complete example is presented in [Mak06].

The GENETSCOPE project at JITC is another project funded by JITC that employs the complete DEVS software engineering process. A ten year old legacy model was taken and using automated XML data mining, the C language written process-model was transformed to object-oriented DEVS model with enhanced MVSC paradigm. Design elements of GENETSCOPE tool are discussed. Its relationship with the overarching DoDAF framework is also presented. Complete example is presented in [Mit06b].

8.1 Joint Close Air Support (JCAS) Model

The Joint Close Air Support Model is expressed in plain English as shown in Figure 8.1 below. It is a small example involving components exchanging messages towards a common objective. The requirements are then translated to various DEVS generating modes as described in Chapter 4. We shall see the execution of JCAS for each of the approaches.

8.1.1 State-based approach

The state transitions are provided using the tabular format as described in section 4.1.1. The components of JCAS model are:

1. JTAC
2. UAV
3. CAOC
4. USMC Aircraft
5. AWACS

The scenario is provided as follows:

JCAS JMT Operational Scenario #1

A. Special Operations Force (SOF) (AFSOC and NSW) JTAC working with Operational Detachment-Alpha (ODA) is tasked to request Immediate CAS on a stationary mechanized target in mountainous terrain. A Predator unmanned aerial vehicle (UAV) is on station for support.

B. SOF JTAC contacts AWACS with request. AWACS passes the request to Special Operations Liaison Element (SOLE) in the Combine Air Operations Center (CAOC).

C. Joint Special Operations Task Force (JSOFT) approves the request and CAOC assigns a section of USMC F/A-18Ds, F-15Es, and a single B-1B. Ordnance consists of 20mm, Joint Direct Attack Munitions (JDAMs), and Laser Guided Bombs

(LGBs).

D. Aircraft get situational brief from AWACS aircraft while in route, then switch to SOF JTAC for Terminal Attack Control and deconfliction from orbiting UAV. A 9-Line brief will be given to each section/single aircraft. JTAC will continue to execute CAS missions until all weapons are expended.

Figure 8.1: JCAS Operational Scenario

Translating the behavior to Tabular format for the entity JTAC, the state machines looks as shown in Table 8.2:

DEVS Internal State Machine (for default behavior)					
Behavior S.No.	**State (phase)**	**Next State (phase)**	**Timeout**	**Outgoing Message**	
1.	RequestImmediateCAS	WaitForAssignment	0	CASResourceSpec	
2.	WaitForAssignment	Passive	Infinity	-	
3.	ProvideTAC	ContinueExecution	1000	-	
4.	ContinueExecution	Passive	0	CeaseAttack	
5.	WaitForTACRequest	Passive	Infinity	-	

DEVS External State Machine responding to incoming messages					
Behavior S.No.	**Incoming Message name**	**State (phase)**	**Next State (phase)**	**Timeout**	**Outgoing Message**
1.	RequestTAC	WaitForTACRequest	ProvideTAC	10	InitialAttack
2.	YouCanUseUSMCAircraft	WaitForAssignment	WaitForTACRequest	0	-

Table 8.2: State machine for component JTAC

Similarly, all other components can be described. The coupled model created manually is shown in Figure 8.2 below:

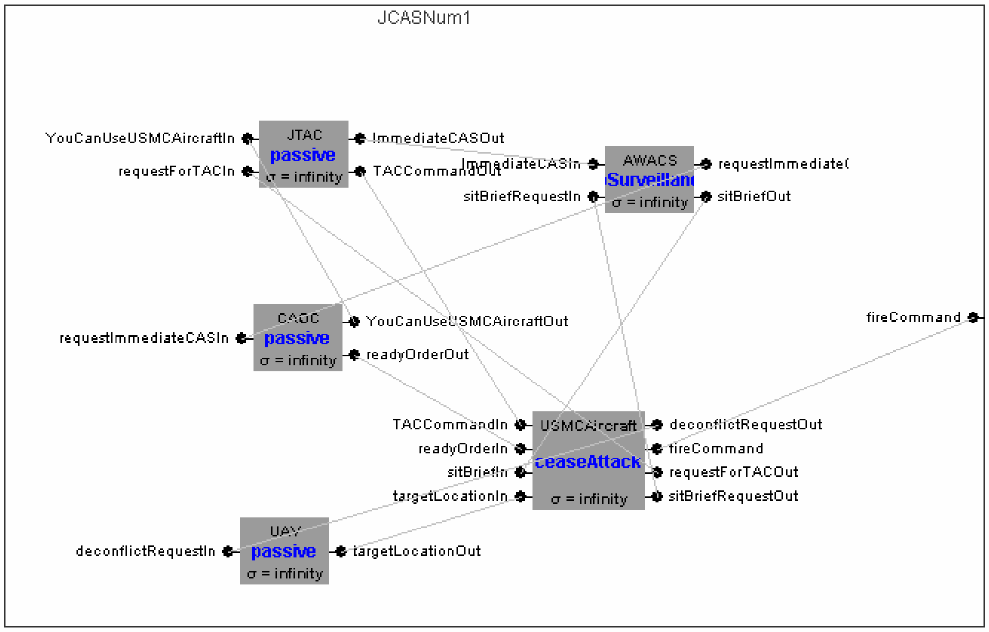


Figure 8.2: Coupled scenario for JCAS model

The execution of the coupled model resulted in the simulation output (on console) of the successful message passing and scenario execution as shown below in Figure 8.3.

```
State at: UAV is: passive       with tN: ?
State at: JTAC is: requestImmediateCAS  with tN: 0.000
State at: CAOC is: passive      with tN: ?
State at: USMCAircraft is: passive      with tN: ?
State at: AWACS is: passive     with tN: ?
        JTAC     sending message: << port: ImmediateCASOut value:
CASResourcesSpec >>
State at: JTAC is: waitForAssignment    with tN: 0.000
        AWACS   recvd message: <<  port: ImmediateCASIn value:
CASResourcesSpec>>
        AWACS    sending message: << port: requestImmediateCASOut value:
CASResourcesSpec >>
        CAOC    recvd message: <<  port: requestImmediateCASIn value:
CASResourcesSpec >>
State at: AWACS is: doSurveillance      with tN: 1.000
        CAOC     sending message: << port: YouCanUseUSMCAircraftOut value:
CASResources
                                 port: readyOrderOut value: getReady >>
        JTAC    recvd message: <<  port: YouCanUseUSMCAircraftIn value:
CASResources >>
State at: CAOC is: passive      with tN: 2.000
        USMCAircraft    recvd message: <<  port: readyOrderIn value: getReady
>>
        USMCAircraft     sending message: << port: requestForTACOut value:
requestTAC >>
        JTAC    recvd message: <<  port: requestForTACIn value: requestTAC >>
```

```
State at: USMCAircraft is: waitForTAC   with tN: 102.000
        JTAC     sending message: << port: TACCommandOut value: initialAttack
>>
State at: JTAC is: continueExecution    with tN: 112.000
        USMCAircraft    recvd message: <<  port: TACCommandIn value:
initialAttack >>
        USMCAircraft     sending message: << port: sitBriefRequestOut value:
sit
BriefRequest port: deconflictRequestOut value: requestDeconflict >>
        UAV     recvd message: <<  port: deconflictRequestIn value:
requestDeconflict >>
State at: USMCAircraft is: attack       with tN: 122.000
        AWACS   recvd message: <<  port: sitBriefRequestIn value:
sitBriefRequest >>
        UAV      sending message: << port: targetLocationOut value: (Lat,Long)
>>
State at: UAV is: passive       with tN: 123.000
        USMCAircraft    recvd message: <<  port: targetLocationIn value:
(Lat,Long) >>
        AWACS    sending message: << port: sitBriefOut value: sitBrief >>
        USMCAircraft    recvd message: <<  port: sitBriefIn value: sitBrief >>
State at: AWACS is: doSurveillance      with tN: 132.000
        USMCAircraft     sending message: << port: fireCommand value: fire >>
State at: USMCAircraft is: attack       with tN: 222.000
        JTAC     sending message: << port: TACCommandOut value: ceaseAttack >>
State at: JTAC is: passive      with tN: 1112.000
        USMCAircraft    recvd message: <<  port: TACCommandIn value:
ceaseAttack >>
Terminated Normally before ITERATION 11 ,time: 1112.0
```

Figure 8.3: DEVS Execution of JCAS model on console

8.1.2 BPMN/BPEL based approach

In this approach we approached the problem using a BPMN diagram. The scenario in Figure 8.1 is expressed as a BPMN diagram shown in Figure 8.4 below. The BPMN diagram was created manually using the tool Borland Eclipse Together 2006.

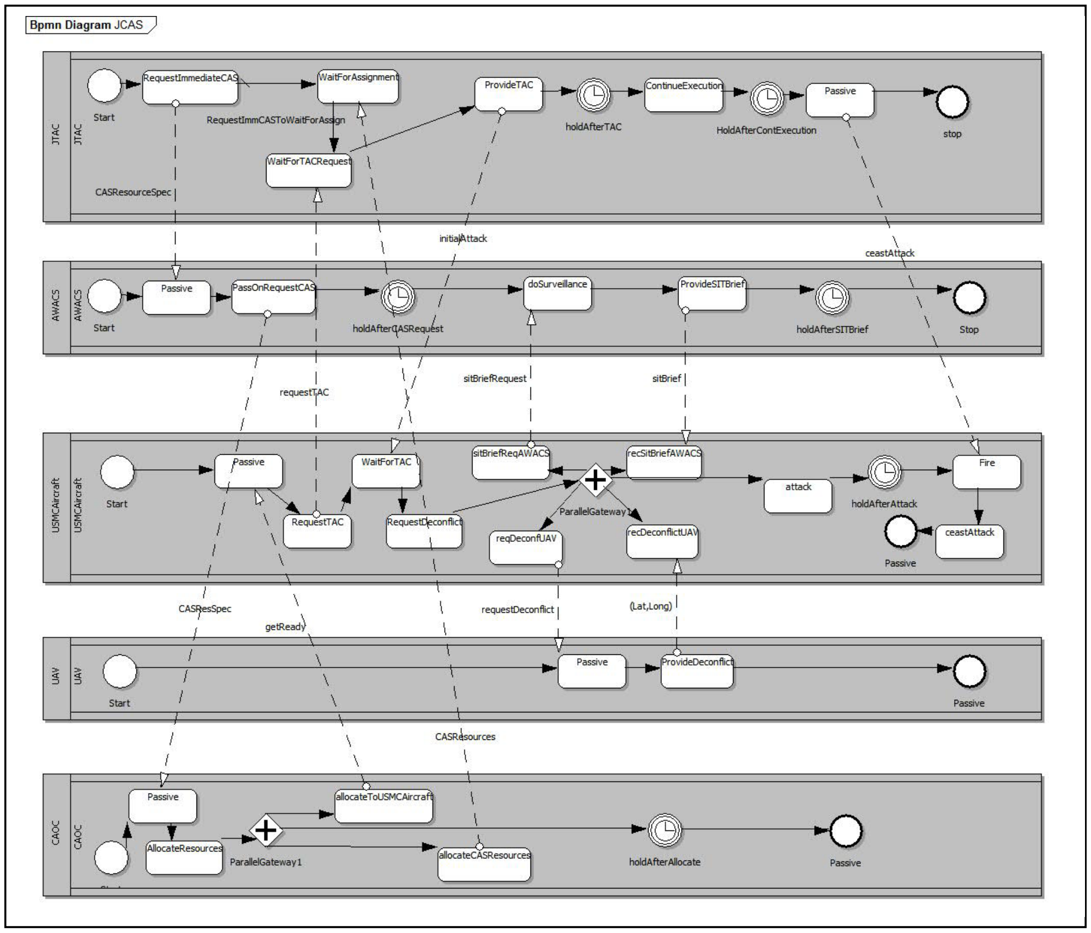
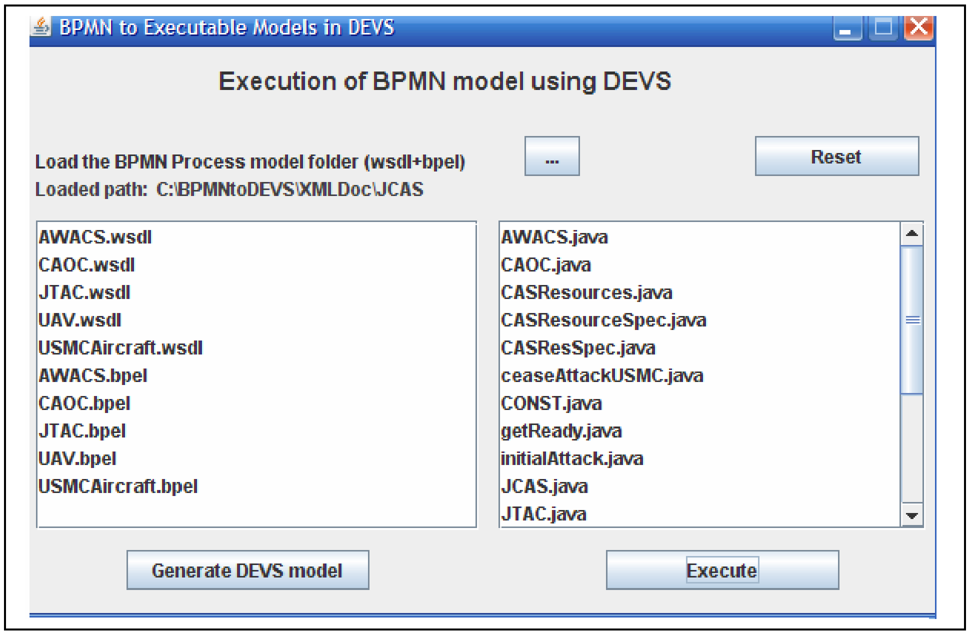


Figure 8.4: JCAS BPMN scenario description

The Eclipse Together tool generated the corresponding .bpel and .wsdl files for the JCAS scenario. In total 10 files were generated (5 .bpel and 5 .wsdl files). The generated files are shown in the left column of Figure 8.5.

We took these generated files to our BPEL-to-DEVS transformation tool as described in Chapter 4 and generated the DEVS model out of these files.


BPMN to Executable Models in DEVS

Execution of BPMN model using DEVS

Load the BPMN Process model folder (wsdl+bpel)
Loaded path: C:\BPMNtoDEVS\XMLDoc\JCAS

```
AWACS.wsdl            AWACS.java
CAOC.wsdl             CAOC.java
JTAC.wsdl             CASResources.java
UAV.wsdl              CASResourceSpec.java
USMCAircraft.wsdl     CASResSpec.java
AWACS.bpel            ceaseAttackUSMC.java
CAOC.bpel             CONST.java
JTAC.bpel             getReady.java
UAV.bpel              initialAttack.java
USMCAircraft.bpel     JCAS.java
                      JTAC.java
```

Figure 8.5: Snapshot of a BPMN-to-DEVS Transformation tool

The transformation process generated the following .java files (which include additional files as well) shown in the right column of Figure 8.5 above.

1. **JCAS.java**
2. **JTAC.java**
3. **AWACS.java**
4. **CAOC.java**
5. **UAV.java**
6. **USMCAircraft.java**
7. CASResources.java
8. CASResourceSpec.java
9. CASResSpec.java
10. ceaseAttackUSMC.java

11. CONST.java
12. getReady.java
13. initialAttack.java
14. latLong.java
15. requestDeconflict.java
16. requestTAC.java
17. sitBrief.java
18. sitBriefRequest.java
19. TimerMessage.java

The additional files correspond to various messages that were exchanged in the scenario. The files in the bold (above) are the main component files that contain the DEVS state machine.

Finally, using the BPMN-to-DEVS tool, the package was compiled run-time and simulation was executed yielding the same result as of Figure 8.3. The Execute button brings up the DEVSJAVA Simulation Viewer (Figure 8.5) which executes the simulation.

8.1.3 Message-Based Restricted NLP-based approach

In this approach the JCAS scenario in Figure 8.1 is expressed in message-based NLP format as described in Section 4.3. The resulting NLP specification is shown in Figure 8.6 below.

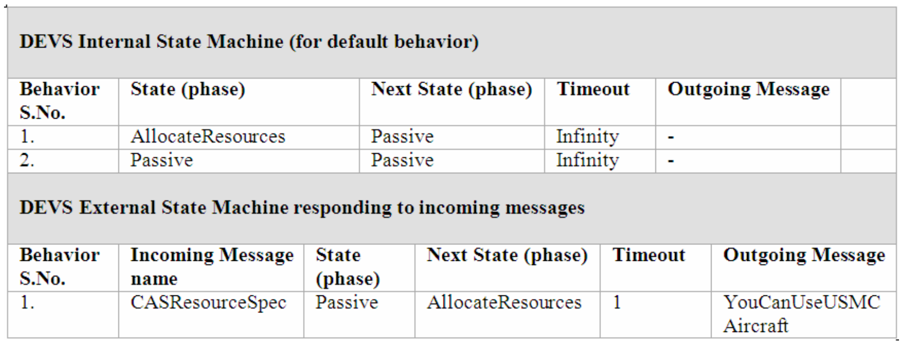
```
JTAC sends  CASResourceSpec to AWACS
Having received CASResourceSpec from JTAC,
    AWACS sends CASResourceSpec to CAOC within 1 minute
Having received CASResourceSpec from AWACS,
    CAOC sends CASResources to JTAC within 1 minute
             and sends getReady to USMCAircraft within 1 minute
Having received getReady from CAOC,
    USMCAircraft sends requestTAC to JTAC within 100 minutes
Having sent CASResourceSpec to AWACS and
     Having received requestTAC from USMCAircraft,
          JTAC sends initateAttack to USMCAircraft within 10 minutes
Having sent requestTAC to JTAC and
   Having received initiateAttack from JTAC
    USMCAircraft sends sitBriefRequest to AWACS within 10 minutes
            and sends requestDeconflict to UAV within 10 minutes
            and sends Fire to external within 100 minutes
            and expects (Lat,Long) from UAV within 100 minutes
            and expects sitBrief from AWACS within 100 minutes

Having received requestDeconflict from USMCAircraft
      UAV sends (Lat,Long) to USMCAircraft within 1 minute
Having sent CASResourceSpec to CAOC and
    Having received sitBriefRequest from USMCAircraft
      AWACS sends sitBrief to USMCAircraft within 10 minutes
Having received initiateAttack from JTAC and
   Having received (Lat,Long) from UAV and
        USMCAircraft sends Fire to external
    Having received sitBrief from AWACS
         USMCAircraft sends Fire to external
```

Figure 8.6: Message-based Restricted NLP description of JCAS scenario

The DEVS models are created based on the methodology described in Section 4.3 leading to the same simulation results as of Figure 8.3.

8.1.4 Automated test case generation for JCAS

As described in Section 7.3, observer models can be created for each of the entity models listed constituting the JCAS coupled model. Figure 8.7. For the JCAS model we created the observer for component CAOC. The CAOC model has the following state description as shown in Figure 8.8 below:



DEVS Internal State Machine (for default behavior)

Behavior S.No.	State (phase)	Next State (phase)	Timeout	Outgoing Message	
1.	AllocateResources	Passive	Infinity	-	
2.	Passive	Passive	Infinity	-	

DEVS External State Machine responding to incoming messages

Behavior S.No.	Incoming Message name	State (phase)	Next State (phase)	Timeout	Outgoing Message
1.	CASResourceSpec	Passive	AllocateResources	1	YouCanUseUSMC Aircraft

Figure 8.7: State-based specification of model CAOC

After generation of the DEVS model for CAOC from Figure 8.7 above, an observer model with the state-machine shown in Figure 8.8b can be automatedly generated from column 2 (incoming message) and column 6 (output message) pair. The basic operation of CAOC Observer is to verify that CAOC performs its I/O operations i.e. on receiving an input message it generates the prescribed output message. Both the input and output message for COAC becomes the external input message for the CAOC Observer component. In this case on receiving *CASResourceSpec* message it should produce *YouCanUseUSMCAircraft* message. Figure 8.8b below provides the DEVS state-machine for CAOC Observer.

DEVS Internal State Machine (for default behavior)

Behavior S.No.	State (phase)	Next State (phase)	Timeout	Outgoing Message	
1.	WatchForError	Passive	Infinity	-	
2.	Passive	Passive	Infinity	-	

DEVS External State Machine responding to incoming messages

Behavior S.No.	Incoming Message name	State (phase)	Next State (phase)	Timeout	Outgoing Message
1.	CASResourceSpec	Passive	WatchForError	100	ErrorReport
2	YouCanUseUSMC Aircraft	WatchFor Error	Passive	0	-

Figure 8.8: State-machine for CAOC Observer

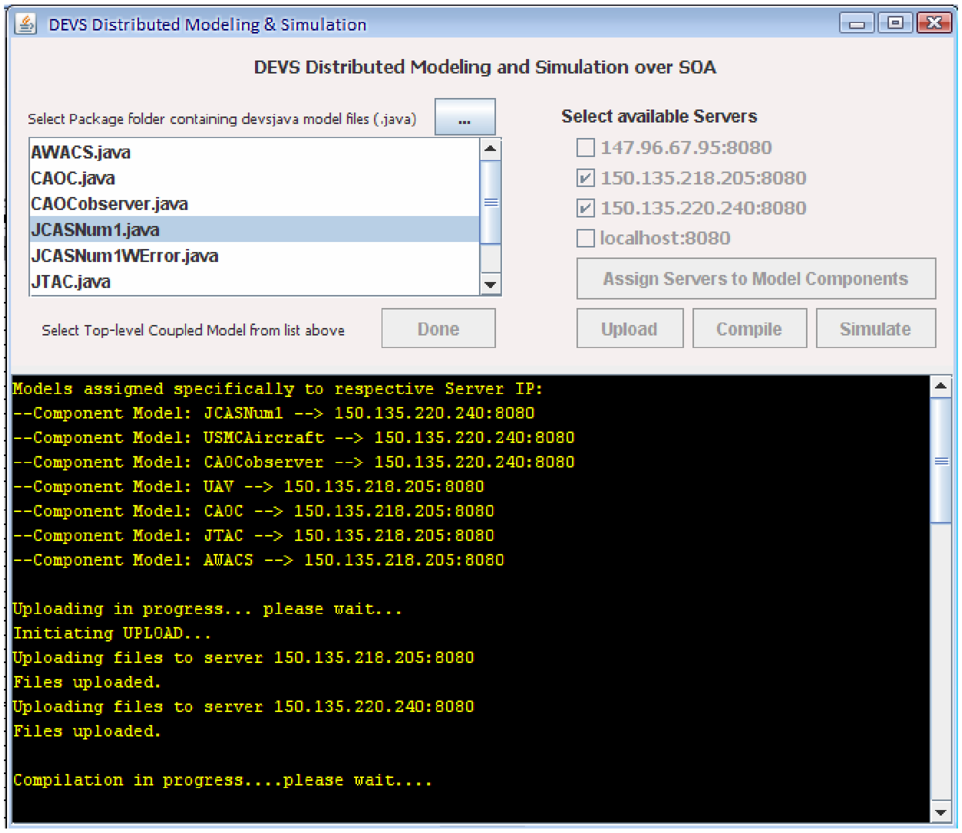
If CAOC fails then 'watchForError' gets triggered, thereby producing an ErrorReport message indicating that CAOC failed to produce the desired output message. A sample timout of 100 seconds is provided as fail-safe time that may incorporate CAOC recycling time in case of delays etc. If COAC produces output message *YouCanUseUSMCAircraft*, the CAOC Observer receives it as an external input message and resets the clock as CAOC performed its I/O pair correctly.

Similarly, observers for each of the DEVS models could be created automatedly.

8.1.5 Net-centric Execution of JCAS

Execution of JCAS DEVS models on net-centric SOA platform was done using the SOADEVS tool. The client application as described in Section 4.3 was used to execute the operation. Two servers were selected to demonstrate the concept (as shown in Figure 8.9). One server is located at ACIMS lab, University of Arizona and other server at

Spain, University Computense de Madrid. Also shown in the figure (in the console window) is the process of files being uploaded, compiled and the simulation-in-progress.


DEVS Distributed Modeling & Simulation
DEVS Distributed Modeling and Simulation over SOA
Select Package folder containing devsjava model files (.java)
...
AWACS.java
CAOC.java
CAOCobserver.java
JCASNum1.java
JCASNum1WError.java
JTAC.java
Select Top-level Coupled Model from list above
Done
Select available Servers
147.96.67.95:8080
150.135.218.205:8080
150.135.220.240:8080
localhost:8080
Assign Servers to Model Components
Upload
Compile
Simulate

```
Models assigned specifically to respective Server IP:
--Component Model: JCASNum1 --> 150.135.220.240:8080
--Component Model: USMCAircraft --> 150.135.220.240:8080
--Component Model: CAOCobserver --> 150.135.220.240:8080
--Component Model: UAV --> 150.135.218.205:8080
--Component Model: CAOC --> 150.135.218.205:8080
--Component Model: JTAC --> 150.135.218.205:8080
--Component Model: AWACS --> 150.135.218.205:8080

Uploading in progress... please wait...
Initiating UPLOAD...
Uploading files to server 150.135.218.205:8080
Files uploaded.
Uploading files to server 150.135.220.240:8080
Files uploaded.

Compilation in progress....please wait....
```

Figure 8.9: SOADEVS client running the JCAS model using Simulation services on two hosts

Finally, when the simulation is over, the console displays the following output. The simulation logs from both of the servers are categorically displayed. Figure 8.10 below shows the complete console log for all the operations done using SOADEVS client.

```
Models assigned specifically to respective Server IP:
--Component Model: JCASNum1 --> 150.135.220.240:8080
--Component Model: USMCAircraft --> 150.135.220.240:8080
--Component Model: CAOCobserver --> 150.135.220.240:8080
--Component Model: UAV --> 150.135.218.205:8080
--Component Model: CAOC --> 150.135.218.205:8080
--Component Model: JTAC --> 150.135.218.205:8080
--Component Model: AWACS --> 150.135.218.205:8080
```

```
Uploading in progress... please wait...
Initiating UPLOAD...
Uploading files to server 150.135.218.205:8080
Files uploaded.
Uploading files to server 150.135.220.240:8080
Files uploaded.

Compilation in progress....please wait....

Starting compilation at remote servers.....
Compiling project at 150.135.218.205:8080...
Project compiled.
Compiling project at 150.135.220.240:8080...
Project compiled.

Waiting to start SIMULATION....

Simulation in Progress....please wait...
Running simulation ...
11 iterations.
Simulators output:

150.135.218.205 output:
       JTAC    sending message: << port: ImmediateCASOut value: CASResourcesSpec
>>
State at: JTAC is: waitForAssignment
       AWACS   sending message: << port: requestImmediateCASOut value:
CASResourcesSpec >>
State at: AWACS is: doSurveillance
       CAOC    sending message: << port: readyOrderOut value: getReady port:
YouCanUseUSMCAircraftOut value: CASResources >>
State at: CAOC is: passive
       JTAC    sending message: << port: TACCommandOut value: initialAttack >>
State at: JTAC is: continueExecution
       UAV     sending message: << port: targetLocationOut value: (Lat,Long) >>
State at: UAV is: passive
       AWACS   sending message: << port: sitBriefOut value: sitBrief >>
State at: AWACS is: doSurveillance
       JTAC    sending message: << port: TACCommandOut value: ceaseAttack >>
State at: JTAC is: passive

150.135.220.240 output:
       USMCAircraft  sending message: << port: requestForTACOut value:
requestTAC >>
State at: USMCAircraft is: waitForTAC
       USMCAircraft  sending message: << port: sitBriefRequestOut value:
sitBriefRequest port: deconflictRequestOut value: requestDeconflict >>
State at: USMCAircraft is: attack
       USMCAircraft  sending message: << port: fireCommand value: fire >>
State at: USMCAircraft is: attack

SIMULATION over!
```

Figure 8.10: Simulation output at client's application using SOADEVS client

8.2 DoDAF-based Activity Scenario

8.2.1 Example: Implementation of an Activity Component

Consider an Activity as mentioned in Zinn [Zin04 pg 65] described in IDEF0 format (Figure 8.11). This activity is governed by the doctrines specified in OV-6a, IDEF3 format, which are described in [43]. Figure 8.12 is a sample OV-5 diagram for "select contractor" and Figure 8.13 is the OV-6a description in IDEF3 format where **X** represents a XOR split and **O** represents an OR split. These are the critical decision making points that impact the outcome of the Activity based on previous step. It is at this point, timing needs to be specified so that 'timeouts' can occur without leading to any ambiguity. Zinn acknowledged this problem in the process.

The information from these two figures is compiled manually to generate the pseudo code in the following format. This manual process amounts to the integration of OV-5 and OV-6a into a single document. The pseudo code is provided below in Figure 12.

The graphical representation in Figure 8.11 is represented textually through the Popkin System Architect as shown in Figure 8.14. Consequently Figure 8.13 and Figure 8.14 gives us the comprehensive information about the Activity, its purpose, its input-output information thru ICOM[3] lines, and pseudocode for operational rules (as defined in OV-6a). Figure 8.11, 8.12 and 8.13 describe a general step approach to arrive at this

[3] In IDEF0 diagrams, Inputs, Controls, Outputs and Mechanisms are collectively referred to as ICOM arrows.

pseudocode, which is then utilized by an agent based modeling software (e,g. SEAS) via Tactical Programming Language (TPL). Once pseudo code has been made available, any software developer who is versed with TPL or any other language can interpret it. This process is then followed for the case study (for all the 11 Activities) considered in [Zin04] Zinn. Zinn brought forward the information expressed in graphical format in OV-5 diagrams and OV-6a doctrines in the form of psedocodes that are realizable into software code. We utilize his efforts and demonstrate how this information can be used to feed the integrated DEVS methodology and development of OV-8 and OV-9.

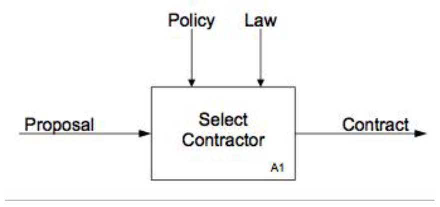


Figure 8.11: OV-5 diagram for "select contractor" in IDEF0 notation (from [Zin04])

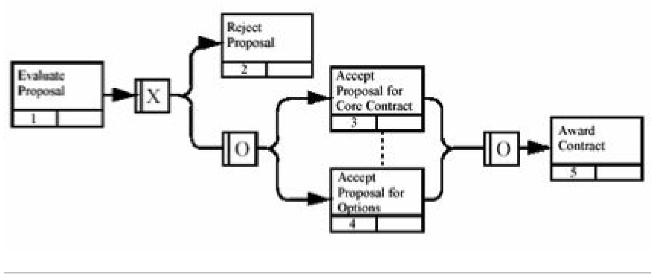

Evaluate Proposal
1
X
Reject Proposal
2
O
Accept Proposal for Core Contract
3
Accept Proposal for Options
4
O
Award Contract
5

Figure 8.12: OV-6a diagram for "select contractor" in IDEF3 notation, (from [Zin04])

Activity 1: Select Contractor

Description: The process used by the company to select the contractor for a new project
Inputs: Proposal: contains the cost, schedule, and technical information as proposed by the contractor
Outputs: Contract - the awarded contract
Controls:
Policy: Company contracting policy
Law: Federal, State and Local regulations

Pseudocode for Activity 1

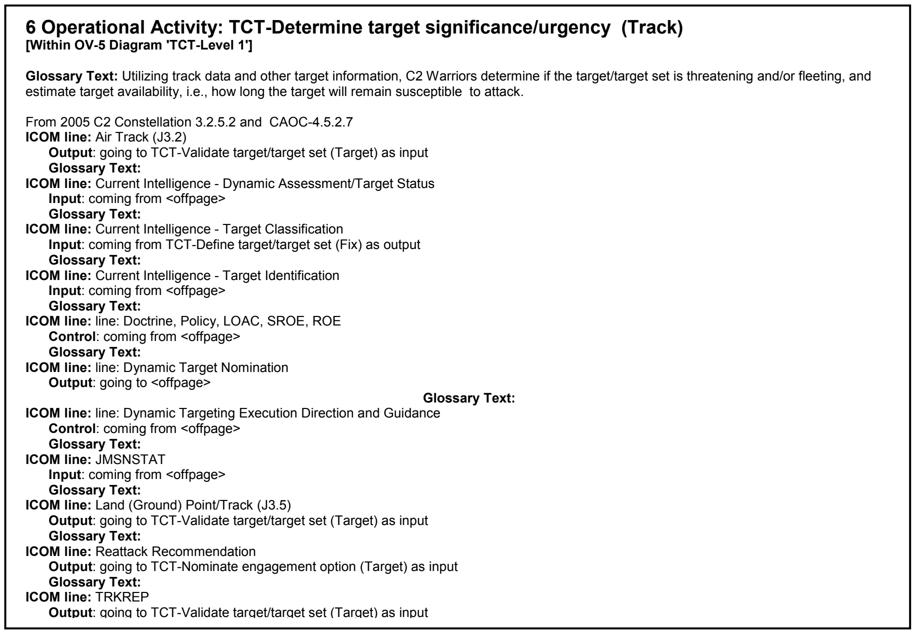
Evaluate Proposal

```
IF (cost > budget) THEN
                Reject Proposal
ELSE
                (Accept Proposal for Core Contract) OR
    (Accept Proposal for Options) OR
                ((Accept Proposal for Core Contract) AND (Accept Proposal for Options))
```

Figure 8.13: Pseudo Code as per Zinn's interpretation and integration procedure [Zin04]

8.2.2 Activity taken from Zinn as an example

Let us consider, the same example that is described in [Zin04]. Let the Activity that is being modeled is defined as Activity 6: TCT-Determine Target Significance/Urgency. There are about 11 Activities that are being evaluated and pseudo-code provided in [Zin04]. Figure 8.14 below provides the Activity Model Report as generated by Popkin System Architect.



6 Operational Activity: TCT-Determine target significance/urgency (Track)
[Within OV-5 Diagram 'TCT-Level 1']

Glossary Text: Utilizing track data and other target information, C2 Warriors determine if the target/target set is threatening and/or fleeting, and estimate target availability, i.e., how long the target will remain susceptible to attack.

From 2005 C2 Constellation 3.2.5.2 and CAOC-4.5.2.7
ICOM line: Air Track (J3.2)
  Output: going to TCT-Validate target/target set (Target) as input
  Glossary Text:
ICOM line: Current Intelligence - Dynamic Assessment/Target Status
  Input: coming from <offpage>
  Glossary Text:
ICOM line: Current Intelligence - Target Classification
  Input: coming from TCT-Define target/target set (Fix) as output
  Glossary Text:
ICOM line: Current Intelligence - Target Identification
  Input: coming from <offpage>
  Glossary Text:
ICOM line: line: Doctrine, Policy, LOAC, SROE, ROE
  Control: coming from <offpage>
  Glossary Text:
ICOM line: line: Dynamic Target Nomination
  Output: going to <offpage>
  Glossary Text:
ICOM line: line: Dynamic Targeting Execution Direction and Guidance
  Control: coming from <offpage>
  Glossary Text:
ICOM line: JMSNSTAT
  Input: coming from <offpage>
  Glossary Text:
ICOM line: Land (Ground) Point/Track (J3.5)
  Output: going to TCT-Validate target/target set (Target) as input
  Glossary Text:
ICOM line: Reattack Recommendation
  Output: going to TCT-Nominate engagement option (Target) as input
  Glossary Text:
ICOM line: TRKREP
  Output: going to TCT-Validate target/target set (Target) as input

Figure 8.14: Activity Report Model for Activity 6 generated thru Popkin System Architect

This Activity Report is nothing but the interface descriptions for an Activity in OV-5 diagram. It tells us that Activity 6 receives input from which other Activities and sends outputs to which Activities. It also provides us the information about the 'control' interfaces that are needed to execute the doctrines and rules. Figure 8.15 depicts the IDEF3 model that implements the OV-6a doctrines and rules for Activity 6.

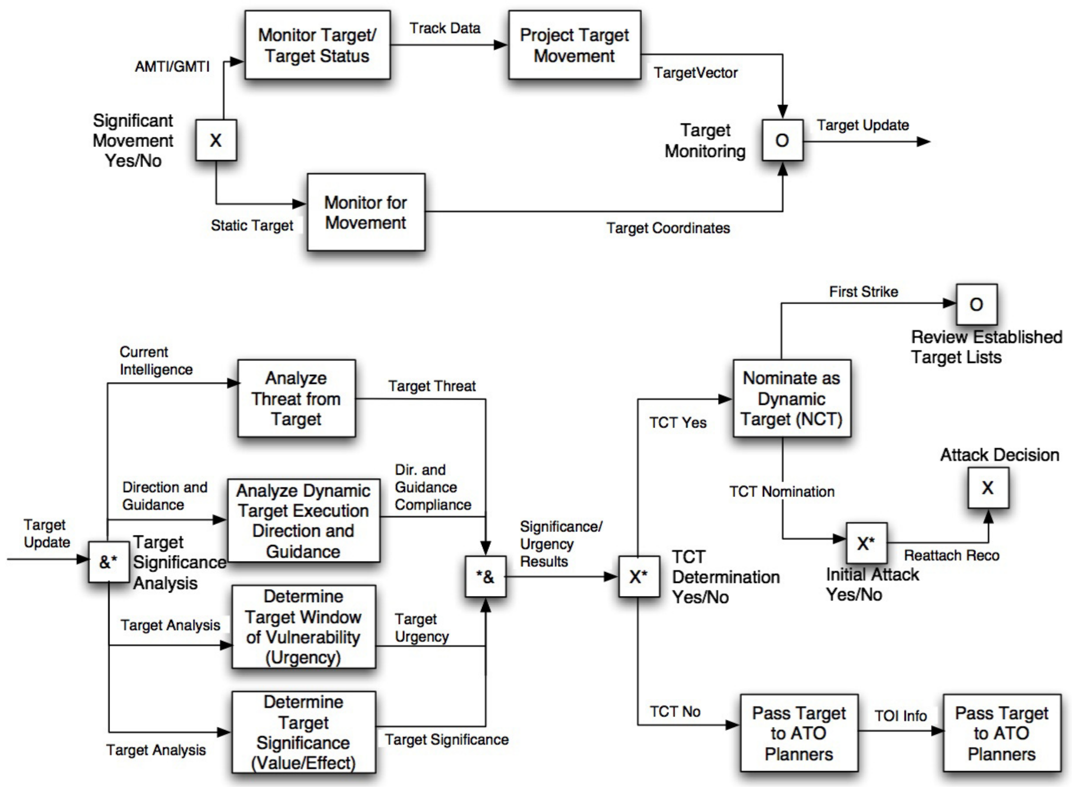

Monitor Target/ Target Status
Track Data
Project Target Movement
AMTI/GMTI
TargetVector
Significant Movement Yes/No
X
Target Monitoring
O
Target Update
Monitor for Movement
Static Target
Target Coordinates
First Strike
O
Review Established Target Lists
Current Intelligence
Analyze Threat from Target
Target Threat
Nominate as Dynamic Target (NCT)
TCT Yes
Attack Decision
X
Direction and Guidance
Analyze Dynamic Target Execution Direction and Guidance
Dir. and Guidance Compliance
TCT Nomination
Target Update
&*
Target Significance Analysis
Significance/ Urgency Results
*&
X*
TCT Determination Yes/No
X*
Reattach Reco
Initial Attack Yes/No
Target Analysis
Determine Target Window of Vulnerability (Urgency)
Target Urgency
Target Analysis
Determine Target Significance (Value/Effect)
Target Significance
TCT No
Pass Target to ATO Planners
TOI Info
Pass Target to ATO Planners

Figure 8.15: IDEF3 representation of Activity 6 (“Conduct Dynamic Assessment of Target” TCT 2005 Architecture, 2003: OV-6a) [Zin04]

The pseudocode for Activity 6 is provided in Figure 8.16 which is compiled manually from the information contained in OV-6a. For complete description of the Activity 6, refer to [Zin04]. Briefly, the context of Activity 6 in TCT architecture is immediately after a target (or target set) is found and fixed. The upper half of Figure 8.15 shows an XOR junction that indicates only one path be taken. The resulting “target update” is then put thru 4 simultaneous analyses indicated by AND junction. This results (after integrated processing) into “Is the target time critical?” If it passes this TCT test it is again presented with a decision-point “Is the initial attack on the target?” The answer to this question results in two different modes of action, indicated by XOR junction. Zinn acknowledges the fact that even though there is certain sequencing present, precise information about the rules defined are left to imagination.

```
IF Significant Movement of target
Then Monitor Target/Target Status
      Project Target Movement
      Target Vector = . . . .. ?
Else Monitor for Movement

Analyze Threat from Target (is the target closing on Friendlies or Fleeing?)
Analyze Dynamic Targeting Ex Direction and Guidance (does this agree with the commander’s
requirements?)
Determine target window of vulnerability (urgency)
Determine target significance – partly based on above findings

IF it is determined to be a TCT based on the above info
Then IF this is the first strike attempt on this target
      Then Goto Activity 7 (Validate Target/Target set)
      Else Goto Activity 8 (Nominate engagement option)
Else Pass target to ATO Planners
      Monitor Target of Interest for Status Change
```

Figure 8.16: Pseudocode for Activity 6 – based on IDEF3 diagram in Figure 8.15, taken from [Zin04]

The next section demonstrates how the information in Figure 8.14 and Figure 8.16 is transformed into DEVS component modeling framework. It also shows how OV-8 and OV-9 gets populated. However, it must be realized that an “operational node” hasn’t been defined with respect to the current example. Consequently, we will assume an entity structure that will illustrate the concept.

8.2.3 DEVS Interpretation of Activity 6

Based on the available information let us assume that dynamic target assessment happens at a particular node. Assume that Activity 6 and its sub-activities are all happening at TCT. Let's call this Operational Node 1, with Id O1. This will comprise our OV-2 diagram containing only one Operational Node executing all the 11 activities [Zin04]. Again, a simple example has been considered to demonstrate the construction of the new OV document, namely OV-8 and OV-9.

The following Table 8.3 assigns identification numbers to various activities

S.No.	Activity	Sub-Activity	Internal-Activity	ID
1	**Activity 6**	Dynamic Target Assessment		A6
2.		Monitor Target/ Target Status		A6.1
3.		Monitor for Movement		A6.2
4.		Project Target Movement		A6.3
5.		Analyze Threat from Target		A6.4
6.		Analyze Dynamic Target Execution/Direction and Guidance		A6.5
7.		Determine Target Window of Vulnerability (Urgency)		A6.6
8.		Determine Target Significance (Value/Effect)		A6.7
9.		Nominate as Dynamic Target (NCT)		A6.8
10.		Pass Target to ATO Parameters		A6.9
11.		Pass Target to ATO Planners		A6.10
12.			Significant Movement Yes/No	A6.11
13.			Target Monitoring	A6.12
14.			Target Significance Analysis	A6.13
15.			Synthesize Results	A6.14
16.			TCT Determination Yes/No	A6.15
17.			Initial Attack Yes/No	A6.16
18.			Review Established Target Lists	A6.17
19.			Attack Decision	A6.18

Table 8.3: Activity-ID mapping for OV-8 and OV-9

Based on the IDEF3 diagram (graphical information for OV-6) in Figure 8.15, and our constructed OV-2 in previous paragraph, we can construct our OV-8 document that lists Activities and their *logical* interface information. We need such port information to be able to create components. Such logical-port construction has been attempted in [Tel04] where the focus was to create an SV executable model. Developing and specifying Activity port-interfaces at this level is a logical step towards SV interface design as tractability is ensured. The OV-8 document below does not address the performance issue at OV level and its refined structure is presented in [Mit06b]. A sample OV-8 document looks like the following Table 8.4:

S.No.	**Activity ID component**	**Connection ID**	**Source Activity**	**Input Interface Name (Logical Port)**	**Message Description /OIEs**	**Container Op Node**	**Source document/ diagram**
1	**A6**					O1	
2.	A6.1	CA6.1	A6.11	inSigMovY	AMT/GMTI	O1	Figure 12/OV-6b,c
3.	A6.2	CA6.2	A6.11	inSigMovN	StaticTarget	O1	Figure 12/OV-6b,c
4.	A6.3	CA6.3	A6.1	inTrkData	TrackData	O1	Figure 12/OV-6b,c
5.	A6.4	CA6.4	A6.13	inCurrInte	Current Intelligence	O1	Figure 12/OV-6b,c
6.	A6.5	CA6.5	A6.13	inDirGuid	Direction and Guidance	O1	Figure 12/OV-6b,c
7.	A6.6	CA6.6	A6.13	inTarAnaly	Target Analysis	O1	Figure 12/OV-6b,c
8.	A6.7	CA6.7	A6.13	inTarAnaly	Target Analysis	O1	Figure 12/OV-6b,c
9.	A6.8	CA6.8	A6.14	inTctYes	TCT Yes	O1	Figure 12/OV-6b,c
10.	A6.9	CA6.9	A6.14	inTctNo	TCT No	O1	Figure 12/OV-6b,c
11.	A6.10	CA6.10	A6.9	inToiInfo	TOI Info	O1	Figure 12/OV-6b,c

12.	A6.11	CA6.11		inIsSigMov	Significant Movement	O1	Figure 12/OV-6b,c
13.	A6.12	CA6.121	A6.2,	inTargCoord	Target Coordinates	O1	Figure 12/OV-6b,c
		CA6.122	A6.3	inTargVec	Target Vector	O1	Figure 12/OV-6b,c
14.	A6.13	CA6.13	A6.12	inTarUpdate	Target Update	O1	Figure 12/OV-6b,c
15.	A6.14	CA6.141	A6.4	inTarThreat	Target Threat	O1	Figure 12/OV-6b,c
		CA6.142	A6.5	inDGCompl	Direction Guidance Compliance	O1	Figure 12/OV-6b,c
		CA6.143	A6.6	inTarUrg	Target Urgency	O1	Figure 12/OV-6b,c
		CA6.144	A6.7	inTarSig	Target Significanc e	O1	Figure 12/OV-6b,c
16.	A6.15	CA6.15	A6.14	inSigUrgRes	Significanc e/Urgency Results	O1	Figure 12/OV-6b,c
17.	A6.16	CA6.16	A6.8	inTctNom	TCT Nomination	O1	Figure 12/OV-6b,c
18.	A6.17	CA6.17	A6.16	inFirstStr	First Strike	O1	Figure 12/OV-6b,c
19.	A6.18	CA6.18	A6.16	inReAtkRec	Reattack Recommen dation	O1	Figure 12/OV-6b,c

Table 8.4: Sample OV-8 document

Based on the information provide in Figure 8.15, we have constructed and identified the interfaces that are being used by different activities to communicate. However, we have not considered the information contained in Figure 8.17 that describes how Activity 6 communicates with other activities. We did not explore connectivity between other destination activities just to keep the example in the needed perspective. However, the procedure is essentially the same with more rows being added to the above table. To give a glimpse on how this interconnected activities (as components) will perform in tandem; notice the inports and outports of Activity 6 in Figure 8.17. The other Activities are shown in the figure below don't have any resemblance to the actual example in [Zin04].

They are just meant for understanding. To understand how Activity 6 works internally based on the different activities in Table 8.3, please look at Figure 8.15.

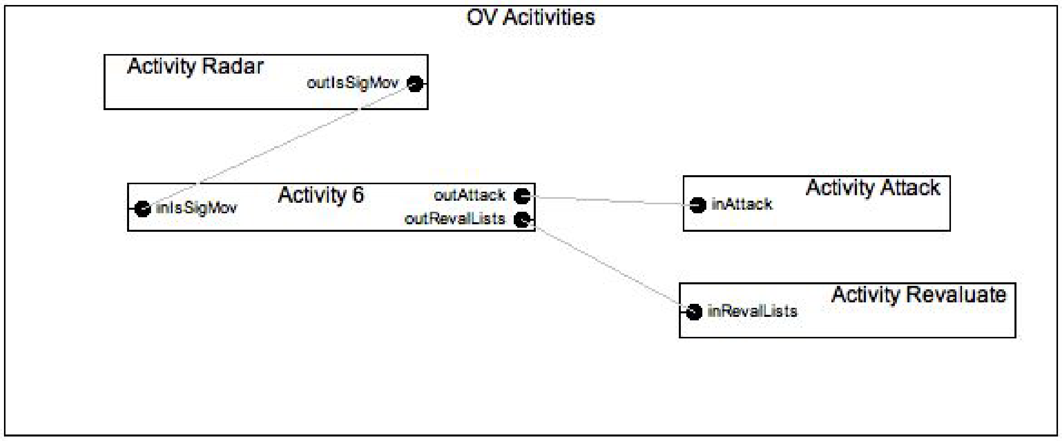


Figure 8.17: DEVS interrelationships of Activity 6 with other Activities.

The coupling relations shown in Figure 8.18 are generated in an automated manner from the data presented in Table 8.4. Columns 2,3,4 and 5 provide sufficient information to generate the following lines of code with simple string manipulations. Consequently, an automated generation of DEVS model is realizable. Hence OV-8 document provides sufficient information to develop a skeleton DEVS model that can make its entry into the Model-repository. Let's name the Model for Acitivity6 as MA6. The inner models are identified in the same predictable manner as MA6.1, MA6.2…MA6.18.

```
ViewableAtomic a61 = new ViewableAtomic("A6.1");
add(a61);
ViewableAtomic a62 = new ViewableAtomic("A6.2");
add(a62);
…..
ViewableAtomic a611 = new ViewableAtomic("A6.11");
add(a611);
….
a611.addOutport("outSigMovY");
```

```
a61.addInport("inSigMovY");
addCoupling(a611,"outSigMovY",a61,"inSigMovY");

a611.addOutport("outSigMovN");
a62.addInport("inSigMovN");
addCoupling(a611,"outSigMovN",a62, "inSigMovN");
…..
```

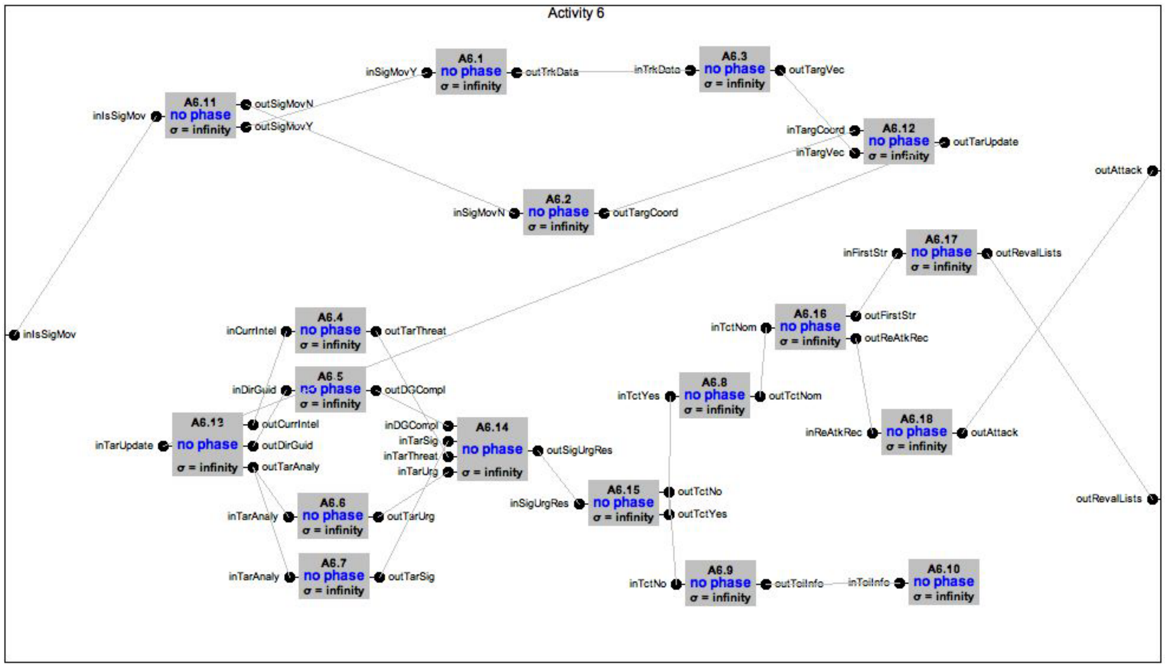


Figure 8.18: DEVS description of Activity 6 in relation to Table 6 Activity components.

The next task in line is the inclusion of pseudo code that contains the doctrines and rules form OV-6a, described in Figure 8.16. Consider these 4 initial lines from Figure 8.16.

```
IF Significant Movement of target
Then Monitor Target/Target Status
     Project Target Movement
     Target Vector = . . . .. ?
Else Monitor for Movement
```

This particular doctrine is to be implemented at A6.11 (refer Table 8.3). This has far reaching advantages. By assigning doctrines and rules to specific Activity components, we are ensuring that each rule is formally implemented and is synchronized with other rules that are 'in operation' at that instant of time. In a sense, which rules are compatible and which can cause 'deadlocks' can be determined by execution of the above Activity6 DEVS model. The sample lines above are implemented in the ***deltext()*** function of component A6.11. The ***deltint()*** function defines the natural course of the activity.

```
public void deltext(double e, messagex){
….
      for(int i=0; i<x.length; i++){
            if(messageOnPort("inIsSigMov"){
                  MessageTypeA msg = (MessageTypeA)x.getValOnPort(i,
"inIsSigMov");
                  If(msg.equals("yes"))
                        holdIn(0, "yesSigMov");
                  else
     if(msg.equals("no"))
                        holdIn(0, "noSigMov"));
      }
…..
}
public message out(){
….
      if(phaseIs("yesSigMov")){
            m.add(makeContent("outSigMovY", new entity("start")));

      if(phaseIs("noSigMov"))
            m.add(makeContent("outSigMovN", new entity("start")));
……
}
```

Similarly, all other Activities will receive inputs from other source Activities in their ***deltext()*** functions that will contain the logic for implementation of doctrines. For convenience purposes, the execution time of these doctrines is considered zero. Notice the ***holdIn()*** function in the code above. However, this is an important place where we can tune and implement the realistic time in issuing commands by human commanders. For example, in a situation where the system is 'waiting' for a command from an authority figure and decision has to arrive until a 'time-out' occurs. In addition, consider that the activity component is executing certain process with respect to its ***deltint()***

function and is in certain ‘phase’ waiting for any external input from other activities. In the situation of not receiving this input within allowable time-window, time-outs can very effectively guide the simulation to its completion and prevent the wait-to-infinity problem.

The OR split problem pointed out by Zinn in IDEF3 methodology has no effect in DEVS methodology. This problem is resolved by making the **&**, **X**, and **O** constructs in IDEF3 methodology as ‘internal-activity’ components (Table 8.3). Once they are componentized, time-outs can be implemented very easily that will completely eradicate this problem. These components are very well documented in DEVS *SimpArc* package Version 3.0. This solution also puts the focus back on the system-logic implementation and test if the communication delays are significant enough that time-outs are occurring frequently.

Finally, the last task is the description of OV-9 document. This document contains information about the Activities happening inside an Operational Node and how the sub-activities are mapped on to the components inside the Operational Node. For simplicity, we are working on the assumption that there is only one Operational Node O1 in the example. As there is no information present on what are its inner components are in [Zin04], we will assume that there are, let’s say, 7 inner components that make up this Node. Four of these seven components are associated with Activity6 and the other three components are associated with some other activities, not considered for illustration purposes.

S.No.	Operational Node	Inner Component Entities	Component Name	Associated Models added to Repository	Hierarchical Parent/Container	DEVS Model Type
1.	**O1**	OCE1	TCT	ME1	-	Digraph
2.		OCE1.1	Radar Tracking System	ME1.1	ME1	Atomic
3.		OCE1.2	Significance Analyzer	ME1.2	ME1	Atomic
4		OCE1.3	Urgency Analyzer	ME1.3	ME1	Atomic
5		OCE1.4	Vigilance Controller	ME1.4	ME1	Atomic
6.		OCE1.5	Attack Evaluator	ME1.5	ME1	Digraph
7.		OCE1.6	Attack Initiator	ME1.6	ME1.5	Atomic
8.		OCE1.7	Attack Terminator	ME1.6	ME1.5	Atomic

Table 8.5: Inner components within Operational Nodes and their mapping with ‘standardized’ DEVS models

The defined components are essentially COTS components with defined behavior. They can even come from System View document SV-4. Consequently, each of them has their ‘models’ for simulation purposes specified in DEVS formalism. These models are essentially Open-source models available to public thru a common repository and are ‘standardized’. The following table depicts the information assumed for construction of OV-9. The inner components depicted in the table below are only for illustration purposes.

Having Table 8.5 as available resources for OV-9, we have enough information to construct the Activity-entity mapping in Table 8.6. We identify and define port-interfaces that need to be added to the entity component models so that they can be coupled to the Activity components. Once OV-9 document is in place, the added interface information is used to update the models defined during the construction of these two documents. We saw in construction of OV-8 document that the resulting model is a stand-alone model that is capable to execute the simulation in 'capability' mode, testing the OV-5 and OV-6 description of the system. A sample OV-9 document looks as following:

S.No.	**Operational Node**	**Inner Component Entities**	**Component Name**	**Activity Component**	**Activity Component Name**	**Interface description**
1.	**O1**	OCE1	TCT			
		OCE1.1	Radar Tracking System	A6.1	Monitor Target/Target Status	monTarE
				A6.2	Monitor for Movement	monTarMovE
				A6.3	Project Target Movement	proTarMovE
				A6.11	Significant Movement Yes/No	sigMovYesNoE
				A6.12	Target Monitoring	tarMonE
				A6.10	Monitor Target of Interest for Status change	monTarInterE
		OCE1.2	Significance Analyzer	A6.13	Target Significance Analysis	tarSigAnalyE
				A6.4	Analyze threat from Target	analyThrTarE
				A6.5	Analyze Dynamic Target Execution Direction and Guidance	analyEDGE
				A6.7	Determine	detTarSigE

					Target Significance	
				A6.14	Synthesize Results	syncE
		OCE1.3	Urgency Analyzer	A6.6	Determine Target Window of Vulnerability	detWinVulE
		OCE1.4	Vigilance Controller	A6.15	TCT Determination Yes/No	tctDetYesNoE
				A6.8	Nominate as dynamic Target	nomDynTarE
				A6.9	Pass Target to ATO Planners	passTarAtoE
				A6.16	Initial Attack Yes/No	initAtckYesNoE
				A6.18	Attack Decision	atckDecE
				A6.17	Review established Target Lists	revEstTarListsE
		OCE1.5	Attack Evaluator	A6.16	Initial Attack Yes/No	initAtckYesNoE
		OCE1.6	Attack Initiator			
		OCE1.7	Attack Terminator			

Table 8.6: OV-9 description document mapping the Entity component inside Operational Node O1 with the Activity Components defined in OV-8 with port-interfaces

OV-9 document aids in bringing the systems perspective to the design and how the system's components initiate the designated Activities. Assignment of an Activity to appropriate component entity is a job of an experienced 'designer', as per the definition of Designer in DoDAF document. This document ensures accountability that there is at least one Component entity that is responsible for the execution of that particular Activity. Notice that all the Activity Components addressed in the example have been assigned at least one Operational Node inner component entity. After the creation of OV-

9 document, the Interface information, in the last column, is used to update the corresponding Activity and the Entity models in the Model Repository that were created during the construction of OV-8. This is again an automated task with simple string manipulation as described earlier, during the construction of OV-8 models.

Hence, during the creation of OV-8 and OV-9 we have populated the Model Repository with Activity Models (MA6.1-MA6.18) and Operational Node's inner component models (ME1, ME1.1-ME1.6), have created an interface between these two aspects of DoDAF design.

8.2.4 Synopsis

Looking Figure 4.18 in an Activity component perspective, we have our defined inputs and outputs, and eventually the activity-ports. In the example above, we have defined the interfaces of an Activity that could be subjected to component coupling and testing. The coupling information can be integrated using OV-3 document, as described in Table 8.4. The timing information is added using the OV-6b and OV-6c diagrams as we have defined 'components', the effects of which have been highlighted in [Mit06a]. This information, along with the pseudocode provided by Zinn, is integrated to develop the DEVS model of the Activity in question. The pseudo code is very well directed to the Activity that is best responsible to execute those 'rules'. At this point the whole purpose of creating OV-8, the rule-Activity mapping, is realized.

OV-9 document deals with the mapping of the Activity components with the entity components. Since Zinn [31] did not define internal components for any Operational Node, we assumed certain inner components and mapped the Activities to these components. Having ensured accountability for each of the Activities, another area that OV-9 contributes to is System Design, Reuse and Composability. We have available with us a document that contains information of the functionalities any particular component can perform or participate in collective functionality. Consider the situation when two or more inner components, from Systems perspective are thrown together to observe, if the system is capable of performing 'something'. This allows us to experiment with different systems who are claiming to exhibit certain functionality. It allows us to test interoperability.

Hence, the resulting integrated information from OV-3, OV-2 and OV-6 is converted to the information in documents OV-8 and OV-9, with the addition of logical ports, dedicated to the M&S area that are focused towards Operational Views.

8.3 Link-16 ATC-Gen Project at JITC

In this section, a testing approach to Link16 standards conformance is described. For details see [Mak06]. The auto correlation experiment is conducted using the automated test generation processes shown in Figure 8.19. The scenario was performed against the Integrated Architecture Behavior Model (IABM) developed by the Joint SIAP System Engineering Organization (JSSEO). The result of this scenario was verified by the ATC-

Gen Test Driver and validated using JITC's Simple J network packet monitoring tool. Due to the classification of this system, the experimental results can not be shown. Thus, the System under Test (SUT) test models are developed to allow the test driver to act as the SUT and allow the experiment to be conducted.

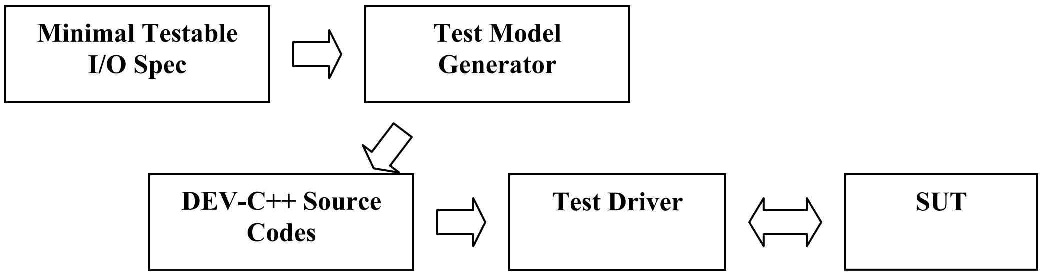


Figure 8.19: Automated Testing

8.3.1 Auto Correlation Scenario

As MIL-STD 6016C stated, when a system receives a remote track from a remote system that is within the correlation window of the local track, it initiates the tentative correlation process. If a second track arrives within the local track correlation window, it shall be correlated and held as common local track by transmitting a correlation request to the remote system. If the local track number is greater than the remote track number, the local system drops its own track and sends out a drop track notification; otherwise, the remote system drops its track and sends out the notification. Figure 8.20 illustrates the auto correlation process in the sequential diagram.

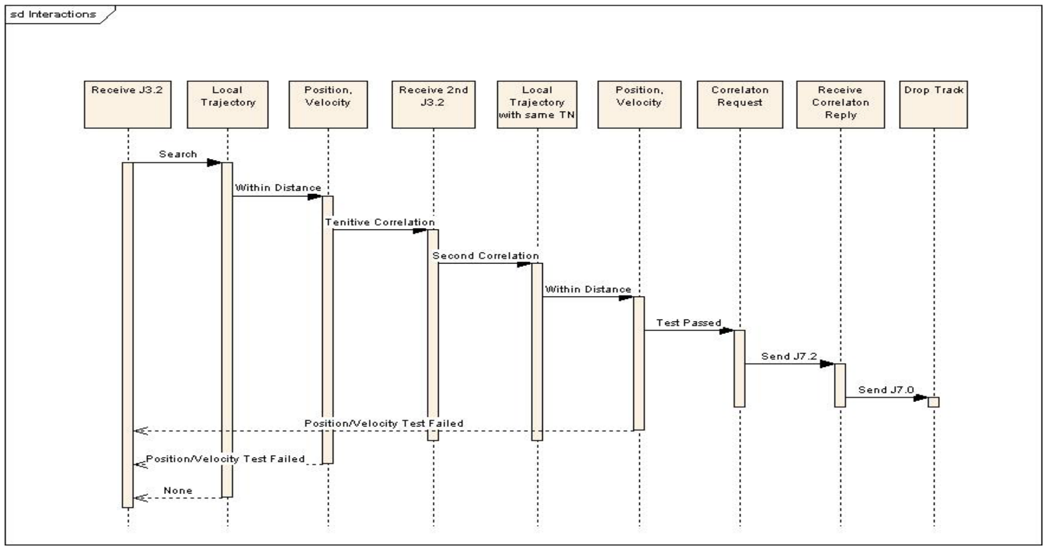


Figure 8.20: Auto Correlation Sequential Diagram

The test engineer follows the sequential diagram to construct the minimal testable pairs. Furthermore, the test models are generated using the Test Model Generator. Figure 8.21 illustrates the minimal testable pairs for SUT and Test Driver.

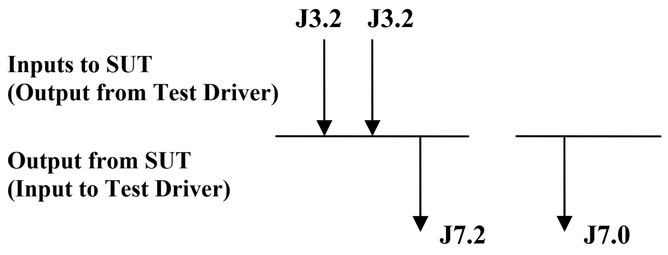


Figure 8.21: Minimal Testable I/O pairs for Auto Correlation

8.3.2 Auto Correlation Experiment Setup & Results

The auto correlation scenario is created to demonstrate the correctness of the models generated by the Test Model Generator. The models are implemented into the SUT and Test Model Test Drivers and communicate via Simple J protocol as illustrated in Figure

8.22. The transmissions and the receipt of the Simple J messages of the scenario are captured by a Simple J network packet monitoring tool. The packet monitor captures and decodes the Simple J messages, and the messages are saved into a log file. The log file is analyzed and the data is verified to ensure that the scenario data is the intended behavior of the Test Driver.

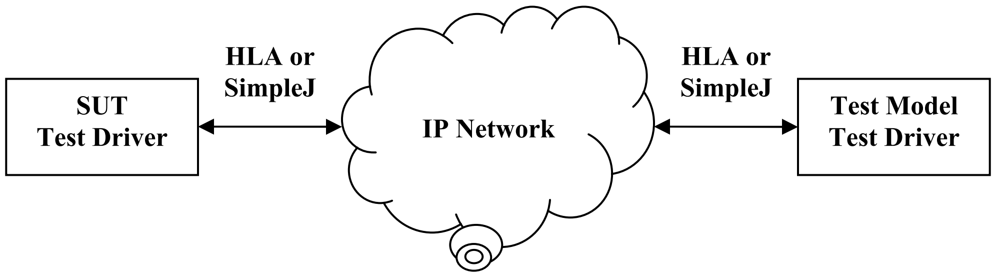
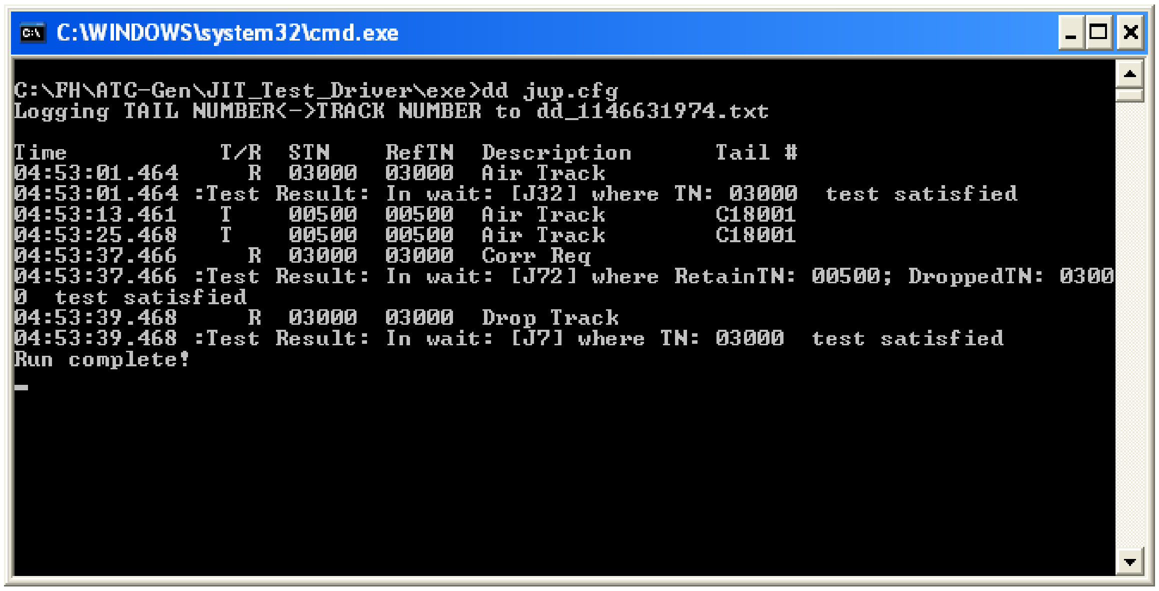


Figure 8.22 Test Drivers Setup Diagram

Successful Auto Correlation

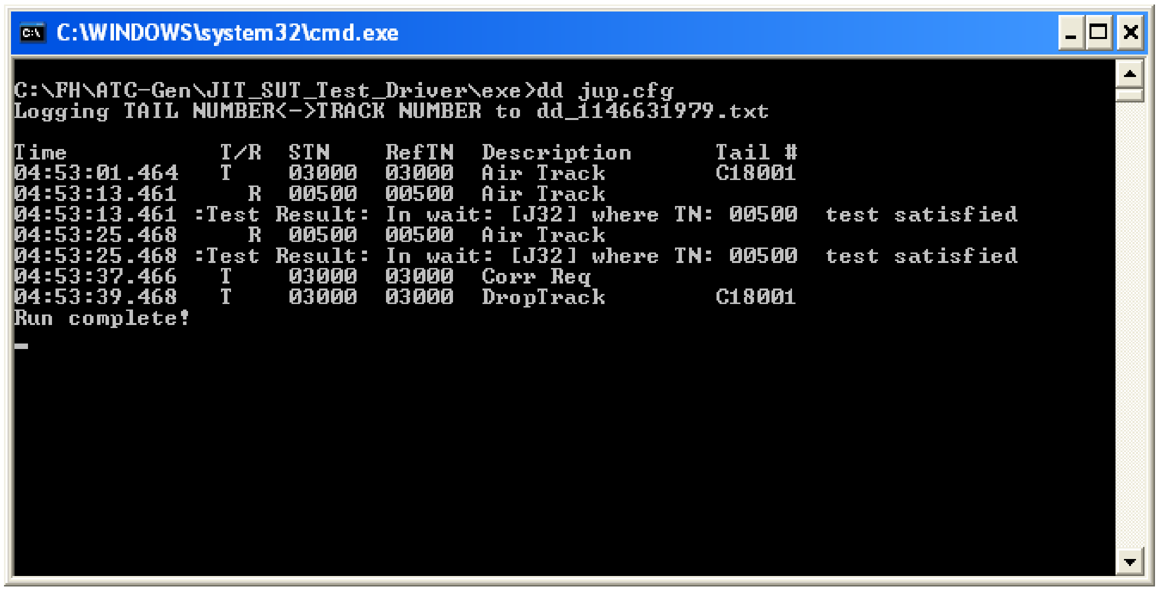
In this scenario, the Test Drivers are communicated via Simple J protocol. The messages setup in the correct sequence and auto correlation is induced. The SUT models and Test Models are generated by the Test Model Generator, and implemented into the Test Driver. The SUT TD has the track number of 03000, and the Test Model TD has the track number of 00500. The two J3.2 track positions of the Test Model TD are exactly the same as the SUT J3.2 track position. This causes the tracks to correlate and creates a common local track with the track number of 00500. The SUT TD sends a correlation request and drops the local track with the track number of 03000. Figure 8.23 illustrates the outputs from the Test Model TD, and Figure 8.24 illustrates the results for the SUT TD.



C:\WINDOWS\system32\cmd.exe

```
C:\FH\ATC-Gen\JIT_Test_Driver\exe>dd jup.cfg
Logging TAIL NUMBER<->TRACK NUMBER to dd_1146631974.txt

Time           I/R  STN     RefTN   Description       Tail #
04:53:01.464      R  03000  03000   Air Track
04:53:01.464 :Test Result: In wait: [J32] where TN: 03000  test satisfied
04:53:13.461    T    00500  00500   Air Track         C18001
04:53:25.468    T    00500  00500   Air Track         C18001
04:53:37.466      R  03000  03000   Corr Req
04:53:37.466 :Test Result: In wait: [J72] where RetainTN: 00500; DroppedTN: 0300
0  test satisfied
04:53:39.468      R  03000  03000   Drop Track
04:53:39.468 :Test Result: In wait: [J7] where TN: 03000  test satisfied
Run complete!
_
```

Figure 8.23: Test Model Test Driver successful Auto Correlation scenario



C:\WINDOWS\system32\cmd.exe

```
C:\FH\ATC-Gen\JIT_SUT_Test_Driver\exe>dd jup.cfg
Logging TAIL NUMBER<->TRACK NUMBER to dd_1146631979.txt

Time           I/R  STN     RefTN   Description       Tail #
04:53:01.464    T    03000  03000   Air Track         C18001
04:53:13.461      R  00500  00500   Air Track
04:53:13.461 :Test Result: In wait: [J32] where TN: 00500  test satisfied
04:53:25.468      R  00500  00500   Air Track
04:53:25.468 :Test Result: In wait: [J32] where TN: 00500  test satisfied
04:53:37.466    T    03000  03000   Corr Req
04:53:39.468    T    03000  03000   DropTrack         C18001
Run complete!
_
```

Figure 8.24: SUT Test Driver successful Auto Correlation scenario

8.3.3 Testing Status

ATC-Gen Test Driver was tested in both standalone and distributed environments. In the standalone environment, it performed Link 16 testing against two Link-16 systems: IABM and Air Defense System Integrator (ADSI). Recently, ATC-Gen Test Driver was participated in a distributed live testing environment in JITC. Table 8.7 summaries the results of the Link 16 functionalities against the systems. In recent developments, three modes of testing have been developed: Active, Reactive and Passive. The Reactive mode is developed at JITC by Dale Fulton and others. In this approach, the user interactively is able to inject parameters to tune SUT to real system. This allows the capability to use the same SUT model with different real-world implementations by dynamically tuning the SUT model. The reactive mode of changing system parameters is similar to reactive-manner of controlling model or simulator parameters discussed in Section 3.4.3.

Link 16 Systems **MIL-STD 6016C Functions**	**IABM**	**ADSI**	**Distributed Environment**
AutoCorrelation	Y	Y	Y
Correlation Window Size	N	Y	Y
Decorrelation	Y	Y	Y
Track Management	Y	N	N
Report Responsibility	Y	Y	Y
Track Quality	Y	Y	Y
Identity Different Resolution	N	Y	Y

Table 8.7: Link 16 functionalities vs. Systems

8.4 GENETSCOPE Project at JITC

SCOPE command is a highly automated, high-frequency (HF) communication system that links U.S. Air Force (USAF) command and control (C2) functions with globally deployed strategic and tactical airborne platforms. SCOPE command replaces existing USAF high-power HF stations with a communication system featuring operational ease of use, dependability, and seamless end-to-end connectivity comparable to commercial telephone services. The network consists of fifteen worldwide HF stations (see Figure 8.25) interconnected through various military and commercial telecommunications media (see Figure 8.26). It increases overall operational and mission capabilities while reducing operation and maintenance costs.

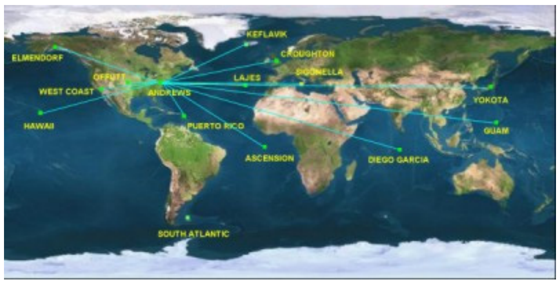


Figure 8.25: Geographic locations of fixed stations

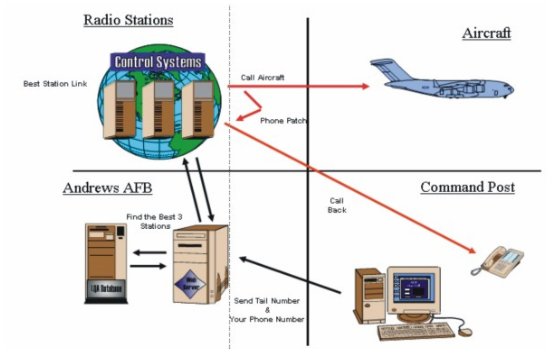


Figure 8.26: Communication flow diagram for SCOPE command

The HF radio equipment includes the Collin's Spectrum DSP Receiver/Exciter, Model RT-2200. The radios feature Automatic Link Establishment (ALE) and Link Quality Analysis (LQA) capability and are adaptable to future ECCM waveforms FSK, MIL-STD-188-110B, and STANAG 5066. The transmit subsystem includes 4-kW solid-state power amplifiers, a high-power transmit matrix, and a combination receive/multicoupler antenna matrix. A typical SCOPE command station includes operator consoles (HFNC), circuit switching equipment (DES, DSN, LCO), HF radios (ALEs), RF matrixes (RTs), and antennas (RXs, TXs). A non-blocking digital electronic switch (DES) connects the station to the local military and/or commercial telecommunication services. The switch features unlimited conferencing, modular sizing, a digital switch network, a precedence function, and capacity for up to 2,016 user lines.

SCOPE command uses a modular, open-system design to automatically manage and control all network operations, including those at split-site stations. To achieve maximum flexibility, the system uses commercially available standards-based software and a multitasking operating system. This approach permits fourteen out of fifteen network stations to operate "lights out" (unmanned) and to be economically controlled from a central location. The control system also includes LAN software, servers, and routers to support unlimited LAN/WAN.

The program includes a Systems Integration Lab (SIL) and test-bed facility located in Rockwell Collins's Texas facility. The SIL is used to predict the impact and risk that any changes or upgrades will have on system performance, integrity, or costs before actual implementation begins. The SIL includes a fully functional SCOPE command station for performing baseline design verification, and interface compatibility and functional verification tests.

Joint Interoperability Test Command (JITC) is the only government agency that is assigned the task to validate and authorize IT systems for military operations [CJC06]. The HF SCOPE command system has also been evaluated by JITC. In collaboration with Dr. Eric Johnson, a simulator was developed in the C language around 1997 that was validated and eventually used by both the government and the industry to conduct experiments and run scenarios. The simulator was an exhaustive and comprehensive effort with respect to the details it implemented and served its purpose well. However, in today's circumstances, the same simulator is obsolete due to the heterogeneous nature of today's network traffic, in which e-mail occupies a considerable percentage of traffic. The simulator is now being upgraded at the ACIMS lab in order to make it more useful for current demands. These demands stem from the possibility of expansion of the current infrastructure of the SCOPE command. Questions arise such as how many stations need be added to service a required workload. Also needing to be investigated are trade-offs such as whether it is more economical to add more stations or increase the number of internal radio levels within stations to meet the anticipated demands. Air traffic has increased manifold since 1997, along with the computing technology. Consequently, the transition effects need to be monitored more closely, and the overall system response time[4] needs to be documented. The significant parameters that have the most impact on system performance have to be identified. To more easily address such questions, an effort is being made to modularize Johnson's 15K lines of code into a component-based structure depicted in Figure 8.27. Once "componentized," the components are made

[4] Response time of a system is defined as the time taken by the system to display significant effect caused by any update in the configuration parameters.

DEVS compliant resulting in a DEVS-based simulation package to support the systems engineering needs of the SCOPE command.[5]

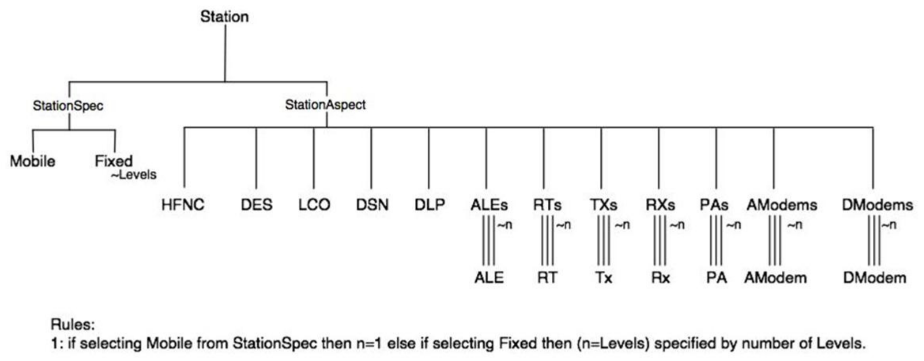


Figure 8.27: System entity structure for SCOPE command system showing the fixed and mobile (aircraft) stations

To study the effect of changes/upgrades introduced to the existing SCOPE command system we built the Experimental frame, based on DEVS principles for our modular DEVS-NETSIM simulation model, named GENETSCOPE [Gen06]. Figure 8.28 shows the block architecture of the simulation model. The right-hand box is the system phenomenon that contains the Automatic Link Establishment (ALE), STANAG 5066 protocols used for establishing links and exchanging data messages between mobile stations and fixed stations. The left-hand box is the experimental frame that generates various scenarios and parameters under study. The scenarios and parameters are fed into the model and performance characteristics are obtained from it, which are then visualized and analyzed in real time as per the extended MVC architecture described in Chapter 3.

[5] A methodology using intermediate XML processing to automate much of the process of "componentizing" legacy simulation code will be reported soon.

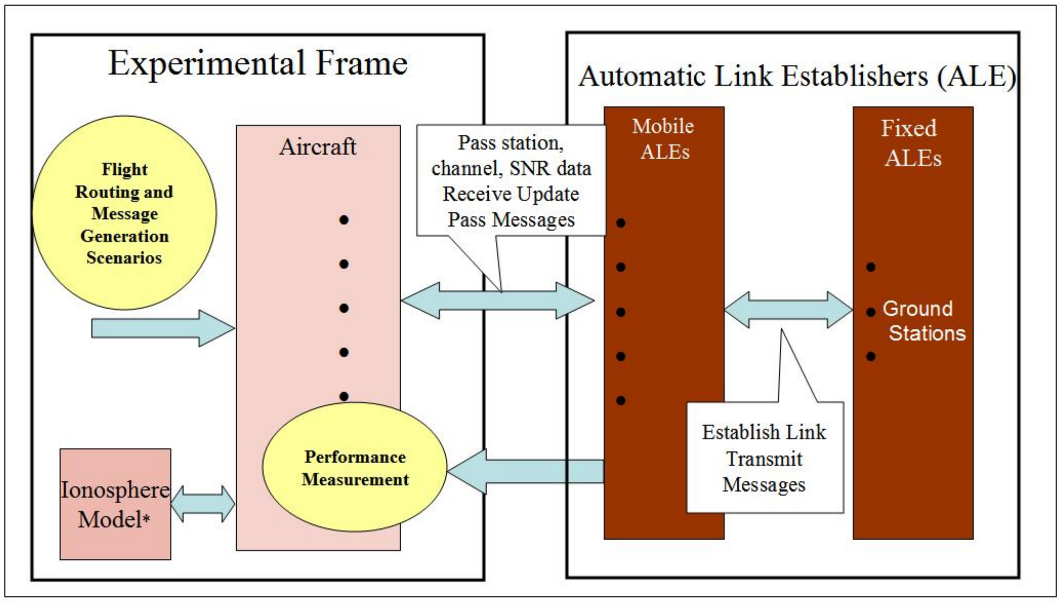

Experimental Frame
Flight Routing and Message Generation Scenarios
Aircraft
Performance Measurement
Ionosphere Model*
Pass station, channel, SNR data Receive Update Pass Messages
Automatic Link Establishers (ALE)
Mobile ALEs
Fixed ALEs
Ground Stations
Establish Link Transmit Messages

Figure 8.28: GENETSCOPE simulation architecture for SCOPE command

8.4.1 SCOPE Command and DoDAF

Certainly, a system like SCOPE command qualifies to be represented as a DoDAF specification. Though not provided in this paper, all three views, viz., Operational, System, and Technical, can be developed. The documents are fairly easy to construct as the system is not in the design phase but is a live system with working standards and people managing the system for as long as twenty years. The physics of the HF communication is still the same, and the radio equipment has set standards that have not been revised that often. What is new in the system is the incorporation of new standards, for example, the STANAG 5066 data-exchange protocol that modulates the modem rates and reliable data delivery across the HF messaging system. This is being added to provide the capability to send e-mail messages through the HF system. The other major thing that has changed is the increased intensity of traffic, demanding upgrades to the existing system. For illustration purposes, suppose that we had the DoDAF description of SCOPE as well as all the details on how the system would be constructed and its functionality implemented. Remaining solely within the DoDAF, there still would not be any means to analyze or experiment with the projected system. As stressed earlier, the DoDAF does not provide for any M&S capability to support the system design process. It only provides a means to build a system on the presumption that analysis has already been done, a “design” is available, and the system is ready to be deployed. The purpose of the DoDAF in this case is nothing more than a documenting procedure.

The methodology presented in this paper takes the DoDAF as a front-end documentation procedure that aids M&S and design objectives. With respect to Figure 4, the central theme of the paper, we present sample OV-8 and OV-9 documents to illustrate how the experimental frame is developed from the DoDAF terminology.

Although the current DoDAF views are insufficient to provide the M&S for the purpose of enhancing and recommending upgrades to the existing SCOPE system, the DEVS approach readily provides the needed tools. Going back to the basic DEVS M&S components (see Figure 3.1), the legacy SCOPE simulation model was transformed by the base high resolution model. The Experimental frame is constructed over this existing system along with various other additions that would control and direct the possible upgrades. This component is responsible to provide environmental conditions, workload generation, performance analysis, system evolution and control, and achievement of steady state. The other advantage of this separation is the construction of a DEVS lumped model in which various details of the base model are abstracted and lumped together. Whereas the base model is oriented to technical components, the lumped model directly addresses system level issues and supports faster simulation runs to answer these questions. As always, the question arises as to how close these results match with the detailed model. The lumped model is preferred if it is able to perform to the same level of accuracy and helps answer the questions raised by the SCOPE command designers. The comparison of a lumped model with a base model is only possible if the underlying M&S formalism supports modular construction of the three components, viz., model, simulator, and Experimental frame [11]. Figure 8.29 summarizes the general idea.

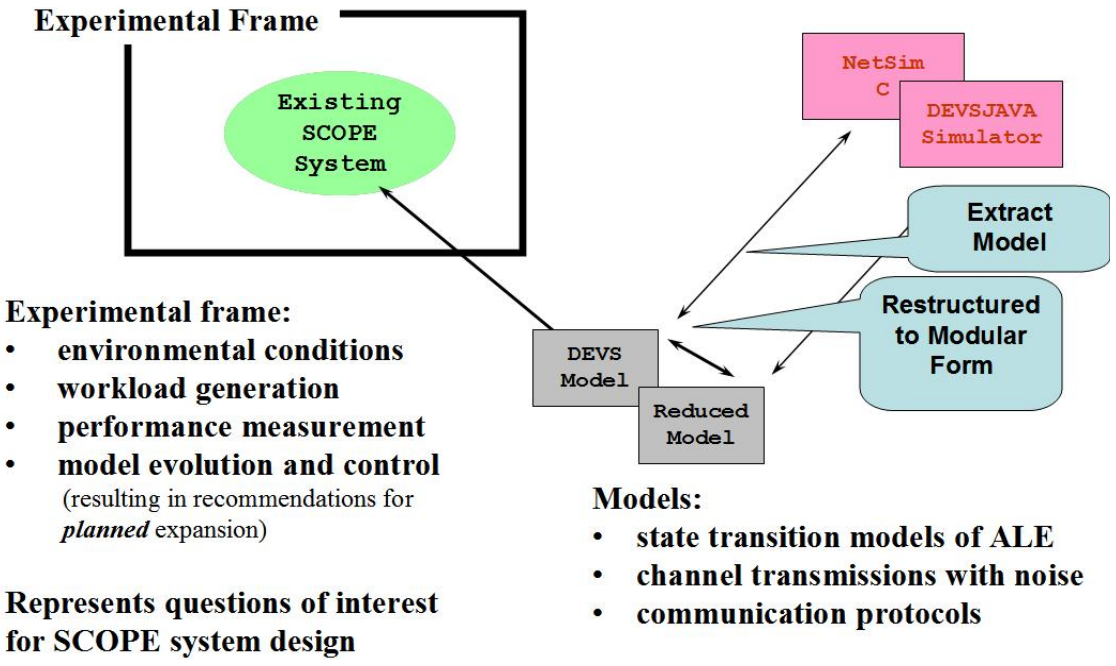

Experimental Frame
Existing SCOPE System
NetSim C
DEVSJAVA Simulator
Extract Model
Restructured to Modular Form
DEVS Model
Reduced Model
Experimental frame:
• environmental conditions
• workload generation
• performance measurement
• model evolution and control (resulting in recommendations for *planned* expansion)
Represents questions of interest for SCOPE system design
Models:
• state transition models of ALE
• channel transmissions with noise
• communication protocols

Figure 8.29: DEVS M&S and the existing SCOPE command system

Sample OV-8 and OV-9 Documents

Let's consider two activities out of many activities that are a part of any HF radio communication, i.e., *sounding* and *listening.* Sounding is defined as the process by which different stations (refer to Figure 8.27) periodically send broadcast messages at different frequencies so that other stations know who else is available on the HF radio sky. Listening is defined as the process by which these stations identify and hear RF tones and go through a demodulation process to decode and decipher the incoming transmission.

S. No.	**Activity**	**Sub-activity**	**Internal Activity**	**ID**
1	**Sounding**			**A1**
2.		Prepare Call		A1.1
3.		Send Call		A1.2
4.		Send Transmission		A1.3
5.	**Listening**			**A2**
6.		Receive Transmission		A2.1
7.		Evaluate Signal		A2.2
8.		Decode Signal		A2.3
9.		Report Message		A2.4

Table 8.8: Activity 4ID mapping for OV-8 and OV-9

Table 8.8 describes the initial process that is done to populate the OV-8 document. It assigns various IDs to different Activities and sub-activities that are then used as reference tokens and automation processes, as described in [Mit06a]. Figure 8.30 depicts the OV-5 for activity *sounding*. Activity *listening* will have a similar Operational View depiction. Table 8.9 presents a sample OV-8 document with refined structure (see Table 3) showing the significant parameter set for *sounding* and *listening* activities. It should be well stressed here that documention and aggregation of this information with the corresponding activity helps find faults in testing the “feasibility” of the system [Mit06a] when M&S is employed.

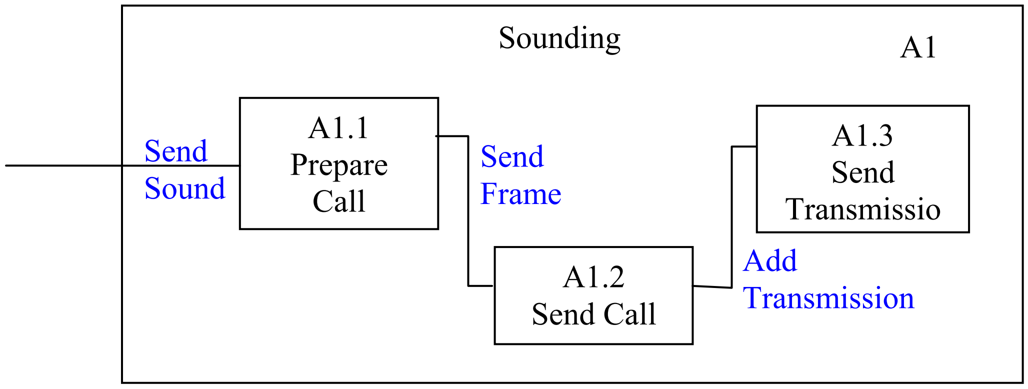


Figure 8.30: OV-5 for activity sounding

S. No.	**Activity ID Compo nent**	**Signify-cant Para-meter**	**Connec tion ID**	**Source Activity**	**Input Interface Name**	**Message Descript-ion**	**Contai ner Op. Node**	**Source Docum ent/ Diagra m**
1	**A1**						Station	
2.	A1.1	**Sounding-interval, duration**	CA1.1	Ax (hypoth etical)	inSta	Send sound	Station	Figure 5.29/ OV-5
3.	A1.2	Message size, frame count	CA1.2	A1.1	inAle	Send frame (s)	Station	Figure 5.29/ OV-5
4.	A1.3	**Duration**	CA1.3	A1.2	inRt	Add transmission	Station	Figure 5.29/ OV-5
5.	**A2**							
6.	A2.1	Duration	CA2.1	Ay (hypoth etical)	inRt	Receive transmission	Station	Figure x
7.	A2.2	Station to-station SNR	CA2.2	A2.1	inAle	SNR	Station	Figure x
8.	A2.3	Received frames, valid frames, duration	CA2.3	A2.2	inAle	Incoming sound	Station	Figure x
9.	A2.4	None	CA2.4	A2.3	inHfnc	Heard station X	Station	Figure x

Table 8.9: Sample OV-8 document

Having constructed the OV-8 document, let us construct the OV-9 documents according to the proposed structure in [Mit06a]. Table 8.10 presents the components that lie within the Operational Node *station* and their assigned IDs for automation purposes. For more details, refer to [Mit06a]. It is worth stressing here that this information comes readily from the SES of the existing SCOPE command system, as shown in Figure 8.27. The inner components within the *station* Operational Node are clearly defined in Figure 8.27.

S. No.	**Operational Node**	**Inner Component Entities**	**Component Name**	**Associated Models Added to Repository**	**Hierarchical Parent/Container**	**DEVS Model Type**
1.	**O1**	OCE1	Station	ME1	-	Digraph
2.		OCE1.1	HFNC	ME1.1	ME1	Atomic
3.		OCE1.2	ALE	ME1.2	ME1	Atomic
4		OCE1.3	RT	ME1.3	ME1	Atomic
5		OCE1.4	TX	ME1.4	ME1	Atomic
6.		OCE1.5	RX	ME1.5	ME1	Atomic
7.		OCE1.6	PA	ME1.6	ME1	Atomic

Table 8.10: Inner components within operational nodes and their mapping with "standardized" DEVS models

S. No.	**Operational Node**	**Inner Component Entities**	**Component Name**	**Activity Component**	**Activity Component Name**	**Interface Description**
1.	**O1**	OCE1	TCT			
		OCE1.1	HFNC	Ax	Time To Sound	tts
				A2.4	Report Message	repMsg
		OCE1.2	ALE	A1.1	Prepare Call	prepCall
				A1.2	Send Call	sendCall
				A2.2	Evaluate Signal	evalSig
				A2.3	Decode Signal	decSig
		OCE1.3	RT	A1.3	Send Transmission	sendTransm
				A2.1	Receive Transmission	recvTransm
		OCE1.4	TX	A1.3	Send Transmission	putTransm

		OCE1.5	RX	A2.1	Receive Transmission	getTransm
		OCE1.6	PA	None	None	None

Table 8.11: Sample OV-9 Document

Hence, during the creation of OV-8 and OV-9 we have populated the model repository with Activity models (MA6.1–MA6.18) and Operational node's inner components models (ME1, ME1.1–ME1.6) and have created an interface between these two aspects of DoDAF design. In the subsequent sections, we shall see how these enhanced OV-8 and OV-9 documents prove to be advantageous in defining the DEVS Experimental frame parameters and hierarchical GUI developments or code development of the simulation model.

8.4.2 SCOPE Architecture Implementation Using Enhanced MVC

Figure 8.31 shows the simulation architecture for GENETSCOPE [Gen06] using the concepts laid out in the paper. With reference to Figure 5.27, the ionosphere model used in the architecture is ICEPAC data. It is worth stressing that the initial NETSIM model written in C language has this database tightly coupled with the model. In our present implementation, we made it modular so that it can be replaced by any other database that could provide the channel propagation values through the ionosphere, e.g., VOACAP. In the current implementation, there is no ICEPAC database included but the complete ICEPAC software that is executed at run time. This is one of the biggest advantages in separating ICEPAC from the model itself. The ICEPAC software is configured through the Experimental frame parameters and is made available for real-time execution as an independent thread for different stations that are active in the running DEVS model. The real-time execution of ICEPAC software involves creation of a dynamic ICEPAC configuration file that contains information about the two stations, their geographical locations in latitude and longitude, the Sun Spot Number (SSN), and the time of year, month, and day. This implementation allows us to get the ionospheric SNR values for any location at any time of the year (for SSN) unlike the earlier implementation (NETSIM-SC) where we were limited to only a handful SSN values (10, 70, 100, and 130) with locations specified in five-degree increments. This has the added benefit of using the exact location of any mobile station rather than using projections within the implemented grid as in the earlier NETSIM-SC. The DEVS layer comprises both models as well as the DEVS simulation environment. The Experimental frame layer also contains the controls required to modify/update the model as well as a simulator as per enhanced MVC. The simulation visualization is modular in construction and reflects the updates in the Experimental frame layer and the DEVS layer. See Figure 8.31.

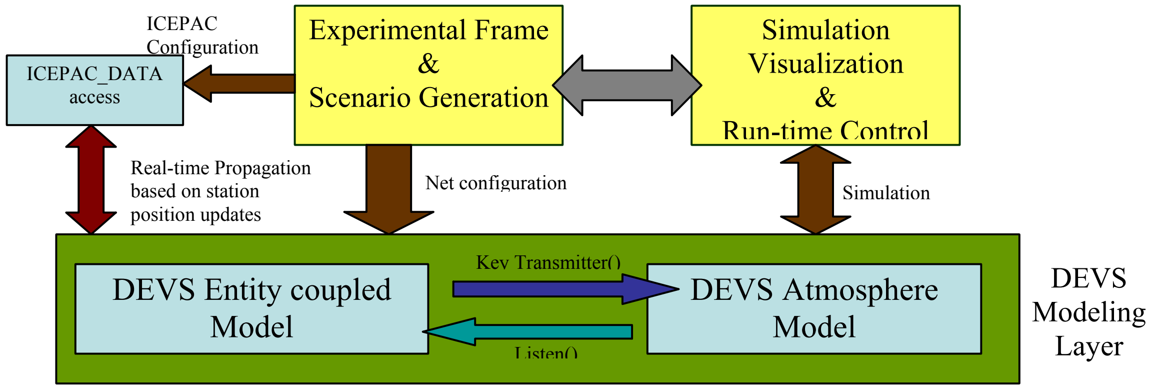
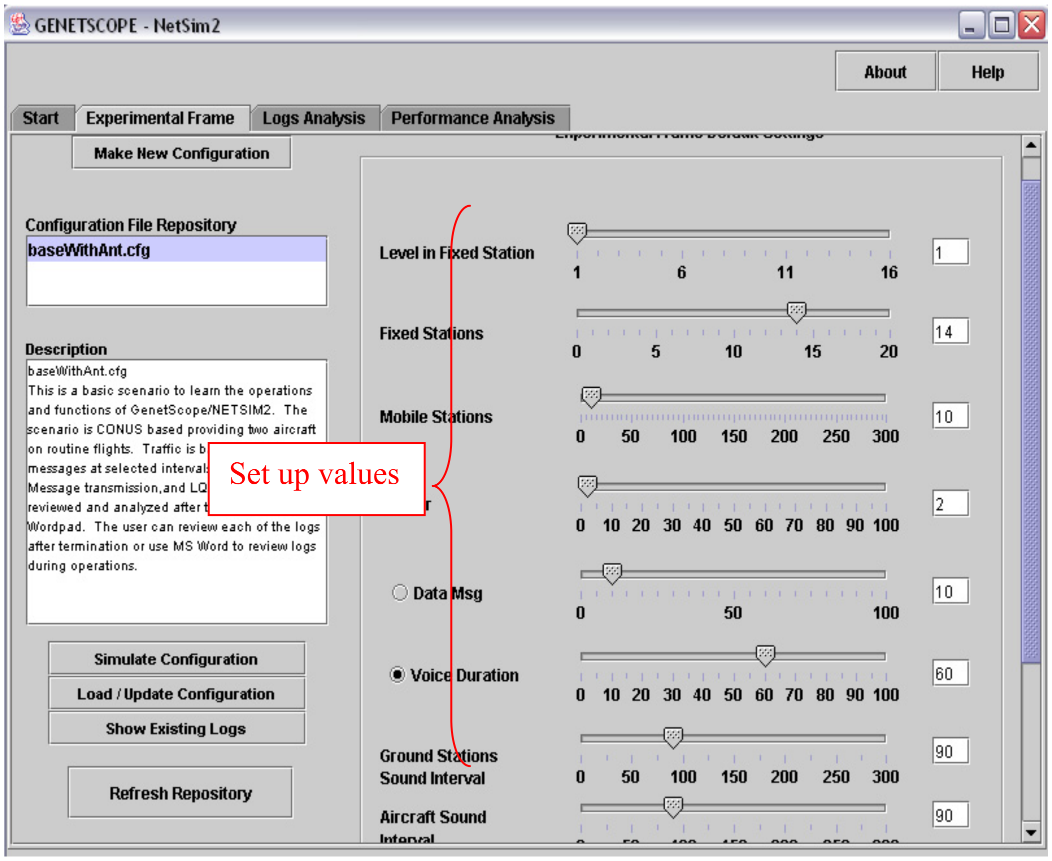

ICEPAC Configuration
ICEPAC_DATA access
Experimental Frame & Scenario Generation
Simulation Visualization & Run-time Control
Real-time Propagation based on station position updates
Net configuration
Simulation
DEVS Entity coupled Model
Key Transmitter()
Listen()
DEVS Atmosphere Model
DEVS Modeling Layer

Figure 8.31: Simulation architecture for the SCOPE command network

The above architecture is shown below in various screen shots taken from the developed GENETSCOPE (beta version). Figure 8.32 shows the Experimental frame and various parameters (along with their default values) used in scenario configuration. The parameters shown in bold below are the parameters that have been identified as *significant* parameters in OV-8 (see Table 8.9, in shaded cells). Similarly, other parameters too come from an elaborate OV-8 document of the SCOPE command. These significant parameters find their way in various configurable parameters all through the model configuration settings as shown in Figures 8.32-34, and the simulation model finds its design through the SES (see Figure 8.27) or the corresponding OV-9 document (see Table 8.11). The total parameter set is comprised of:

1. Number of fixed stations,
2. Number of levels inside a fixed station,
3. Number of mobile stations (aircrafts),
4. Messages per hour,
5. **Data message size,**
6. **Voice call duration,**
7. **Ground stations sounding interval, and**
8. SNR threshold for a received signal.



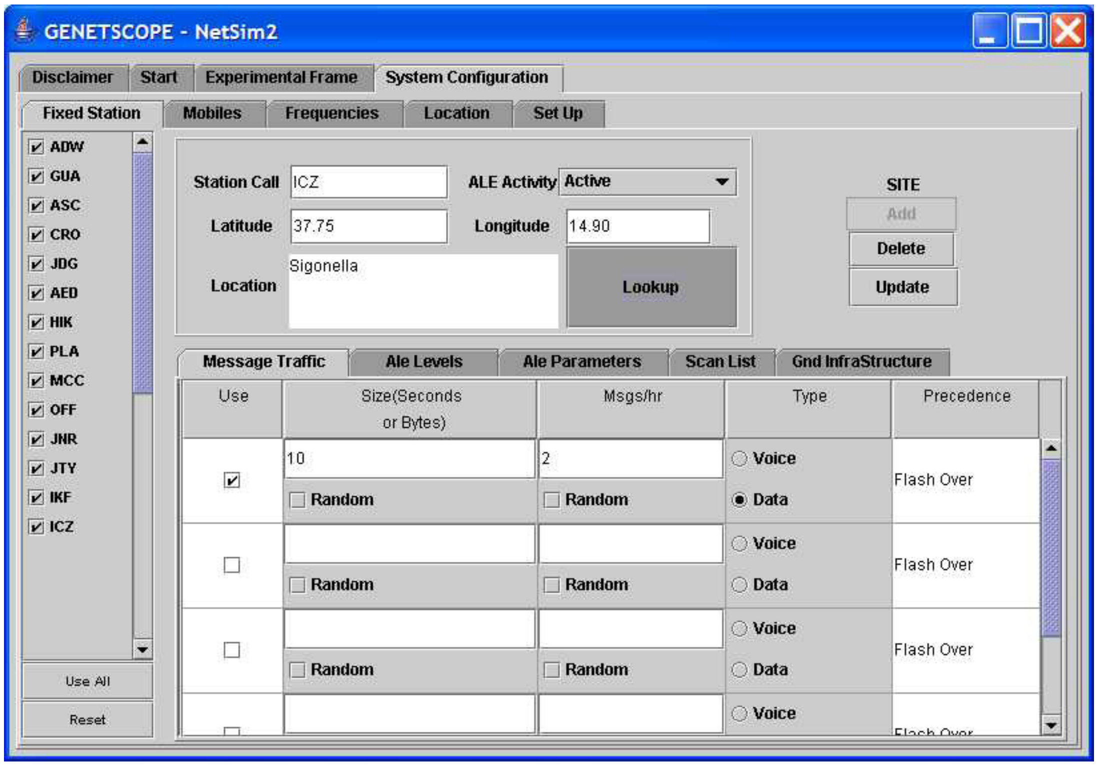
Configuration File Repository
baseWithAnt.cfg

Description
baseWithAnt.cfg
This is a basic scenario to learn the operations and functions of GenetScope/NETSIM2. The scenario is CONUS based providing two aircraft on routine flights. Traffic is b messages at selected interval Message transmission,and LQ reviewed and analyzed after t Wordpad. The user can review each of the logs after termination or use MS Word to review logs during operations.

Level in Fixed Station	1
Fixed Stations	14
Mobile Stations	10
r	2
( ) Data Msg	10
(●) Voice Duration	60
Ground Stations Sound Interval	90
Aircraft Sound Interval	90

Figure 8.32: Experimental frame for GENETSCOPE

Once the experimental frame parameters are configured, these parameters are channeled down to the individual components. The top-level design parameters then bound the other internal component parametric settings. For example, Figure 8.33shows a typical configuration of the ground station Sigonella. The left column in Figure 8.33 shows all the fourteen ground stations, and the individual details about each station can be seen by pressing the *Lookup* button. Figure 8.33 also shows the message traffic that is transmitted by this station. Notice that the Experimental frame settings are shown as the traffic stream originated from this station. Similarly, a mobile station configuration panel is shown in Figure 8.34. The user can select any specific mobile aircrafts bounded by the number of mobile stations specified in the Experimental frame. The next figure, 8.35, basically lets the user enter call-signs to these mobile stations and invites the user to enter aircraft-specific details like message traffic, flight path (see Figure 8.36), radio parameters, and channel frequencies being used. Other internal details of station configuration can be seen in the GENETSCOPE software user's manual [Gen06]. The purpose of showing GUI snapshots in Figures 8.33-36 is to illustrate how top-down design parameters (from OV-8) can be taken down to the component level (through both OV-8 and OV-9). The other important aspect of this process is that during simulation run-time, if the Experimental frame parameters are changed to study any particular parameter, that change is channeled across the whole system model configuration using "interrupts," thereby exploiting the discrete event simulation methodology. The update of any Experimental frame parameter is taken by the simulation model as an "external" event.



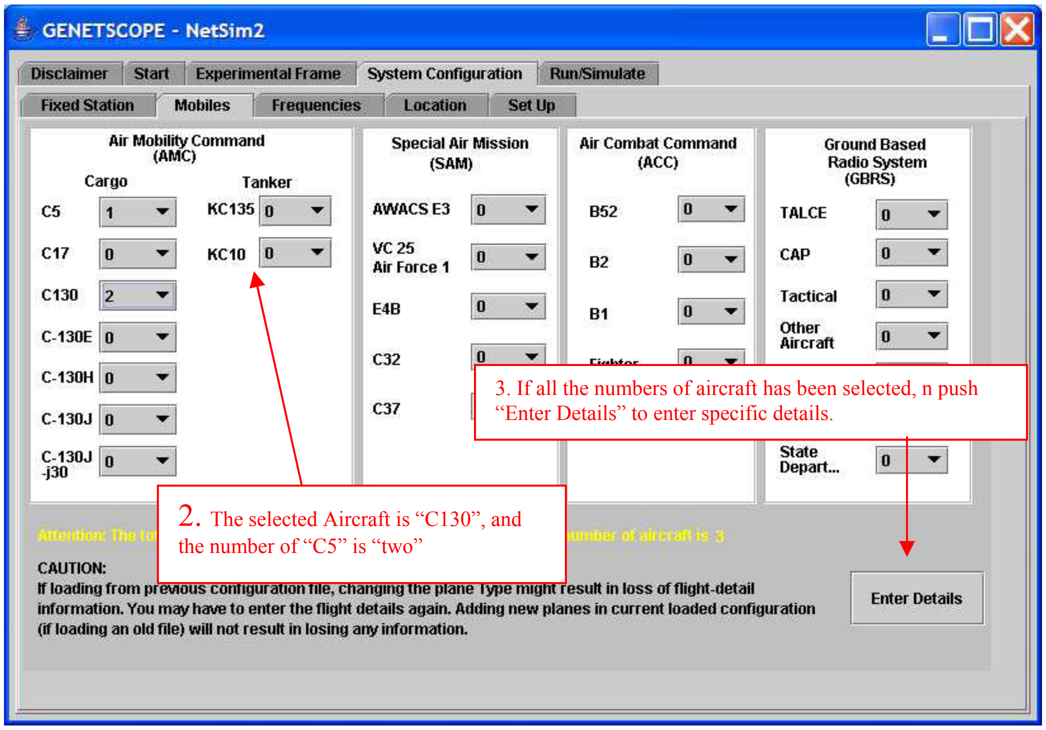
☑ ADW
☑ GUA
☑ ASC
☑ CRO
☑ JDG
☑ AED
☑ HIK
☑ PLA
☑ MCC
☑ OFF
☑ JNR
☑ JTY
☑ IKF
☑ ICZ

Station Call: ICZ
ALE Activity: Active
Latitude: 37.75
Longitude: 14.90
Location: Sigonella

Use	Size(Seconds or Bytes)	Msgs/hr	Type	Precedence
☑	10 ☐ Random	2 ☐ Random	○ Voice ● Data	Flash Over
☐	☐ Random	☐ Random	○ Voice ○ Data	Flash Over
☐	☐ Random	☐ Random	○ Voice ○ Data	Flash Over
☐			○ Voice	Flash Over

Figure 8.33: Ground station configuration screen for Naval Air Station Sigonella

The last piece of information being fed through the Experimental frame is the ICEPAC setting, based on the Sun Spot Number (SSN). Once the system model is configured through the Experimental frame settings, the user is directed toward the simulation setup. Figure 8.37 shows the final setup screen after which the user then moves on to the run-time simulation screen (see Figure 8.38) to execute the simulation. When the user clicks the *Write Files* button in Figure 8.37, it results in writing up of the detailed configuration file for repository purposes.



Air Mobility Command (AMC) Cargo		Air Mobility Command (AMC) Tanker	
C5	1	KC135	0
C17	0	KC10	0
C130	2		
C-130E	0		
C-130H	0		
C-130J	0		
C-130J -j30	0		

Special Air Mission (SAM)	
AWACS E3	0
VC 25 Air Force 1	0
E4B	0
C32	0
C37	

Air Combat Command (ACC)	
B52	0
B2	0
B1	0
Fighter	0

Ground Based Radio System (GBRS)	
TALCE	0
CAP	0
Tactical	0
Other Aircraft	0
State Depart...	0

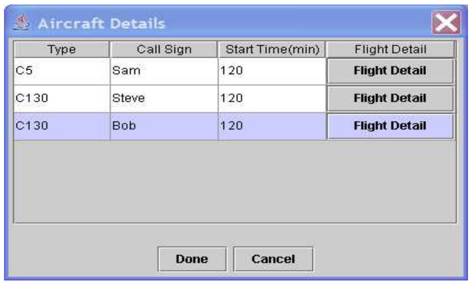
Attention: The to[illegible] umber of aircraft is 3

CAUTION:
If loading from previous configuration file, changing the plane Type might result in loss of flight-detail information. You may have to enter the flight details again. Adding new planes in current loaded configuration (if loading an old file) will not result in losing any information.

2. The selected Aircraft is "C130", and the number of "C5" is "two"

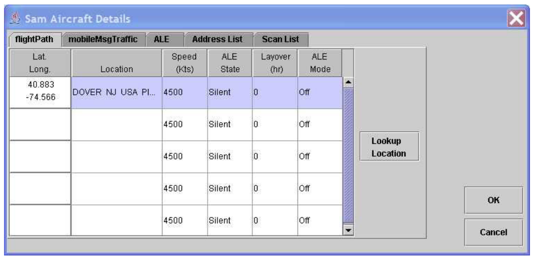
3. If all the numbers of aircraft has been selected, n push "Enter Details" to enter specific details.

Figure 8.34: Mobile station configuration screen where the total count is bounded by the Experimental frame



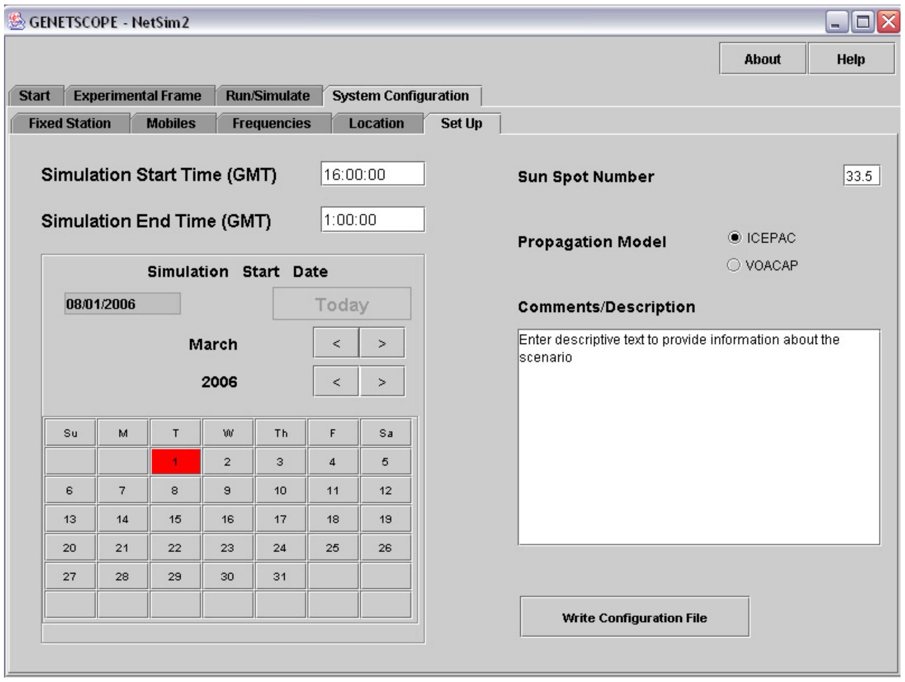
Type	Call Sign	Start Time(min)	Flight Detail
C5	Sam	120	Flight Detail
C130	Steve	120	Flight Detail
C130	Bob	120	Flight Detail

Figure 8.35: Callsign entry for a mobile station



Lat. Long.	Location	Speed (Kts)	ALE State	Layover (hr)	ALE Mode
40.883 -74.566	DOVER NJ USA PI...	4500	Silent	0	Off
		4500	Silent	0	Off
		4500	Silent	0	Off
		4500	Silent	0	Off
		4500	Silent	0	Off

Figure 8.36: Flight path of mobile aircraft and other details



Simulation Start Time (GMT) 16:00:00

Simulation End Time (GMT) 1:00:00

Simulation Start Date

08/01/2006

March

2006

Su	M	T	W	Th	F	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Sun Spot Number 33.5

Propagation Model [✓ ICEPAC] VOACAP

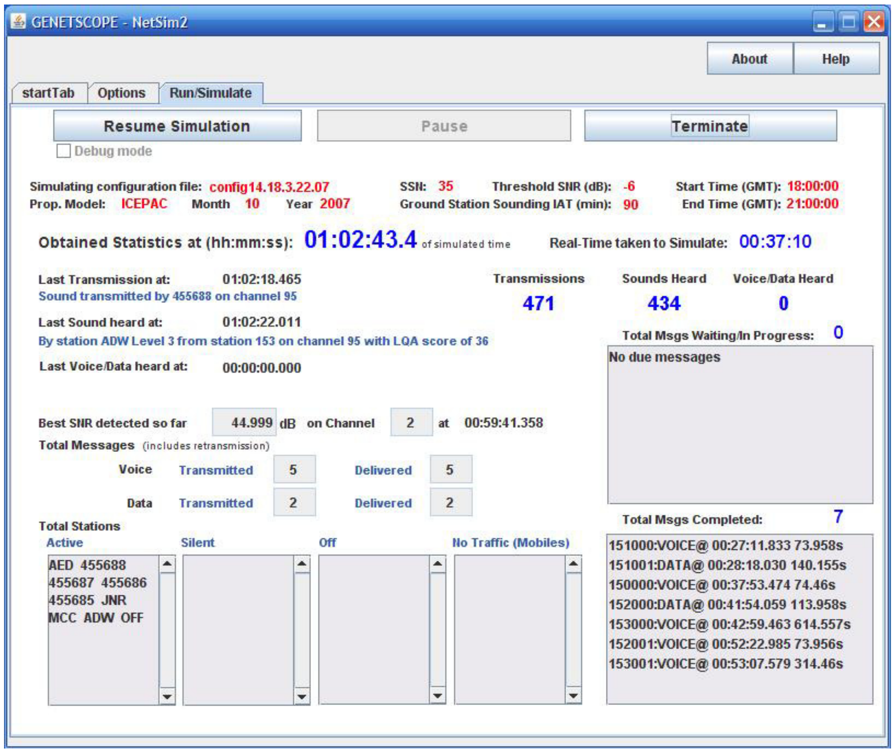
Comments/Description

Enter descriptive text to provide information about the scenario

Figure 8.37: Experimental frame and ICEPAC data configuration through selection of SSN

Figure 8.38 shows the simulation clock as it happens in real time and the obtained statistics. The above snapshots complete the architectural components specified in Figure 8.31. Figure 8.38 has the functionalities that are described earlier in the paper: e.g., run-time configuration updating and simulation control. It has four buttons at the top of the screen, viz.:

1. *Run Abstract Model* (using lumped parameters),
2. *Run Detailed Model* (using detailed parametric settings),
3. *Pause* (to interrupt the simulation),
4. *Terminate* (to end the simulation).



Simulating configuration file: config14.18.3.22.07
Prop. Model: ICEPAC Month 10 Year 2007
SSN: 35 Threshold SNR (dB): -6
Ground Station Sounding IAT (min): 90
Start Time (GMT): 18:00:00
End Time (GMT): 21:00:00

Obtained Statistics at (hh:mm:ss): 01:02:43.4 of simulated time Real-Time taken to Simulate: 00:37:10

Last Transmission at: 01:02:18.465
Sound transmitted by 455688 on channel 95
Last Sound heard at: 01:02:22.011
By station ADW Level 3 from station 153 on channel 95 with LQA score of 36
Last Voice/Data heard at: 00:00:00.000

Transmissions	Sounds Heard	Voice/Data Heard
471	434	0

Total Msgs Waiting/In Progress: 0
No due messages

Best SNR detected so far 44.999 dB on Channel 2 at 00:59:41.358
Total Messages (includes retransmission)

Voice	Transmitted	5	Delivered	5
Data	Transmitted	2	Delivered	2

Total Stations

Active	Silent	Off	No Traffic (Mobiles)
AED 455688 455687 455686 455685 JNR MCC ADW OFF			

Total Msgs Completed: 7

```
151000:VOICE@ 00:27:11.833 73.958s
151001:DATA@ 00:28:18.030 140.155s
150000:VOICE@ 00:37:53.474 74.46s
152000:DATA@ 00:41:54.059 113.958s
153000:VOICE@ 00:42:59.463 614.557s
152001:VOICE@ 00:52:22.985 73.956s
153001:VOICE@ 00:53:07.579 314.46s
```

Figure 8.38: Run-time simulation visualization screen for rapid feedback

The *Pause* button is of special interest here, as the user can interrupt the running simulation (manual reactive control described in Chapter 4) and change the Experimental frame or system configuration settings while the simulation is in action. Once the parameters have been updated, the user can resume the simulation and can see the impact of that update on the above “active” simulation visualization screen. One such example may be the two obtained values of total transmissions and total sounds heard. If the number of sounds heard is not up to the mark (with respect to a validated real-world scenario), the user may change sound-interval time or any other parameter that would impact this number, or may conclude that the model is not “performing” correctly. The rapid impact of any such parameter can be studied by pausing the simulation and changing it and then observing the effects in the simulation pane.

The DEVS layer in Figure 8.31 is implemented in the following manner. The simulation engine running behind uses the following code.

```
NetsimSC net = new NetsimSC(createdConfigFile, debugOption);
tCoord = new TunableCoordinator(net);
tCoord.initialize();
tCoord.setTimeScale(0.0001);
tCoord.simulate(Integer.MAX_VALUE);
```

The model configuration is written into a configuration file that is used to create the DEVS digraph model, with automated coupling using the system SES shown in Figure 8.27. The DEVS model is then passed on to the TunableCoordinator derived from DEVS RTcoordinator class. The TunableCoordinator is initialized and is then directed to simulate for a maximum number of iterations, which means that simulation will proceed indefinitely (in logical sense). The *Pause* button executes the following line.

```
tCoord.interrupt();
```

After the simulation is paused and updates are made, the simulation is restarted by simply calling the coordinator to “simulate.”

```
tCoord.simulate(Integer.MAX_VALUE);
```

The simulation core functionality provided by the DEVS simulation protocol facilitates interrupting the coordinator and makes real-time parametric and component structures at run time as described in sections 6 and 7 earlier.

Figure 8.38 contains a very limited set of aggregated information. However, run-time graphs and projections can be very well aligned with this visualization to see patterns and the direction in which the simulation is proceeding. Logs are generated for each simulation run. This visualization pane shows the important information of the Experimental frame (in red) and the run-time information from the system model (in blue), which, needless to say, is according to the enhanced MVC (through the development of appropriate interfaces between these layers). The View layer (see Figure 3.2) in the current example shows only the model and the Experimental frame control visualization. The Experimental frame control is *controller B* in Figure 3.2, i.e., parameters that "control" the model. The lowest layer, i.e., *controller A* in the enhanced MVC process, is not the focus of the GENETSCOPE project and consequently not illustrated here. Its implementation is illustrated in the work [Nut05].

8.4.3 Implications of the Example Above and NR-KPP

Having laid out the framework to conduct and design the experiments, the next item on the agenda is to identify the *measures of effectiveness* (MoEs) that eventually will be considered in making recommendations for any update or modification needed in the current SCOPE command infrastructure. Since the SCOPE command is a deployed system, we were given various statistical reports by JITC [JITC] in order to determine these MoEs. The point of this exercise is to provide sufficient analysis through simulation of the modeled system so that the impact of any particular infrastructural change intended in the system can be observed on these MoEs. Some of the MoEs that were identified are as follows:

1. Longest time taken by any e-mail on HF network,
2. Number of e-mails sent and number of e-mails actually delivered,
3. Average message transmission time at any station per hour,
4. Messages attempted versus messages received per hour,
5. Bandwidth usage at Central Network Command Station (CNCS[6]),
6. Number of planes in "good" signal to noise ratio (SNR) range per hour.

The parameters that are to be set in order to recommend any upgrades in the current infrastructure can be listed as follows:

1. Average number of daily flights,
2. Minimum number of messages attempted by any station,
3. Number of fixed stations participating in any mission scenario,
4. Number of active levels within a fixed station,
5. Minimum and maximum message size in KB,
6. Minimum and maximum duration of a phone call (VOICE message),
7. Minimum data rate by any ALE radio-modem.

As can been seen clearly, there is not a one-to-one mapping between MoEs and experimentation parameters. The MoEs tell us about the effectiveness of any mission that would be executed. They are holistic measures that tell about the fitness, capacities, and

[6] CNCS is the gateway for any land-based network (SIPRNET or NIPRNET) to be connected to the SCOPE command HF network. All e-mails are routed through CNCS.

limitations of the system. M&S is the preferred means for assessing the impact of parameters on MoEs, with the goal of determining the most significant parameters. A simulation execution environment can help this investigation through a rapid feedback cycle where the analyst can change parameter values on the fly and quickly assess their impact on holistic measures. These MoEs impact evaluations very well and become part of the result set as mentioned in Chapter 3, while the parameters identified become part of the Experimental frame layer as shown in Figure 8.31.

Similarly, for any DoDAF architecture, the MoEs are also specialized for that particular architecture. Considering the breadth of the SCOPE command system, some of the MoEs mentioned above also apply to any net-centric architecture. Within the DoD, JITC has the sole responsibility of certifying the Information Technology (IT) and National Security Systems (NSS) for interoperability purposes [Buc04]. The major T&E problem identified today by JITC is how to verify that a solution provided by any architecture is data integrated and net centric in operation. The traditional T&E approaches are optimized to verify performance and effectiveness of point solutions, but new criteria are needed to reflect the realities of systems operating within networked systems. Such criteria are just beginning to emerge and are not yet matured for immediate and widespread use of T&E [Buc04].

The NR-KPP assesses net-readiness information assurance (IA) requirements, and end-to-end operational effectiveness of that exchange with respect to the COIs mentioned earlier. Description of Key Interface Profile (KIP) with relation to this scenario is beyond the scope of this paper. The major object underlying NR-KPPs is to identify verifiable performance parameters and associated metrics required to evaluate timely, accurate, and complete exchange and use of information to satisfy the information needs for a given capability [Buc04].

CHAPTER 9: DISCUSSION

This chapter discusses the DUNIP process with current state of the art in model-based engineering processes. Two paradigms have been chosen: MDA and SCR. MDA or Model-Driven Architecture is philosophy as put forward by Object Modeling Group (OMG) that comprises of many standards like UML, XMI, Meta-Object Facility (MOF) and others. SCR is the Software Cost Reduction methodology developed at Naval Research Laboratory to develop models based on requirements specified in tabular format.

9.1 MDA and DUNIP

DUNIP is built on the paradigm of Model-Based Engineering, or Model-Driven Architecture (MDA). However, the scope of DUNIP goes beyond the MDA objectives. Potential concerns with the current MDA state of art include:

- MDA approach is underpinned by a variety of technical standards, some of which are yet to be specified (e.g. executable UML)
- Tools developed my many vendors are not interoperable
- MDA approach is considered too-idealistic lacking iterative nature of Software Engineering process
- MDA practice requires skilled practitioners and design requires engineering discipline not commonly available to code developers.

Further, MDA does not have any underlying Systems theory and groups like INCOSE[7] are working with OMG to adapt UML to systems engineering. Testing is included only as an extension of UML, known as executable UML [Mel02], for which there is no current standard. Consequently, there is no testing framework that binds executable UML and simulation-based testing.

Despite these shortcomings, MDA has been adopted by Joint Single Integrated Air Picture (SIAP) Systems Engineering Organization (JSSEO) and various recommendations have come forth to enhance the MDA process. JSSEO is applying MDA approach toward development of aerospace Command and Control (C2) capabilities, for which a single integrated air picture is foundational. The data-driven nature of C2 System of Systems (SoS) means that powerful MDA concepts adapt well to collaborative SoS challenges.

Current DoD enterprise-level approaches for managing SoS interoperability, like the Net Centric Operations and Warfare Reference Model (NCOW/RM), DoD Architecture Framework (DoDAF) and the Joint Technical Architecture (JTA), simply do not have the technical strength to deal with the extremely complex engineering challenges [Jac04]. We proposed enhanced DoDAF [Mit06a] to provide DEVS-based Model engineering. MDA as implemented by industry and adapted by JSSEO, does have the requisite technical power, but requires innovative engineering practices.

[7] International Council on Systems Engineering

Realizing the importance of MDA concepts and the executable profile of UML, the basic objective of which is to simulate the model, JSSEO is indirectly looking at the Modeling & Simulation domain as applicable to SoS engineering. The following table brings out the shortcomings of MDA in its current state and the capabilities provided by DEVS technology and in turn, DUNIP process.

Desired M&S Capability	**MDA**	**DUNIP**
Need for executable architectures using M&S	Yes, although not a standard yet	Yes, underlying DEVS theory
Applicable to GIG SOA	Not reported yet	Yes
Interoperability and cross-platform M&S using GIG/SOA	--	Yes, DEVSML and SOADEVS provides cross-platform M&S using Simulation Web Services
Automated test generation and deployment in distributed simulation	--	Yes, based on formal Systems theory and test-models autogeneration at various levels of System specifications
Test artifact continuity and traceability through phases of system development	To some extent, model becomes the application itself	Yes
Real time observation and control of test environment	--	Dynamic Model Reconfiguration and run-time simulation control integral to DEVS M&S. Enhanced MVC framework is designed to provide this capability

Table 9.1: Comparison of MDA and DUNIP

MDA as applied to Integration of Process-Driven SOA Models

In an independent study [Zdu02], Model Driven Software Development (MDSD) was applied to the integration of process-driven SOA models. UML2 was used as the basis towards integration. Their approach is based on the notion of domain-specific languages

(DSL) for modeling various types of models. Once DSL has been identified, its meta-model is created that represents this particular modeling domain. Meta-models are defined in terms of meta-meta-model. In UML, this is the meta object facility (MOF). They created a meta-meta-model that would define both the UML2 meta-model and their selected DSL extensions. The whole objective is to find a common ground and a way to express the relationship between a meta-model and the implementation code. This kind of capability where a single meta-meta-model can be used to integrate two different DSLs towards a common model allowing specific constraints of each meta-model is very much needed in SOA domain as multiple tools and standards exist preventing such integration. To integrate two models with different DSLs, the models are first decomposed at the meta-model level, required information extracted and supplemented (on the basis of meta-meta-model), which results in an integrated model.

In our DUNIP process, such collaboration comes naturally due to the proposed DEVS atomic and coupled Document Type Definitions (DTDs) that specify any DEVS model in any domain specific language implementations. The underlying DEVS Modeling Language (DEVSML) meta-model that defines these atomic and coupled DTDs is used for validating any DEVS model. The current DEVSML implementation has successfully integrated two DSL implementations (GenDEVS-ACIMS and xDEVS-Spain) on common DEVSML atomic and coupled DTDs.

9.2 DUNIP and SCR

Software Cost Reduction (SCR) method allows development of formal requirements using a tabular notation. The SCR toolset includes an editor for building the specifications, a consistency checker for testing the specifications for consistency with formal requirements model, a simulator for symbolically executing the specifications and a verifier for checking that the specifications satisfy selected applications properties [Heit95]. SCR has been used to define requirements for embedded systems as well as software systems. SCR is more exhaustive and complete in terms of model checking and consistency checking. It is at a higher order of resolution where state variables can be a part of the specification definition.

In DUNIP, although it is based on DEVS, the state-variables are not considered in the automated DEVS model generation as described in Chapter 4. Our current work falls in the category of a subset of DEVS specifications, where only message passing between the components is considered. The motivation of this research effort stems from the need of absence of an M&S framework for Net-centric systems collaborating over the GIG. The systems are at a much higher level of abstractions than any embedded system where state-variable bear much importance and criticalities. The current version of DUNIP addresses the need of these abstract systems. Inclusion of state-variables, more like on the lines of SCR will be included in future, to develop more sophisticated models.

CHAPTER 10: CONCLUSIONS AND FUTURE WORK

This research effort has provided contribution towards development of an integrated solution for the problem of executable models from user specified system requirements in structured English format. The solution is made available as a prototype called DUNIP which is an acronym for DEVS Unified Process. It added capabilities of enhanced MVC framework, DEVS Modeling Language and SOADEVS to the existing DEVS framework to make it net-centric capable.

The enhanced MVC complements the basic DEVS framework components, viz., the Experimental frame, the model, and the simulator. The integration of these two frameworks results in a well constructed control panel that provides a more comprehensive feature set and controls to calibrate the model and configure the simulation. The recent advances in DEVS technology, like variable structure modeling, real-time simulation tuning with rapid feedback, and model/simulator calibration, have been described; they help in the analysis and study of fast-changing network scenarios. The first major advantage of incorporating these technologies is the study and visualization of the "transition" effects when the model configuration is modified in a running simulation. Various methods of controlling simulation execution were explored as well as ways in which they can be used in different scenarios. The second major advantage of this enhanced MVC framework is the capability to reach the desired mission effectiveness or performance benchmarks in an active simulation. With variable structure capability, along with setting the bounds of any result parameter, the system can be observed to arrive at the corresponding "steady state." This methodology also aids in determining the most significant parameters for any complex system for which theoretical analysis is not feasible. These parameters are discussed with respect to the Net-Ready Key Performance Parameter (NR-KPP) set, in relation to DoDAF, and the advantages of identification of these parameters during the operational view design phase are emphasized.

We have addressed the problem of model interoperability with a novel approach of developing DEVSML as the transformation medium towards composability and dynamic scenario construction. The composed coupled models are then validated using the proposed universal atomic and coupled DTDs. The simulators validated at the server's end are maintained centrally such that the efforts of the community can be brought together through the standardized processes. Other advantage of using DEVSML as the communication medium gives the coder the independence to concentrate on the behavior of the component in their native languages (C++ and Java). In addition, it gives them the capability to share and integrate their models with that of other remote models and get that integrated validated model back in their own language. It also gives models the capability to get simulated with various simulator implementations that are stored at Server. This information is stored in meta-data that is contained in every model. Currently, this capability is meant only for Java but efforts are in progress to develop the corresponding methodology in C++ and will be reported in future. The research also proposes modification in DEVS formalism towards making them Service capable such that model continuity can be exploited towards deploying any DEVS component as a Service.

We addressed the problem of net-centricity with the development of SOADEVS, which is the SOA implementation of DEVS simulation engine so that models can be executed remotely as well as in a distributed manner using Simulation as a Service within SOA framework. The SOADEVS framework provides the capability to send models to a remote location, run the simulation from other computer and partition the hierarchical network over a set of server farms that host Simulation service.

The integration of enhanced MVC, DEVSML and SOADEVS along with the automated model generation from multifarious modes of requirement specifications resulted in a unifying framework called DUNIP. Figure 6.1 is reproduced again to summary the contribution of DUNIP.

This research effort has described various elements of DEVS Unified Process. With the developed DEVS Unified Process we now have the capability to:

1. Transform various forms of requirement specifications to DEVS models in an automated manner.
2. Generate automated Tester models from DEVS models to verify the Input/Output behavior of any DEVS component.

3. Transform any DEVS model to a Platform Independent Model (PIM) using DEVSML for model and library reuse and sharing leading to collaborated development
4. Simulate any valid DEVSML using the SOADEVS architecture exploiting the transparent simulator paradigm for model interoperability execution (for models implemented in disparate languages e.g. Java and C++)
5. Transform any DEVSML model to a Service component in SOA

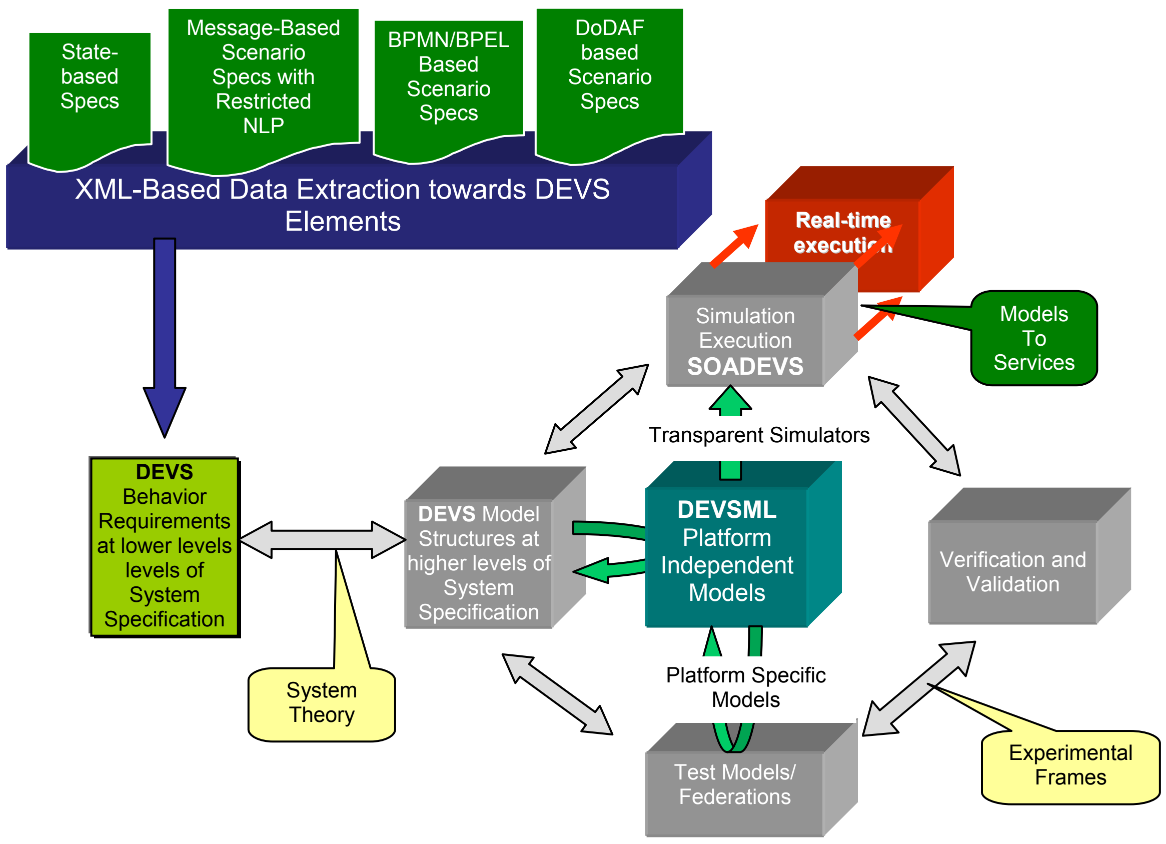

State-based Specs
Message-Based Scenario Specs with Restricted NLP
BPMN/BPEL Based Scenario Specs
DoDAF based Scenario Specs
XML-Based Data Extraction towards DEVS Elements
Real-time execution
Simulation Execution SOADEVS
Models To Services
Transparent Simulators
DEVS Behavior Requirements at lower levels levels of System Specification
DEVS Model Structures at higher levels of System Specification
DEVSML Platform Independent Models
Verification and Validation
Platform Specific Models
System Theory
Test Models/ Federations
Experimental Frames

Figure 10.1: The Complete DEVS Unified Process

Further, the problem of DoDAF in making it executable is looked into sufficient detail. Although the current DoDAF specification provides an extensive methodology for system architectural development, it is deficient in several related dimensions – absence of integrated modeling and simulation support, especially for model-continuity throughout the development process, and lack of associated testing support. To overcome these deficiencies, we described an approach to support specification of DoDAF architectures within a development environment based on DEVS-based modeling and simulation. The result is an enhanced system lifecycle development process that includes model-continuity based development and testing in an integral manner.

We have also introduced two new Operational Views OV-8 and OV-9 to address the additional information that is needed to make the DoDAF M&S compatible. We have also demonstrated the process to create OV-8 and OV-9 from the existing Operational Views. OV-8 contains the information about the Activity Component structure and how different Activities are interfaced with each other using the specified logical interfaces. OV-9 contains information about the constituent components inside an Operational Node and its corresponding DEVS model structure along with their mapping to the Activity components in OV-8. Together OV-8 and OV-9 provide a means to correlate Activity Components with accountable entities in an Operational node using logical interfaces. It is after the transformation of OV-8 and OV-9 into DEVS models that rules assigned to specific Activity or Entity components makes OV-8,9 server their complete purpose.

Automation using XML and simulation-tuning are important concepts that can be well executed and performed under current DEVS technology.

We also discussed the applicability of Modeling and Simulation for DoDAF and how this research effort is aligned with DoDAF architectures. We also demonstrate how DUNIP as a whole is used in various active projects at JITC including the GENETSCOPE and ATC-Gen project.

10.1 Future Work

The present research work has the following scope for future development:

- Towards standardization of DEVS formalism

  The DEVSML framework developed the atomic and coupled DTDs as meta-models towards collaborative DEVS model development. They are proposed with an idea towards their standardization where the DEVS community can come to a common ground for model reuse and repository management.

- Enhancement of DoDAF towards development of 'executable' architectures

  DoDAF Operational View was enhanced towards creation of two new views OV-8 and OV-9 which augment the information contained in OV-2,3,5,6. These new OVs are dedicated to the application of Modeling and Simulation domain towards creation of executable architecture from DoDAF OV specifications. Efforts are needed to include them in the next version of DoDAF specifications.

- A Prototype solution with underlying formal systems theory applied in whole or in-part to active projects at JITC.

  Many aspects of DUNIP have been applied independently to projects like ATC-Gen and GENETSCOPE, however, not as per say. Efforts in future would be directed in the framework of DUNIP development process.

- Refine the DUNIP process

  A Prototype was demonstrated as a final outcome of this research effort. More features like, validation, consistency checking, etc. should be added to develop it as a COTS product.

- Inclusion of more requirement specifications formats

  The current research effort described four formats to specify abstract requirement scenarios. The resulting DEVS models are at a higher level of abstraction. More formats could be included that utilize the full power of DEVS formalism and address criteria like elapsed time and state-variables.

- Performance evaluation of distributed SOADEVS protocol

  The SOADEVS protocol required tailoring of DEVS simulation protocol for SOA domain. Performance evaluation of this version is required to compare it with performance of DEVS protocol with current implementations like DEVS/RMI, DEVS/CORBA etc.

- Make it easier for other DEVS groups to participate in DEVSML and SOADEVS development by registering their simulators

  DEVSML is developed as a framework for collaborative model development and portable model specifications resulting from net-centric collaboration using XML middleware. Remote simulation is one capability that is also provided by DEVSML. Various simulator versions from different groups should be gathered and worked upon towards standardized DTDs for an efficient model-sharing system. Currently, two simulator implementations, viz. GenDEVS-ACIMS and xDEVS-Spain have been used to provide proof of concept. Better design of website offering DEVSML service should be designed that would facilitate various groups to submit their simulator implementations.

- Make prototype tool as an Educational aide

  The demonstrated prototype should be enhanced for teaching DEVS-based Modeling and Simulation courses. Various manuals and GUI enhancements would be added that facilitate learning and future development.

REFERENCES

Chapter 1

[ACI06] ACIMS software site: http://www.acims.arizona.edu/SOFTWARE/software.shtml Last accessed Nov 2006

[Car05] Carstairs, D.J., "Wanted: A New Test Approach for Military Net-Centric Operations", Guest Editorial, ITEA Journal, Volume 26, Number 3, October 2005

[Cho01] Cho, Y., B.P. Zeigler, H.S. Sarjoughian, Design and Implementation of Distributed Real-Time DEVS/CORBA, IEEE Sys. Man. Cyber. Conf., Tucson, Oct. 2001.

[CJC04] Chairman, JCS Instruction 3170.01D "Joint Capabilities Integration and Development System," 12 March 2004.

[CJC06] Chairman, JCS Instruction 6212.01D "Interoperability and Supportability of Information Technology and National Security Systems," March 8, 2006

[Dod03a] DoDAF Working Group , DoD Architecture Framework Ver. 1.0 Vol. III: Deskbook, DoD, Aug. 2003.

[Dod03b] DOD Instruction 5000.2 "Operation of the Defense Acquisition System," 12 May 2003.

[Gen06] GENETSCOPE(Beta Version) Software User's Manual, available from ACIMS center, University of Arizona.

[Hux04] X. Hu, and B.P. Zeigler, " Model Continuity in the Design of Dynamic Distributed Real-Time Systems", accepted by *IEEE Transactions On Systems, Man And Cybernetics— Part A: Systems And Humans*

[Mak06] E Mak, S Mittal, MH Hwang, "Automating Link-16 Testing using DEVS and XML", submitted to Journal of Defense Modeling and Simulation

[Pra05] "A Survey of Automated Test Case Generation"

[Sar00] Sarjoughian, H.S., B.P. Zeigler, "DEVS and HLA: Complimentary Paradigms for M&S?" Transactions of the SCS, (17), 4, pp. 187-197, 2000

[Wai01] Wainer, G., Giambiasi, N., Timed Cell-DEVS: modeling and simulation of cell-spaces". Invited paper for the book Discrete Event Modeling & Simulation: Enabling Future Technologies, Springer-Verlag 2001

[Weg02] Wegmann, A., "Strengthening MDA by Drawing from the Living Systems Theory", Workshop in Software Model Engineering, 2002

[Zei00] Zeigler, B., Kim, T., Praehofer, H., *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000

[Zha05] Zhang, M., Zeigler, B.P., Hammonds, P., DEVS/RMI-An Auto-Adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies, ITEA Journal, July 2005

Chapter 2

[Neb06] C Nebut, F Fleurey, YL Traon, J Jezequel, "Automatic Test Generation: A Use Case Driven Approach", IEEE Transactions on Software Engineering, 32(3), March 2006

[Utt06] M Utting, A Pretshner, B Legeard, "A Taxonomy of Model-based Testing", Working paper April 2006, University of Waikato, Hamilton, New Zealand ISSN 1170-487X

[Mak06] E Mak, S Mittal, MH Hwang, "Automating Link-16 Testing using DEVS and XML", submitted to Journal of Defense Modeling and Simulation

[Pra05] "A Survey of Automated Test Case Generation"

[Tor05] R. Torkar, "A Literature Study of Software Testing and the Automated Aspects Thereof". S-CORE Scientific Report, University of Troliihattan/Uddevalla, Sweden, techreport.tex, Rev. 95, 2005-05-10 @ 08:24:27Z

[Zei05a] BP Zeigler, D Fulton, P Hammonds, J Nutaro, "Framework for M&S-Based System Development and Testing in Net-centric Environment,", ITEA Journal, November 2005

[Zei05b] BP Zeigler, S Mittal, "Enhancing DoDAF with DEVS-Based System Life-cycle Process", IEEE International Conference on Systems, Man and Cybernetics, Hawaii, October 2005

[Jur04] N Juristo, AM Morano, S Vegas, "Reviewing 25 Years of Testing Technique Experiments", Empirical Software Engineering, Vol 9, 7-44, Kluwer Academic Publishers, Netherlands

[Off04] J. Offutt, A. Abdurazik, A. Baldini, "A Controlled Experiment Evaluation of Test Cases Generated from UML Diagram", 2004

[Sal04] A.M. Salem, L. Subramaniam, "Utlizing UML use cases for testing Requierements", International Conference on Software Eningeering research and Practice (SERP) 2004.

[Off03] J. Offutt, S. Liu, A. Abdurazik, P. Ammann, "Generating Test data from State Based Specifications", The Journal of Software Testing, Verfication and Reliability, 13(1):25-53.

[Sin03] A Sinha, CS Smidts, A Moran, "Enhanced Testing of Domain Specific Applications by Automatic Extraction of Axioms from Functional Specifications", In Proceedings of the 14th International Symposium on Software Reliability Engineering, pages 181-190, 2003

[Sit03] RO Sinnott, "Architecting Specification for Test Case Generation", In Proceedings of the 1st International Conference on Software Engineering and Formal Methods, pages 24-32, 2003

[Tot03] A. Toth, D. Varro, A. Pararicca, "Model Level Automatic Test Generation for UML State-charts", Sixth IEEE workshop on Design and Diagnostics of Electronic Circuits and System, (DDECS 2003)

[Bau02] B Baudry, F Fleurey, JM Jzquel, YLTraon, “Genes and Bacteria for Automated Test Cases Optimization in the .NET Environment”, IN Proceedings of the 13th International Symposium on Software Reliability Engineering, Pages 195-206, 2002

[Bri02] L Briand, Y Labiche, “A UML-Based Approach to System Testing”, Journal of Software and Systems Modeling, pp 10-42, 2002

[Bas02] F Basanieri, A Bertolino, E Marchetti, “The Cow_Suite Approach to Planning and Deriving Test suites in UML Projects”, Proceedings of Fifth International Conference on UML: Model Engineering Languages, Concepts, and Tools, pp. 383-397, 2002

[Cur02] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI," IEEE Internet Computing, vol. 6, no. 2, pp. 86-93, March-April 2002.

[Leg02] B Legeard, F Peureux, M Utting, “Automated Boundary Testing form Z and B”, Proc. Conf. Formal Methods, Europe, 2002

[Rie02] M Riebisch, I Philippow, M Gotze, “UML-Based Statistical Test-case Generation”, Proc. of International Conference on Net.ObjectDays, Vol. 2591, pp. 394-411, 2002

[Sri02] J Srinivasan, N Leveson, “Automated Testing from Specifications”, IN Proceedings of the 21st Digital Avionics Systems Conference, Volume 1, pages 6A2-1—6A2-8, 2002

[Wil02] C.E. Williams, “Software Testing and UML”, International Symposium on Software Reliability Engineering (ISSRE’99), Boca, Raton, November 1999

[Lin01] JC Lin, PL Yeh, “Automatic Test Data Generation for Path Testing using GAs”, Information Sciences: An International Journal, 131(1-4):47-64, 2001

[Pas01] I Parissis, J Vassy, “Strategies for Automated Specification-Based Testing of Synchronous Software”, In Proceedings of the 16th IEEE International Conference on Automated Software Engineering, pages 364-367, 2001

[Rot01] G Rothermel, RJ Untch, C Chu, “Prioritizing Test Cases for Regression Testing”, IEEE Transactions on Software Engineering, 17(10):929-948, 2001

[Tah01] L Tahat, B Vaysbur, B Koreland, A Bader, “Requirement-based Automated Black-box Test Generation”, Proc. of 25th Annual International Computer Software and Applications Conference, 2001

[Fro00] P Frohlich, J Link, “Automated Test Case Generation from Dynamic Models”, Proc. 14th European Conference on Object Oriented Programming, 2000

[Lut00] P Lutsky, “Information Extraction from Documents for Automating Software Testing”, Artificial Intelligence in Engineering, 14(1):63-69, 2000

[Off00] J. Offutt, A. Abdurazik, “Using UML Collaboration Diagrams for Static checking and Test generation”, Third International Conference on UML, York, UK, October 2000

[Whi00] JA Whittaker, “What is Software Testing? And Why it is so Hard?”, IEEE Software, January 2000

[Kim99] Y Kim, H Honh, S Cho, D Bae, S Cha, "Test Cases Generation from UML State Diagrams", IEEE Proc. Software, 146(4); 187-192, Aug 1999

[Off99a] J. Offutt, A. Abdurazik, "Generating Test cases from UML Specification", Second International Conference on the Unified Modeling Language (UML99), pp. 416-429, Fort Collins, CO, October 1999

[Off99b] AJ Offut, S Liu, "Generating Test Data from SOFL Specifications", Journal of Systems and Software, 49(1):49-62, 1999

[Par99] RP Pargas, MJ Harrold, RR Peck, "Test Data Generation using Genetic Algorithms", Software Testing, Verification and Reliability, 9(4):263-282, 1999

[Rys99] J Ryser, M Glinz, "A Secnario-Based Approach to Validating and Testing Software Systems using Statecharts", Proc. 12th International Conference on Software and Systems Engineering and their Applications", Dec. 1999

[Sou99] D D'Souza and A Wills, "Interaction Models: Uses, Case Actions, and Collaborations", Objects, Components, and Frameworks with UML: The Catalysis Approach", Addison-Wesley, 1999

[Rys98] J Ryser, S Berner, M Glinz, "On the State of the Art in Requirement-Based Validation and Test of Software", technical report, Inst. Fur Informatic, Univ. of Zurich, 1998

[Coc97] A Cockburn, "Structuring Use cases with Goals", Journal of Object Oriented Programming, pp. 35-40, 56-62, Sept./Oct., Nov./Dec., 1997

[Mic97] CC Michael, G McGraw, M Schatz, CC Walton, "Genetic Algorithms for Dynamic Test Data Generation", In Proceedings of the 12th IEEE International Conference on Automated Software Engineering, pages 307-308, 1997

[Eic96] NS Eickelmann, DJ Richardson, "An Evaluation of Software Test Architectures", Proceedings of 18th IEEE International Conference on Software Engineering, 1996

[Jon96] BF Jones, H Sthamer, DE Eyres, "Automatic Structural Testing Using Genetic Algorithm", Software Engineering Journal, 11(5):299-306, 1996

[Avr95] A Avritzer, EJ Weyuker, "The Automatic Generation of Load Test Suites and the Assessment of the Resulting Software", IEEE Transactions on Software Engineering, 21(9):705-716, 1995

[Dic93] J Dick, A Faivre, "Automating the Generation and Sequencing of Test Cases from Modelbased Specifications", Proceedings of International Symposium on Formal Methods, Europe, 1993

[Mey92] B Meyer, "Applying Design by Contract", Computer, 25(10):40-51, Oct 1992

[Ber91] G Bernot, MC Gaudeland, B Marre, "Software Testing Based on Formal Specifications: A theory and a Tool", Software Engineering Journal, 6(6):387-405, 1991

[Mye78] G.J. Myers, "The Art of Software Testing", John Wiley and Sons, New York, NY, USA, 1978

[Mar] M.E.Vieira, M.S. Dias, D.J. Richardson, "Object-Oriented Specification-Based Testing Using UML Statechart Diagrams", University of California

[Har] J. Hartmann, M. Viera, H. Foster, A. Ruder, "UML Based Test Generation and Execution",

[Zei00] Zeigler, B., Kim, T., Praehofer, H., *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000

[Fuj99] Fujimoto, R.M., *Parallel and Distribution Simulation Systems*, Wiley, 1999

[Seo04] Seo, C., Park, S., Kim, B., Cheon, S., Zeigler, B.P., Implementation of Distributed High-performance DEVS Simulation Framework in the Grid Computing Environment, Advanced Simulation Technologies conference (ASTC), Arlington, VA, 2004

[Che04] Cheon, S., Seo, C., Park, S., Zeigler, B.P., Design and Implementation of Distributed DEVS Simulation in a Peer to Peer Networked System, Advanced Simulation Technologies Conference, Arlington, VA, 2004

[Zha05] Zhang, M., Zeigler, B.P., Hammonds, P., DEVS/RMI-An Auto-Adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies, ITEA Journal, July 2005

[Kim04] Kim, K., Kang, W., CORBA-Based, Multi-threaded Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-hierarchical One, International Conference on Computational Science and Its Applications, ICCSA, Italy 2004

[Van01] Vangheluwe, H., Bolduc, L., Posse, E. DEVS Standardization: some thoughts, Winter Simulation Conference 2001

[Jan06] Janousek, V., Polasek, P., Slavicek, P., Towards DEVS Meta Language, In ISC Proceedings, Zwinjnaarde, BE 2006, p 69-73 ISBN-90-77381-26-0

[Fis02] Fishwick, P., XML Based Modeling and Simulation Using XML for Simulation Modeling, Proceedings of the 34th conf erence on Winter Simulation; exploring new frontiers, pg. 616-622, 2002

[Lar02] Lara, J., Vangheluwe, H., AToM3 as a Meta-CASE environment (DFD to SC), 4th International Conference on Enterprise Information Systems 2002

[Bad04] Badros, G. JavaML: a Markup Language for Java Source Code, Proceedings of the 9th International World Wide Web Conference on Computer Networks: the international journal of computer and telecommunication networking, pages 159-177

[Mar06] Martin, JLR, Mittal, S., Pena, MAL, Cruz, JM., A W3C XML Schema for DEVS Scenarios, DEVS Symposium 2007

[Yun02] Yung-Hsim, W., Yao-Chung, L., A Modeling and Simulation Example Using DEVSW, 2002

[ACI06] ACIMS software site:
http://www.acims.arizona.edu/SOFTWARE/software.shtml Last accessed Nov 2006

Chapter 3

[ACI06] http://www.acims.arizona.edu/SOFTWARE/software.shtml, last accessed Sep., 2006

[Bro98] Brown, A.W., and K. C.Wallnau. 1998. The current state of CBSE. *IEEE Software* 15 (5): 37-46.

[Bus98] Buss, A., Jackson, L. "Distributed Simulation Modeling: A Comparison of CORBA, HLA, and RMI". Proceedings of the 1998 Winter Simulation Conference. 1998.

[Che90] Chean, M., and L. A. B. Fortes. 1990.A taxonomy of reconfigurable techniques for fault-tolerant processor arrays. *IEEE Computer* 23 (1): 55-69

[Che02] Chen, X. 2002. Dependence management for dynamic reconfiguration of component-based distributed systems. In *Proceedings of the 17th IEEE International Conference on Automated Software Engineering*, pp. 279-84.

[CJC06] Chairman, JCS Instruction 6212.01D "Interoperability and Supportability of Information Technology and National Security Systems," March 6, 2006. Available from http://jitc.fhu.disa.mil/jitc_dri/pdfs/6212_01.pdf

[Cho01] Cho, Y., B.P. Zeigler, H.S. Sarjoughian, Design and Implementation of Distributed Real-Time DEVS/CORBA, IEEE Sys. Man. Cyber. Conf., Tucson, Oct. 2001.

[Dic05] Department of Defense Interoperability Communications Exercise (DICE) 2005, available at: jitc.fhu.disa.mil/dice_conf/2005/initial/downloads..., last accessed Oct 2005

[Dod03a] DoDAF Deskbook

[Gen06] GENETSCOPE(Beta Version) Software User's Manual, available from ACIMS center,

[Hux03] X. Hu, B. P. Zeigler, S. Mittal, "Dynamic Configuration in DEVS Component-based Modeling and Simulation", SIMULATION: Transactions of the Society of Modeling and Simulation International, November 2003

[Hux05] X. Hu, and B.P. Zeigler, " Model Continuity in the Design of Dynamic Distributed Real-Time Systems", accepted by *IEEE Transactions On Systems, Man And Cybernetics— Part A: Systems And Humans*

[Kim90] T.G. Kim, C. Lee, E.R. Christensen, B.P. Zeigler, "System entity structuring and model base management", IEEE Transactions on Systems, Man and Cybernetics, Volume 20, Issue 5, Sept-Oct., 1990

[Mit03d] S. Mittal, B. P. Zeigler, "Modeling/Simulation Architectures for Autonomous Computing", Autonomic Computing Workshop: The Next Era of Computing, January 2003

[Mit04c] Mittal, S., B. P. Zeigler, Phillip Hammonds, Mahesh Veena, "Network Simulation Environment for Evaluating and Benchmarking HLA/RTI Experiments", JITC Report, Fort Huachuca, December 2004.

[Mit06a] Mittal.S., Mak, E. Nutaro, J.J., "DEVS-Based Dynamic Modeling & Simulation Reconfiguration using Enhanced DoDAF Design Process", special issue on DoDAF, Journal of Defense Modeling and Simulation, Dec 2006

[Mit05b] Mittal, S., B. P. Zeigler, "Dynamic Simulation Control with Queue Visualization", Summer Computer Simulation Conference SCSC'05, Philadelphia, July 2005

[Mit07e] Mittal, S., Risco-Martin, J.L., Zeigler, B.P. "DEVSML: Automating DEVS Simulation over SOA using Transparent Simulators", DEVS Syposium, 2007

[Nut05] J. Nutaro, P. Hammonds, "Combining the Model/View/Control Design Pattern with the DEVS Formalism to Achieve Rigor and Reusability in Distributed Simulation", Journal of Defense Modeling and Simulation, May 2005

[Sar01a] Sarjoughian, H., Cellier, F.E., Editors, Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies, Spring-Verlag, NY, **2001.**

[Sar01b] H. Sarjoughian, B. Zeigler, and S. Hall, "A Layered Modeling and Simulation Architecture for Agent-Based System Development", Proceedings of the IEEE 89 (2); 201-213, 2001

[Tol03] A. Tolk, S. Solick, "Using the C4ISR Architecture Framework as a Tool to Facilitate V&V for Simulation Systems within the Military Application Domain", Simulation Interoperability Workshop, Spring 2003

[Uhr93] Uhrmacher, A. M. 1993. Variable structure models: Autonomy and control—Answers from two different modeling approaches. In *Proceedings on AI, Simulation, and Planning in High Autonomy Systems*, pp. 133-9.

[Zei00] B. P Zeigler, H. Praehofer, T. G. Kim, "Theory of Modeling and Simulation", Academic Press, 2000University of Arizona. [Zei03] Zeigler, B.P., DEVS Today: Recent Advances in Discrete Event-based Information Technology, MASCOTS Conference, 2003

[Zei05] B. P. Zeigler, D. Fulton, P. Hammonds, J. Nutaro, "Framework for M&S-Based System Development and Testing in Net-centric Environment," , ITEA Journal, Volume 26, Number 3, October 2005

[Zei85] Zeigler, B. P., and R. G. Reynolds. 1985. Towards a theory of adaptive computer architectures. In *Proceedings of the 5th International Conference on Distributed Computing Systems*, pp. 468-75.

[Zei93] Zeigler, B. P., and A. Louri. 1993.A simulation environment for intelligent machine architecture. *Journal of Parallel and Distributed Computing* 18:77-88.

Chapter 4

[bpm] Business Process Modeling Notation (BPMN) www.bpmn.org

[bpel] Business Process Execution Language (BPEL) http://en.wikipedia.org/wiki/BPEL

[omg] Object Modeling Group (OMG) www.omg.org

[rat] IBM Rational Software www.rational.com

[spa] Enterprise Architect http://www.sparxsystems.com.au/

[vis] Microsoft Visio: http://office.microsoft.com/en-us/visio/default.aspx

[Atk04] K. Atkinson, "Modeling and Simulation Foundation for Capabilities Based Planning", Simulation Interoperability Workshop Spring 2004

[CJC04] Chairman, JCS Instruction 3170.01D "Joint Capabilities Integration and Development System," 12 March 2004.

[CJC06] Chairman, JCS Instruction 6212.01D "Interoperability and Supportability of Information Technology and National Security Systems," March 8, 2006

[Dan04] F. Dandashi, H. Ang., C. Bashioum, "Tailoring DoDAF to Support a Service Oriented Architecture", White paper, Mitre Corp, December 2004

[Dod03a] DoDAF Working Group , DoD Architecture Framework Ver. 1.0 Vol. III: Deskbook, DoD, Aug. 2003.

[Dod03b] DOD Instruction 5000.2 "Operation of the Defense Acquisition System," 12 May 2003.

[Dod04c] DoD Architecture Framework Working Group 2004, DOD Architecture Framework Ver. 1.0 Volume 1 Definitions and Guidelines, 9 February 2004, Washington, D.C.

[Dod05c] DoD Metadata Registry and Clearinghouse, http://diides.ncr.disa.mil/mdregHomePage/mdregHome.portal/, last accessed Jan 9, 2005

[Hux04] X. Hu, and B.P. Zeigler, " Model Continuity in the Design of Dynamic Distributed Real-Time Systems", accepted by *IEEE Transactions On Systems, Man And Cybernetics— Part A: Systems And Humans*

[Lee05] J. Lee, M. Choi, J. Jang. Y, Park, J. Jang,, B. Ko, "The Integrated Executable Architecture Model Development by Congruent Process, Methods and Tools", The Future of C2, 10th International Command and Control Research and Technology Sysmposium

[Mit06a] Saurabh Mittal, "Extending DoDAF with Bifurcated Model-continuity Based Life-cycle Methodology," Journal of Defense Modeling and Simulation, Vol.3, No. 2., 2006.

[Ros04] J.D. Rosen, J.L. Parenti, J. Hamilton, "The Domain of Interoperability in the US Department of Defense", Chapter 2, Joint Command and Control Interoperability: Cutting the Guardian Knot". Available at: http://www.eng.auburn.edu/users/hamilton/security/spawar/ .

[Sar01] Discrete Event Modeling and Simulation Technologies: A Tapestry of Systems and AI-Based Theories and Methodologies Editors: Hessam S. Sarjoughian , François E. Cellier, Spring-Verlag, NY, **2001.**

[Tel04] C. Kobryn, C. Sibbald, "Modeling DoDAF Compliant Architectures: The Telelogic Approach for complying with DoD Architectural Framework", White paper, October 2004.

[Wad02] L.W Wagenhals, S. Haider, A.H.Levis, "Synthesizing Executable Models of Object Oriented Architectures", Workshop on Formal Methods Applied to Defense Systems, Adelaide, Australia, June 2002

[Zei00] B. P Zeigler, H. Praehofer, T. G. Kim, "Theory of Modeling and Simulation", Academic Press, 2000

[Zei03] B. P.Zeigler, DEVS Today: Recent Advances in Discrete Event-based Information Technology, MASCOTS Conference, 2003

[Zei05a] B.P. Zeigler, S. Mittal, "Enhancing DoDAF with DEVS-Based System Life-cycle Process", IEEE International Conference on Systems, Man and Cybernetics, Hawaii, October 2005

[Zei05b] B. P. Zeigler, D. Fulton, P. Hammonds, J. Nutoro, "Framework for M&S-Based System Development and Testing in Net-centric Environment,", ITEA Journal, November 2005

Chapter 5

[Mit06b] Saurabh Mittal, Eddie Mak, and James Nutaro, "DEVS-Based Dynamic Model Reconfiguration and Simulation Control in the Enhanced DoDAF Design Process," Journal of Defense Modeling and Simulation, Vol.3, No. 4., 2006.

[Mak06] Eddie Mak, "Automated Testing using XML and DEVS," http://www.acims.arizona.edu, last accessed June 30, 2006

[Nap97] Technology for the United States Navy and Marine Corps, 2000-2035 Becoming a 21st-Century Force: Volume 9: Modeling and Simulation (1997), National Academy Press.

[Nap02] Modeling and Simulation in Manufacturing and Defense Acquisition: Pathways to Success (2002), National Academy Press.

[Nut05] James Nutaro, Phil Hammonds, "Combining the Model/View/Control Design Pattern with the DEVS Formalism to Achieve Rigor and Reusability in Distributed Simulation," Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, pp. 19-28, Vol. 1, No. 1, 2005

[Zei05] B.P. Zeigler, D. Fulton, P. Hammonds, J. Nutaro, "Framework for M&S-Based System Development and Testing in Net-centric Environment," ITEA Journal of Test and Evaluation, Vol. 26, No. 3, 2005.

Chapter 6

[ACI06] ACIMS software site: http://www.acims.arizona.edu/SOFTWARE/software.shtml Last accessed Nov 2006

[Bad05] Badros, G. JavaML: a Markup Language for Java Source Code, Proceedings of the 9th International World Wide Web Conference on Computer Networks: the international journal of computer and telecommunication networking, pages 159-177

[Cho01] Cho, Y., B.P. Zeigler, H.S. Sarjoughian, Design and Implementation of Distributed Real-Time DEVS/CORBA, IEEE Sys. Man. Cyber. Conf., Tucson, Oct. 2001.

[DML] DEVSML – A Web Service Demonstration http://150.135.218.205:8080/devsml/

[Hux03] Hu X., A Simulation Based Software Development Methodology for Distributed Real-time Systems, PhD Dissertation, Fall 2003

[Jan06] Janousek, V., Polasek, P., Slavicek, P., Towards DEVS Meta Language, In ISC Proceedings, Zwinjnaarde, BE 2006, p 69-73 ISBN-90-77381-26-0

[Mit06a] Mittal.S., Mak, E. Nutaro, J.J., “DEVS-Based Dynamic Modeling & Simulation Reconfiguration using Enhanced DoDAF Design Process”, special issue on DoDAF, Journal

[Mit07e] Mittal, S., Risco-Martin, J.L., Zeigler, B.P. “DEVSML: Automating DEVS Simulation over SOA using Transparent Simulators”, DEVS Syposium, 2007

[OPML] OOPML: http://xml.coverpages.org/oopml.html

[OXML] o:XML: www.o-xml.org

[Sar00] Sarjoughian, H.S., B.P. Zeigler, "DEVS and HLA: Complimentary Paradigms for M&S?" Transactions of the SCS, (17), 4, pp. 187-197, 2000

[Sun] http://java.sun.com/developer/technicalArticles/WebServices/soa/

[Wai01] Wainer, G., Giambiasi, N., Timed Cell-DEVS: modeling and simulation of cell-spaces”. Invited paper for the book Discrete Event Modeling & Simulation: Enabling Future Technologies, Springer-Verlag 2001

[WSD] WSDL http://www.w3.org/TR/wsdl

[XML] XML: http://www.w3.org/XML/

[Zei00] Zeigler, B., Kim, T., Praehofer, H., *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000of Defense Modeling and Simulation, Dec 2006

[Zha05] Zhang, M., Zeigler, B.P., Hammonds, P., DEVS/RMI-An Auto-Adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies, ITEA Journal, July 2005

Chapter 7

[ACI06] ACIMS software site: http://www.acims.arizona.edu/SOFTWARE/software.shtml Last accessed Nov 2006

[Atk04] K. Atkinson, “Modeling and Simulation Foundation for Capabilities Based Planning”, Simulation Interoperability Workshop Spring 2004

[Car05] Carstairs, D.J., “Wanted: A New Test Approach for Military Net-Centric Operations”, Guest Editorial, ITEA Journal, Volume 26, Number 3, October 2005

[CJC04] Chairman, JCS Instruction 3170.01D “Joint Capabilities Integration and Development System,” 12 March 2004.

[CJC06] Chairman, JCS Instruction 6212.01D “Interoperability and Supportability of Information Technology and National Security Systems,” March 8, 2006

[DML] DEVSML – A Web Service Demonstration http://150.135.218.205:8080/devsml/

[Dod03a] DoDAF Working Group , DoD Architecture Framework Ver. 1.0 Vol. III: Deskbook, DoD, Aug. 2003.

[Dod03b] DOD Instruction 5000.2 "Operation of the Defense Acquisition System," 12 May 2003.

[Dun07] DUNIP: A Prototype Demonstration http://www.acims.arizona.edu/dunip/dunip.avi

[Hux04] X. Hu, and B.P. Zeigler, " Model Continuity in the Design of Dynamic Distributed Real-Time Systems", accepted by *IEEE Transactions On Systems, Man And Cybernetics— Part A: Systems And Humans*

[Weg02] Wegmann, A., "Strengthening MDA by Drawing from the Living Systems Theory", Workshop in Software Model Engineering, 2002

[xDEVS] XDEVS web page: http://itis.cesfelipesegundo.com/~jlrisco/xdevs.html

Chapter 8

[Buc04] J.B. Buchheister, "Net-centric Test & Evaluation", Command and Control Research and Technology Symposium: The Power of Information Age Concepts and Technologies, 2004, available at: www.dodccrp.org/events/2004/CCRTS_San_Diego/CD/papers/208.pdf, last accessed October 2005

[CJC06] Chairman, JCS Instruction 6212.01D "Interoperability and Supportability of Information Technology and National Security Systems," March 8, 2006

[Gen06] GENETSCOPE(Beta Version) Software User's Manual, available from ACIMS center, University of Arizona.

[JITC] JITC reports for SCOPE command for year 2005, JITC, latest data as of Oct 2005.

[Nut05] James Nutaro, Phil Hammonds, "Combining the Model/View/Control Design Pattern with the DEVS Formalism to Achieve Rigor and Reusability in Distributed Simulation," Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, pp. 19-28, Vol. 1, No. 1, 2005

[Mak06] Eddie Mak, "Automated Testing using XML and DEVS," http://www.acims.arizona.edu, last accessed June 30, 2006

[Mit06a] Saurabh Mittal, "Extending DoDAF with Bifurcated Model-continuity Based Life-cycle Methodology," Journal of Defense Modeling and Simulation, Vol.3, No. 2., 2006.

[Mit06b] Mittal.S., Mak, E. Nutaro, J.J., "DEVS-Based Dynamic Modeling & Simulation Reconfiguration using Enhanced DoDAF Design Process", special issue on DoDAF, Journal of Defense Modeling and Simulation, Dec 2006

[Tel04] C. Kobryn, C. Sibbald, "Modeling DoDAF Compliant Architectures: The Telelogic Approach for complying with DoD Architectural Framework", White paper, October 2004.

[Zin04] A. W. Zinn, “The Use of Integrated Architectures to support Agent Based Simulation: An Initial Investigation”, MS Thesis, AFIT/GSE/ENY/04-M01, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 2004

Chapter 9

[Zdu02] Zdun, U., Dustdar, S. (2007). Model-Driven Integration of Process-Driven SOA Models. *International Journal of Business Process Integration and Management*, Inderscience, forthcoming.

[Hiet95] Heitmeyer, C., Bull, A., Gasarch, C., Labaw, B., SCR: A Toolset for Specifying and Analyzing Requirements. Proceedings of the Tenth Annual Conference on Computer Assurance (COMPASS '95), Gaithersburg, MD, June 25-29, 1995, pp. 109-122