Université Fédérale
Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Présentée et soutenue par :
Sangeeth saagar PONNUSAMY
le lundi 26 septembre 2016

Titre :

SIMULATION PRODUCT FIDELITY: A QUALITATIVE & QUANTITATIVE SYSTEM ENGINEERING APPROACH

École doctorale et discipline ou spécialité :
EDSYS : Systèmes embarqués 4200046

Unité de recherche :
Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS (UPR 8001)

Directeur/trice(s) de Thèse :

Vincent ALBERT

Jury :

Alexandre NKETSA, Président du Jury
Allel HADJALI, Rapporteur
Gabriel WAINER, Rapporteur
Hans VANGHELUWE, Examinateur
Marc PANTEL, Examinateur
Vincent ALBERT, Directeur de Thèse
Patrice THEBAULT, Invité

To
Amma
&
Appa

Acknowledgements

No work, however small it may be, will ever be complete without acknowledging the myriad of people who have contributed to it. First and foremost, I would like to thank my academic and industrial supervisors, Dr.Vincent Albert and Patrice Thebault for their wisdom, passion and diligence. Thank you for introducing me to the best of both industrial and academic worlds with unparalleled freedom and for being constant source of support and encouragement throughout my PhD. It has been a rewarding experience, both personally and professionally, and now you can live in peace free of my persistent and often hyperbolic questions!

I would like to thank the jury members, in particular Prof. Wainer and Hadjali for their time to review my (not so coherent) thesis and suggest valuable insights and corrections. I extend my gratitude to the other members of the jury, especially, Prof. Vangheluwe, Prof. Nketsa and Dr.Pantel for their time, effort, patience and feedback.

I would like thank the Simulation R&T team of Airbus for the wonderful three years. In particular, I would like to thank Richard for his support, 'hyper hero' Olivier for being a source of discussion (and coffee!), Dino, and all the finest members of the team. I must also thank Bernard for his unique wisdom and attention to detail (especially at my bizarre English!) and Stephane for his support, especially during the beginning of my PhD. I would like to thank the entire Simulation team, its various departments, its former and current heads Vincent and Bruno, Evelyne and all its members for accommodating me (and my French!). I am fortunate to work with many excellent people of finest caliber all across Airbus and as it would be a futile exercise to list every one of them, for the sake of brevity (and laziness), I simply say, Thank you!

I would like to thank my laboratory, LAAS-CNRS, the System Integration team and its members, in particular, Hamid for his support, Alexandre for his reviews and Abd-el-Kader for his discussions. The presence of Min Zhu, Li Zheng, Rui Xue, Diego and other made it all the more memorable and a fun place to work. I must also thank Bernard and Silvano of the VERTICS team for helping out with tools and the entire Informatique Critique community. I gratefully acknowledge the support from ANRT for this PhD and a special thank you to my doctoral school EDSYS, its secretary Helen in particular for her assistance. The administrative staff of CNRS and Airbus also have withstood my constant barrage of queries and helped me out in need, albeit at the last minute in some cases.

Though, one need not explicitly say thank you to their friends, nevertheless I must do it. I am fortunate to have excellent amigos all these years who were always there for me and have been a never ending source of fun (and frustration!). Thank you Somaji, 'singam' Sri, 'sheikh' Rajesh & Falguni, Hindu & 'vin' Thibaut, LSK, STK, 'animo' and 'netta' Prems, 'thala' Thiya, Ananthu, Sivaji, 'pigbull' Danny, Sagar, Vasu & Dom, 'humpty' Harish, 'puli' Laxman, 'akka' SuSam and the magnificient 'Vettipasanga' group. I must also thank Rakesh, Jyothi, Krishna, Vikram, Ankit and scores of others – *Nandri Makkale!*

I am also grateful to all my teachers and other wonderful personalities who have taught me, guided me and instilled confidence in me all through these years. I am indebted to the love and affection of my family members including anna, akka, meema, sithappa, abhi, bonda, priya and others.

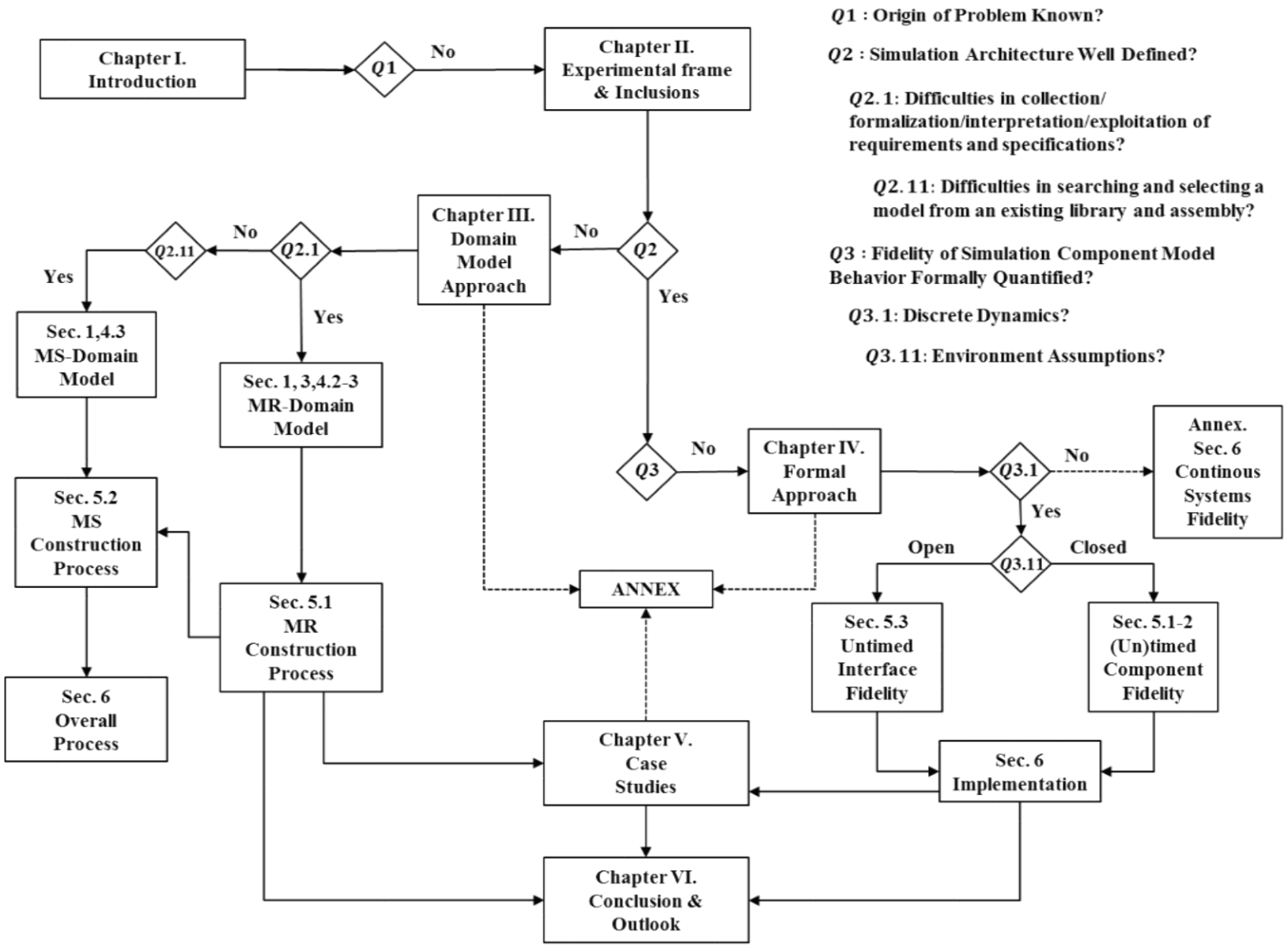
Lastly, though this must go without saying, this journey would not have been simply possible without my parents – Thank you, I owe you everything!

ABSTRACT

In using Modeling and Simulation for the system Verification & Validation activities, often the difficulty is finding and implementing consistent abstractions to model the system being simulated with respect to the simulation requirements. A proposition for the unified design and implementation of modeling abstractions consistent with the simulation objectives based on the computer science, control and system engineering concepts is presented. It addresses two fundamental problems of fidelity in simulation, namely, for a given system specification and some properties of interest, how to extract modeling abstractions to define a simulation product architecture and how far does the behaviour of the simulation model represents the system specification. A general notion of this simulation fidelity, both architectural and behavioural, in system verification and validation is explained in the established notions of the experimental frame and discussed in the context of modeling abstractions and inclusion relations. A semi-formal ontology based domain model approach to build and define the simulation product architecture is proposed with a real industrial scale study. A formal approach based on game theoretic quantitative system refinement notions is proposed for different class of system and simulation models with a prototype tool development and case studies. Challenges in research and implementation of this formal and semi-formal fidelity framework especially in an industrial context are discussed.

THESIS ROADMAP

A brief roadmap of how to read this thesis according to the reader's need is presented in this section. Broadly, our unified approach to the problem of fidelity of simulation models consists of two axes of research, namely, semi-formal axis based on domain model approach using ontologies and formal axis based on theory of formal verification, game theory and control. Though these two approaches are complimentary to each other in developing a simulation product with sufficient fidelity, in principle they can be read independent of the other. Similarly, within an approach, there are subsections which are at times could be considered independent according to the reader's need. In order to facilitate this and to provide a coherent vision on our approach which attempts to deal with various facets of the fidelity problem, a road map of thesis is presented from the perspective of a reader's need. For example, a reader might choose a different approach to overcome fidelity issues at the architectural level but might still choose to consider the behavioural approach presented in this thesis and vice versa.


Chapter I. Introduction
Q1
No
Chapter II. Experimental frame & Inclusions
Q1 : Origin of Problem Known?
Q2 : Simulation Architecture Well Defined?
Q2. 1: Difficulties in collection/ formalization/interpretation/exploitation of requirements and specifications?
Q2. 11: Difficulties in searching and selecting a model from an existing library and assembly?
Q3 : Fidelity of Simulation Component Model Behavior Formally Quantified?
Q3. 1: Discrete Dynamics?
Q3. 11: Environment Assumptions?
Chapter III. Domain Model Approach
Q2
No
Yes
Q2.1
No
Yes
Q2.11
Yes
Sec. 1,4.3 MS-Domain Model
Sec. 1, 3,4.2-3 MR-Domain Model
Sec. 5.2 MS Construction Process
Sec. 5.1 MR Construction Process
Sec. 6 Overall Process
Q3
No
Chapter IV. Formal Approach
Q3.1
No
Annex. Sec. 6 Continous Systems Fidelity
Yes
Q3.11
Open
Closed
Sec. 5.3 Untimed Interface Fidelity
Sec. 5.1-2 (Un)timed Component Fidelity
ANNEX
Chapter V. Case Studies
Sec. 6 Implementation
Chapter VI. Conclusion & Outlook

CONTENTS

INTRODUCTION **1**

1. Motivation 1
2. Objectives 4
3. Thesis Contributions 10
4. Thesis Plan 13

EXPERIMENTAL FRAME & INCLUSIONS **15**

1. Background 15
2. Preliminaries 19
3. Theory of Modeling & Simulation Framework 23
4. Fidelity Framework & Inclusions 32
5. Conclusion 36

SYSTEM SIMULATION DOMAIN MODEL APPROACH **37**

1. Introduction 37
2. State of Art 38
3. Preliminaries 40
4. Simulation Fidelity Domain Model 41
5. Process Overview 48
6. Operational Perspective 55
7. Conclusion 56

BEHAVIOURAL FIDELITY METRIC **59**

1. Introduction 59
2. Behavioural Fidelity 59
3. State of Art 66

4. Preliminaries 68
5. Formal Fidelity Quantification 70
6. Implementation 86
7. Conclusion 90

APPLICATION CASE STUDIES 93

1. Domain Model Approach Case Study - Aircraft Nacelle Anti-Ice System 93
2. Formal Approach Case Studies 101
3. Conclusion 110

OUTLOOK & CONCLUSION 111

1. Outlook on Semi-Formal Approach 111
2. Outlook on Formal Approach 114
3. Conclusion 121

BIBLIOGRAPHY 123

LIST OF PUBLICATIONS 137

ANNEXURE 141

LIST OF SYMBOLS & ACRONYMS

c	:	Abstraction Class
EF	:	Experimental Frame
EF_{sim}	:	Simulation Experimental Frame
EF_{sys}	:	System Experimental Frame
I	:	Mode Interconnection
IA	:	Interface Automata
IO	:	Input Output
IOU	:	Intention Of Use
Ker	:	Nullspace Operator
MR	:	Model Requirements
MS	:	Model Specification
$\mathrm{M_A}$	:	Experimental Frame Acceptor
$\mathrm{M_{Env}}$	:	Experimental Frame Environmental Components
$\mathrm{M_G}$	:	Experimental Frame Generator
$\mathrm{M_{Sim}}$	:	Simulation Model
$\mathrm{M_{Sys}}$	:	System Model
$\mathrm{M_T}$	:	Experimental Frame Transducer
OC	:	Operating Condition
OM	:	Operating Mode
SD	:	System Design Specification
SDU	:	Simulation Domain Of Use
SOU	:	Simulation Objective Of Use
SUT	:	System Under Test
TR	:	Test Requirements
X	:	State Set of a Transition System
$\mathcal{R}_\varepsilon$	:	Quantitative Reachability
$\varepsilon_{\mathrm{F}}^{\mathrm{abs}}$	:	Absolute Fidelity Distance

ε_F^{rel}	:	Relative Fidelity Distance
ε_{SDU}	:	SDU Fidelity Distance
ε_{SOU}	:	SOU Fidelity Distance
T^{IA}	;	Interface Automata
$\alpha_{\varepsilon_{SDU}}$	:	Implemented Abstraction
$\alpha_{\varepsilon_{SOU}}$	:	Allowable Abstraction
ΔT	:	Transition Timing Distance
$\preccurlyeq$	:	Preorder Relation
β	:	Derivability
γ	:	Applicability
H	:	Linear Surjective Map
T	:	Untimed Automata
f	:	Function
t	:	Time
$\mathcal{T}$	:	Timed Transition System
$\mathcal{L}$	:	Language of a Transition System
$\mathfrak{E}$	:	Domain Model Concept
$\mathfrak{M}$	:	Domain Model
$\mathfrak{I}$	:	Domain Model Instance
$\mathfrak{g}$	:	Game Model
$\mathfrak{r}$	:	Domain Model Relationship
α	:	Abstraction
ρ	:	Play in a Game
τ	:	Transition
φ	:	Property

LIST OF FIGURES

Figure 1: System V cycle 1
Figure 1.1: Measured Fidelity Approach 3
Figure 1.2: Fidelity Challenges 4
Figure 2.1: Proposed Approach 9

Figure 2.1: Simulation Product 19
Figure 2.2: Airbus V Cycle 20
Figure 2.3: Simulation Product – SDU & SOU 21
Figure 2.4: Simulation Product Development Overview 22
Figure 2.5: Simulation Product Development Process 22
Figure 3.1: EF & SUT 24
Figure 3.2: Experimental Frame, Model & Simulator 24
Figure 3.3: Real World vs Simulation World 25
Figure 3.4: Experimental Frame 26
Figure 3.5: Applicability, Derivability & Abstractions - modified from 28
Figure 3.6: SOU & SDU - Applicability, Derivability & Abstractions 31
Figure 3.7: EF Definition & SUT 32
Figure 4.1: Abstraction in Modeling & Simulation 33

Figure 4.1: SBFIO Domain Model 41
Figure 4.2: Operating Modes & Transition Diagram 43
Figure 4.3: Mode Dependency Example 44
Figure 4.4: Operational Modes 44
Figure 4.5: Modeling Abstraction Taxonomy 46
Figure 5.1: MR Construction Process Overview 49
Figure 5.2: Lattice for Voltage Assumption Class 52
Figure 5.3: Model Selection Results 54
Figure 5.4: Model Assembly 55
Figure 6.1: Operational View of M&S domain model 56

Figure 2.1: Simulation Fidelity 62
Figure 2.2: System & Simulation Model 62
Figure 2.3: System and Simulation Models 64
Figure 5.1: Linear Weighting 71
Figure 5.2: Simulation Models Relative Fidelity 72

Figure 5.3: Blocking Game 73
Figure 5.4: Non-blocking Game 74
Figure 5.5: Quantitative Timed Reachability Graph 76
Figure 5.6: Trace Inclusion vs Refinement principles 78
Figure 5.7: Controller System model 79
Figure 5.8: Controller Simulation Models 79
Figure 5.9: EF & SUT Composition 81
Figure 6.1: Implementation 87
Figure 6.2: Branching in Timed Automata 88
Figure 6.3: Branching Implementation 89
Figure 6.4: Automata Games Implementation 89

Figure 1.1: NAIS System 94
Figure 1.2: Experimental Frame of NAIS Controller 94
Figure 1.3: NAIS TR & SD Instances 96
Figure 1.4: NAIS TR & SD Inferred Instances 96
Figure 1.5: NAIS Instances Design Space Exploration 97
Figure 1.6: SUT Consistency Evaluation 98
Figure 1.7: Architectural Comparison 99
Figure 1.8: NAIS Instances Design Space Exploration 99
Figure 1.9: Operating Modes of Valve and Solenoid 100
Figure 2.1: Processor Experimental Frame 101
Figure 2.2: Buffer System Model 102
Figure 2.3: Buffer Simulation Models 102
Figure 2.4: Trajectories Fidelity Distribution 103
Figure 2.5: Quantitative Reachability 104
Figure 2.6: Trajectories Distribution 104
Figure 2.7: Buffer Timed System Model 105
Figure 2.8: Buffer Timed Simulation Model 105
Figure 2.9: Trace length vs Number of traces 106
Figure 2.10: Total number of plays distribution 106
Figure 2.11: Transition difference distribution 107
Figure 2.12: NAIS Controller Model 108
Figure 2.13: Fidelity distribution 109
Figure 2.14: Fidelity evolution 109

Figure 2.1: Timed Automata & Reachability 115
Figure 2.2: Effect of Sampling 116
Figure 2.3: Fidelity Tolerance Example 117
Figure 2.4: Model Synthesis 118
Figure 2.5: Controller Simulation Model Synthesis 120
Figure 2.6: Reachability perspective of synthesis procedure 121

LIST OF TABLES

Table 4.1: Model Abstraction Library 47
Table 5.1. Battery Model Abstraction Library 51

Table 2.1: Quantitative Reachability Graph 64
Table 2.2: Dynamic System Classification I 65
Table 2.3: Dynamic System Classification II 65
Table 5.1: Equal weighted error of models 84
Table 5.2: Equal weighted error of models 86

CHAPTER I

INTRODUCTION

Verification and Validation (V&V) activities are carried out to determine the compliance of a system, also called as System Under Test (SUT), with their specifications and fitness for their intended use respectively. Such V&V activities are usually illustrated in the classical V cycle as seen in figure 1 [Airbus] and this cycle can be broadly classified into two parts. The left branch of the cycle corresponds to design V&V where the SUT is virtual i.e. under construction and the right branch corresponds to product V&V where the SUT is physical i.e. built.

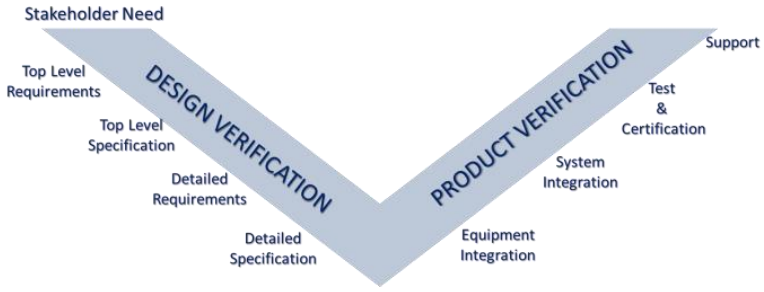


Figure 1: System V cycle, [Airbus]

In the V&V of complex engineering systems, the SUT is integrated with the other systems called environmental systems to perform some test cases and evaluate its behaviour against some user defined criteria such as performance, robustness etc. However, due to realistic limitations such as safety, cost, risk, and availability of systems this is seldom possible and these environmental systems are usually replaced by their representations i.e. models. Thus it becomes necessary to develop reasonable abstractions i.e. models of such environmental systems such that the resulting V&V activity yields same conclusions such as the ones carried out with real systems. This ability of models to replace systems by faithfully reproducing their behaviour is called simulation fidelity or simply 'fidelity' and it has been widely discussed in literature [Gross,1999], [Kim,2004], [Sancandi,2011], [Roza,2004].

1. MOTIVATION

Modeling and Simulation (M&S), in general, are analysis and decision means to assess performances, functionalities and operations of a system of interest [Brade,2004]. M&S is being increasingly used in product life cycle development in general and V&V activities in particular. It is used in both phases of the V cycle illustrated in figure 1, to perform V&V of the specification and the design of the SUT in the design verification phase, and of the integrated configuration and the SUT operational environment in extreme conditions, such as failure, in the product verification phase.

The pervasive beneficial impact of M&S is especially relevant in high technology industries such as aerospace where simulation can add value addition to the whole product development chain. At the Airbus Operations SAS of the Airbus Group, hitherto referred simply as Airbus, there is an ever-increasing tendency to use M&S during the aircraft life cycle with an ultimate objective of using Virtual Testing (VT) as a means of certification. Virtual testing can be defined as a method of testing based on the usage of simulation instead of physical test and on the usage of modeling of the physical article, instead of the physical article [CRESCENDO,2009]. A natural and logical evolution to the widespread use of M&S during the development of new aircraft, is to extend this VT as an acceptable Means Of Compliance (MOC) for certification. However, this necessitates the demonstration of the adequacy of M&S process in representing the reality. Thus the effectiveness of simulation in reproducing the reality i.e. *fidelity* needs to be evaluated *apriori* to base design choices or certification decisions of systems on simulation results.

In a classical industrial environment, a system and its representations i.e. simulation models are often developed by different stakeholders with different objectives. These SUTs, along with other systems or their models, are then tested at different V&V scenarios by a test team. System designers, who design and develop systems, are often domain experts but do not necessarily have a multi-system end user perspective. On the other hand, testers or the simulation users are not domain experts but know the context under which a SUT will be used. Then, the challenge for the model developer, who is usually in between these two stakeholders, is how to develop models of the systems called *simulation model* in the context of system V&V. In using M&S as a means for such system V&V, the model developer needs to find and implement abstractions of the system being simulated with respect to the simulation requirements. However, this is often a challenging task since this fidelity requirement is seldom expressed even if the context of use is well known and often it is overlooked. In addition, as the systems are getting more complex so do the M&S activities. Even with the advent of powerful computing resources, the sheer complexity of phenomena to be modeled in addition to non-technical factors such as lack of rigorous and standardized process makes M&S activities challenging. There is also neither an agreed standard to define or measure this complexity of model, nor a methodology for model developer to choose it [Brooks,1996]. This motivates an important question of how to ensure adequate level of fidelity between a system and its simulation with respect to its V&V objectives all along the product development cycle? In order to answer to this question, it is important to study the current ‘as-is’ M&S process which is briefly presented in the following section.

1.1 MEASURED FIDELITY APPROACH

The current practices with regard to simulation fidelity in general, and Airbus in particular, is the conventional bottom up approach process. In this process, a simulation model is developed independent of the context under which it will be used and the fidelity is only measured *post priori* the experiment i.e. simulation. This approach, also called as the measured fidelity approach, either results in over fidelity i.e. too many unnecessary details in the model for the scenario being tested or under fidelity i.e. too little details resulting in costly rework and thereby increasing the cost and time of the overall system V&V process. In addition, this approach necessitates the knowledge of executability of the V&V plan on the means of simulation and the confidence of the results. However, these fidelity requirements (expected capabilities, tolerances etc.) are not explicitly represented in the system V&V plan and fidelity assessment is relied upon by traditional but arduous method based on expert review, heuristics and past experience.

This approach is illustrated in the figure 1.1 below and it can be seen that the simulation operation domain i.e. V&V plan and SUT is not taken into account in the simulation design domain i.e. simulation specification and fidelity is only measured at the end with respect to the real system's behaviour. Such an approach of measuring the simulation fidelity at the end is called measured fidelity [Ponnusamy,2014].

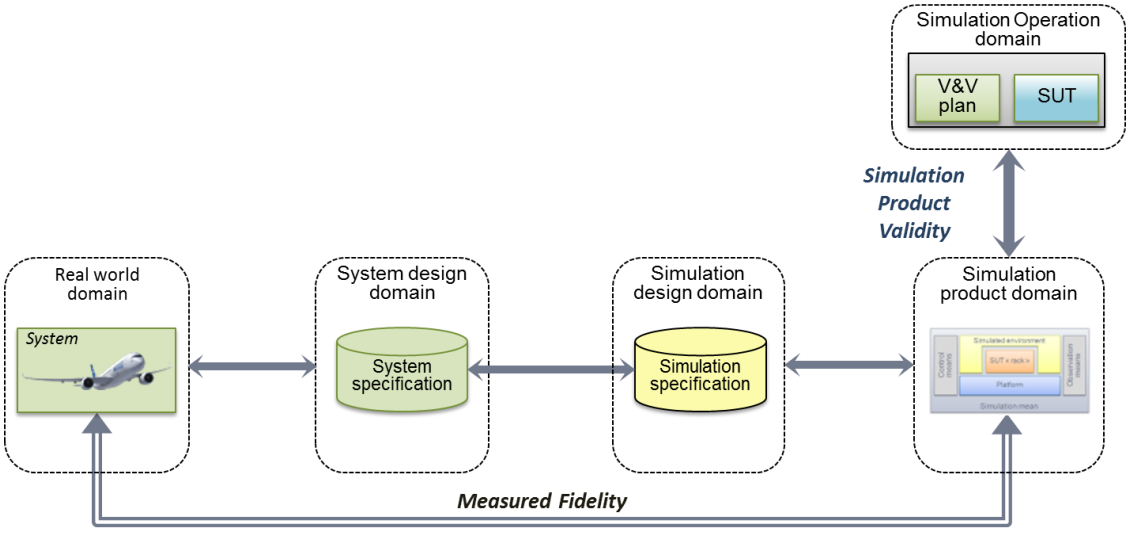


Figure 1.1: Measured Fidelity Approach

The problems of fidelity could be mitigated by explicitly taking into account the context of usage i.e. simulation operation domain into the simulation specification i.e. simulation design domain and this 'designed fidelity' approach is briefly discussed in the next section. A brief overview of the various challenges in the measured fidelity approach is further elaborated in section 2.2 in the context of the need for a designed fidelity approach.

1.2 DESIGNED FIDELITY APPROACH

A paradigm shift to a design fidelity approach [Ponnusamy,2014] where the modeling process is driven by the associated fidelity and validity requirements motivates the following questions.

1. How to assess the distance between a system and its simulation in general and with respect to its V&V objectives in particular?
2. Regarding the V&V objectives, what are the fidelity requirements on the means of simulation?
3. How to develop simulation models with respect to fidelity requirements?
4. How to develop a consistent approach to evaluate fidelity of simulation models along the product development chain?

It may be seen that realization of such an approach will help improve the level of confidence in the simulation results for system V&V and help better utilization of simulation resources by selecting the best resource according to test objectives. Identification of such a consistent and continuous way to improve simulation products will help improving product life cycle quality while controlling their cost and mitigating risk. However, this designed fidelity approach, which is

essentially the distance between the system specification and the simulation specification in figure 1.1, entails the following challenges which are broadly classified into three categories, namely,

1. Capture fidelity : How to capture the fidelity needs from the end user?
2. Manage fidelity : How to build the simulation specification according to the captured needs and the current system knowledge i.e. system specification?
3. Implement fidelity : How to ensure a consistent implementation?

The challenges are illustrated on the overall simulation product development process in the figure 1.2.

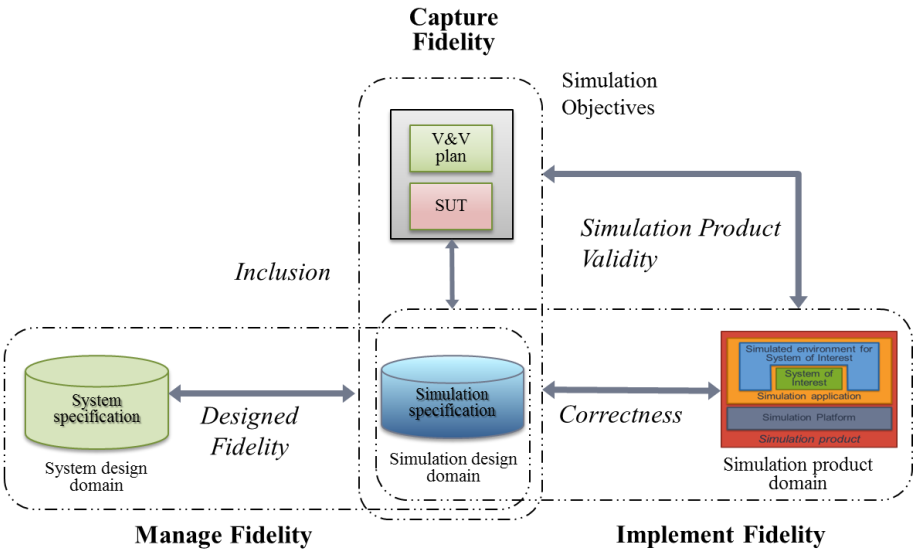


Figure 1.2: Fidelity Challenges

It may be noted that the critical challenge is on the capture and manage fidelity aspects since implement fidelity is arguably a verification i.e. correctness problem. In addition, the inclusion of simulation objectives into the simulation design domain with respect to the system being modeled i.e. system specification could also be seen. In this context, let us introduce our thesis objectives in the next section.

2. OBJECTIVES

The objective in the context of designed fidelity approach in *capturing* and *managing* fidelity is briefly given as follows,

- *Define a method to capture the fidelity needs of simulations intended to be used for the V&V of avionic systems (subclass of cyber physic systems) from design phase to final product.*

- *Propose a method to monitor the required fidelity level through the assessment of the validity of a simulation.*

2.1 SCOPE OF STUDY

The scope of the study has been limited to the design and product V&V of avionic systems through simulation. The study is focused on capturing fidelity requirements and implementing it through different modeling abstractions for such systems. The fidelity resulting from implementation aspects such as model of computation, hardware or software or model output perception such as visual displays has not been studied.

In addition, in the industrial context, V&V processes are only applicable to technical product requirements i.e. requirements having a direct impact on the fit, form or function of the product. The V&V of process or program requirements are covered by process assurance and project management activities respectively and are thus outside the scope of the current study.

In the following sections, the key objective of the study in capturing and managing fidelity [Ponnusamy,2015] is briefly discussed.

2.2 CAPTURING & MANAGING FIDELITY: NEED FOR A UNIFIED APPROACH

In a top down approach to M&S, owing to the fact that most of the models are rigorously verified but seldom validated, the onus must be on inclusion of validity objectives a priori in model building process as discussed in section 1.2. In this context, the first logical step would be to capture the fidelity aspects from the perspectives of simulation user or test team (simulation operation domain) and system designer (system design domain) i.e. fidelity requirements and fidelity capabilities respectively, to build a simulation specification (simulation design domain) with adequate fidelity. Fidelity requirements, according to Roza in [Roza,1999], is a formal description of the level of realism a model or simulation must display in order to achieve or to fulfill the needs and objectives of the user of the model or simulation. Similarly, the design decisions employed by the system designer to build the system specifications have to be considered which gives a measure of available fidelity i.e. capability. Once the fidelity is captured in terms of its requirements, the next step is to manage or assess this requirements vis à vis the capabilities via abstractions and the inclusion relations introduced in section 3.4 of chapter II between them. In this top down approach, simulation developer has to take design decisions as to what are the *possible abstractions* of the system specification with respect to the *fidelity requirement*? However, as remarked in [Brooks,1996], there exists no agreed standard or a guideline to choose this level of model complexity owing to the innate nature of problem in quantifying this complexity i.e. abstraction level vis à vis requirements.

This capture of requirements and development of models for cyber physical systems resulting from the confluence of control, communication and computing paradigms [Clark,2013] is an active research area as such systems are becoming ubiquitous especially in transportation domain such as in avionic systems. In V&V of such systems, there is a problem of heterogeneity due to different modeling formalisms used by different stakeholders leading to interoperability issues particularly during the model integration phase. In addition, the complexity of the process is higher, especially in an industrial context of simulation product development the modeling phase involves requirement collection, conceptual modeling, model formulation, model construction, assembly and deployment on the platform. Then the simulation phase involves experimentation according to V&V plan, data collection, analysis and conclusion. All such activities involve multiple levels of abstraction, stakeholders, formalisms and tools. In particular, the system specification (system

design domain) and/or simulation requirements (simulation operation domain) are not only at different levels of abstraction (design vs operational) but may also be expressed formally (e.g.: models), informally (e.g.: text) or a combination of both. Such complexities give raise to two key challenges for the model developer, namely, how to define requirements, called as Model Requirements (MR), from informal system specification and scenario description? and how to rigourously specify model behaviour based on this MR? A single approach to tackle this complexity is neither feasible nor practical and a rigorous multi modal system engineering approach is needed.

In this model based approach, effectiveness largely depends on the degree to which design concerns captured in the different abstraction layers by different stakeholders are orthogonal, i.e. how much the design decisions in the different layers are independent [Clark,2013]. Such a multi modal or multi view modeling approach has been widely discussed in terms of reasoning, functional modeling, qualitative modeling and visuo-spatial reasoning etc. [Fishwick,1993]. However, such studies are discussed mostly in modeling perspective and the problem of fidelity is not explicitly addressed. The unified approach presented in this thesis attempts to leverage the flexibility of semi-formal approaches and rigor of formal approaches to address the problems of complexity in the designed fidelity approach [Ponnusamy,2014]. In particular, the semi-formal approach concerns the system design and simulation requirements expressed in an informal context such as natural language texts whereas the formal approach concerns the same knowledge expressed through behavioural models. In reality, a model developer has to deal with both the formal and informal system design and simulation requirements to build a simulation model and it is important to provide the practicing model developer a perspective and mechanism to build these models with adequate fidelity. The two perspectives are necessarily based on the level of abstraction in the M&S process, i.e. capture and manage fidelity, both informally (qualitatively) and formally (quantitatively). These two perspectives are briefly presented in the following sections.

2.2.1 Semi-formal Perspective

The semi-formal perspective addresses the first or top level challenge of how to define MR from informal system specification and scenario description? It essentially concerns only the structure of the dynamics which is usually expressed informally in natural language texts, and not their quantifiable effect. In other words, semi-formal perspective deals with the different levels of abstraction resulting in a structure of the system dynamics or dynamics itself albeit at higher abstraction level. This top level perspective to M&S is equally important in understanding and explaining complex systems, as the current languages of systems engineering are usually informal (text and pictures) but seldom formal or even semi-formal (rigorous domain-specific languages).

In system engineering, especially in an industrial context, it is known that each component systems are developed by different multidisciplinary teams often working transversely and transnationally. These component systems usually interact with each other to perform, for example, a Multi System Function (MSF) in an integrated system. In the M&S of such complex systems, one of the key challenges is the lack of common understanding between the stakeholders, semantic inconsistency, and interoperability [Benjamin,2009]. For example, '*calculate and display aircraft position*' function is performed by Global Positioning System (GPS) and inertial data system which are communicated to Cockpit Display System (CDS) to inform the pilot. However, from this

informal textual description, identification of the functional contribution of each such system to MSF, its composition, interaction etc. could be difficult and this equally true for identification of other perspectives on the system.

The designed fidelity approach, introduced in section 1.2, necessitates collection of such knowledge about the system to be modeled and scenarios under which it will be operated respectively. This is then used to build the Model Requirements (MR) incorporating only the essential elements needed for the test which will then be used to develop a Model Specification (MS). However, owing to the complexity of different domains of knowledge involved which are often implicit and incomplete, it is a tedious task to define this MR manually. This is compounded due to the lack of a consistent derivation of low level V&V requirements from high level V&V objectives. The requirements traceability between these two domains is seldom one-to-one and the inclusions of low level requirements in the high level requirements are traditionally managed by heuristics, domain expertise, margins and experience. In addition to this lack of standardization and incompleteness of the domain knowledge, there exists no standard method to exploit or reuse its contents. This is usually done manually through stakeholder expertise and document review which is not only cumbersome but also time consuming and often redundant. These challenges in simulation model development necessitate a Model Based System Engineering (MBSE) approach which enables a common understanding by making domain assumptions explicit and separate domain knowledge from the operational knowledge [Noy,2001]. Such a semi-formal approach must be flexible enough to accommodate multiple viewpoints on the system which are often interrelated and at the same time be rigorous enough to identify incompleteness or inconsistencies between them. In addition, it must be amenable for exploitation through some query mechanisms, archival and must be scalable with respect to the domain knowledge and user implementation complexity.

2.2.2 Formal Perspective

The semi-formal perspective has limitations in capturing the dynamics of (reactive) systems and a fidelity approach will only be complete if the fidelity requirements usually expressed as a distance notion, e.g.: tolerances over desired behaviour, are adequately captured in modeling. The formal perspective, thus addresses second or low level question of how to rigourously specify a model's behaviour with respect to the system it represents? In general, in a classical or even a MBSE modeling approach, as remarked by Cowder et al [Cowder,2003], most of the design activities, around 90% in some cases, are based on the variants of the existing designs. This is true in V&V activities too where existing models are often reused to build a more complex but variant models of environmental systems of the SUT. In certain cases, models of such systems called design models might be available but could not be used due to practical constraints on resources, platform limitations and compositional complexity. However, as remarked in section 1.1, such models are developed independent of their end-use fidelity requirements with no formally i.e. mathematically rigorous, guaranteed bounds on their behaviours especially after composition, resulting in behavioural fidelity issues only to be discovered at the later stages. In addition, with current ad-hoc methods of model development, a model is developed as 'fidle' as possible i.e. as detailed as possible hoping it would cater to as many test scenarios as possible during the V&V activity. This is clearly a sub-optimal process especially in the context of changing specifications and scenarios. On the other hand, questions on a model's fidelity for a scenario different from the one for which

the model is originally developed for are neither answered formally nor apriori to the actual simulation.

In order to mitigate such problems, a component based approach is needed which quantifies fidelity of simulation model component's behaviour with respect to *all* the system's behaviour, both globally and with respect to the V&V objectives. In addition, in such an approach a composition with other models should not result in a compositional complexity i.e. assuming fidelity distance of components M_1 and M_2 be ε_1 and ε_2 respectively with respect to their system specifications, when they are composed to form a third component, $M_1 \oplus M_2 = M_3$, then the fidelity of this resulting component, ε_3 must be bounded, $\varepsilon_3 \leq \varepsilon_1 + \varepsilon_2$ [Tripakis,2016]. This behavioural fidelity perspective implies that there must be formal i.e. mathematically rigorous way to quantify the simulation model with respect to the system being modeled.

Formal Methods are descriptive notations and analytical methods used to construct, develop and reason about mathematical models of system behaviour. A *formal method* is a *formal analysis* carried out on a *formal model,* a model defined using a formal notation [Tiwari,2003]. A formal notation is a notation having a precise, unambiguous, mathematically defined syntax and semantics. Formal method uses mathematical reasoning to guarantee that properties are always satisfied by a formal model. There are various techniques available such as deductive techniques (theorem proving) [Duffy,1991], model checking [Clarke,2000], and abstract interpretation [Cousot,1992]. Since formal methods possess sound mathematical basis, a false assertion is not possible. In general, a formal verification approach intends to prove that the system satisfies (or not) a given property and this verification problem could be formalized as a reachability analysis problem in a finite labeled transition system which includes the problems of proving safety, liveness etc. [Vardi,2009]. This is a method to show that the system defined by a computational model such as automata, satisfies the desired properties, i.e. all the behaviours generated by the system are those accepted by the specification defined by a specification language such as temporal logic [Pnueli,1977]. Though fidelity per se does not intend to (dis)prove a property but only is a measure of closeness to system being modeled, the principles of formal verification such as reachability analysis could be used to rigorously quantify the fidelity [Ponnusamy,2015]. In other words, classical formal verification techniques usually done to demonstrate morphism [Ziegler,2000] between specification and implementation is extended to show the morphism between the system specification and its abstraction (simulation model) for the purpose of quantifying the degree of similarity between them i.e. fidelity distance.

The formal approach, in general, helps in standardizing and automatizing V&V activities and has been increasingly used in the field of software and hardware design [Clarke,2000]. Though they are widely used in software verification and to some extent in system design [Alur,2015], application of such approach in simulation design domain especially in the context of fidelity has not been done adequately. The benefit of using classical formal method, especially in the early design verification phase has been widely discussed in literature [Ben,2003],[Clarke,2000],[OSKI]. In the case of V&V by simulation, the exponential growth of verification effort with design size could be greatly alleviated by using formal tools along classical simulation especially before model composition and integration in the simulation platform.

It is also to be noted that when a formal model is created from an informal knowledge usually expressed at higher abstraction levels such as in a semi-formal perspective discussed in section 2.2.1 to perform a formal analysis, it needs to be ensured that whatever is proved about the formal model also applies to what is modeled. Then review or analysis should be used to demonstrate that

the formal statement is a conservative representation of the informal requirement. Thus the two perspectives could be seen to be complimentary in capturing and assessing fidelity at different layers of abstraction to ensure overall fidelity of the resulting simulation product.

2.3 KEY BENEFITS

The intended key benefits of the unified approach proposed in this thesis based on the formal and semi-formal perspective to the M&S stakeholders are briefly presented below.

Model Developer
- Help find the set of allowable abstractions for model to be developed with respect to V&V objectives.
- Provides synthetic & accurate descriptions of simulation end user's intention in order to implement only relevant details.
- Use templates and pattern to describe the designed fidelity and apply abstraction.

Simulation User
- Help find the model developed by design abstraction that fit their needs in terms of V&V objectives.
- Use templates to describe their intention.
- Increase confidence in the simulation results.

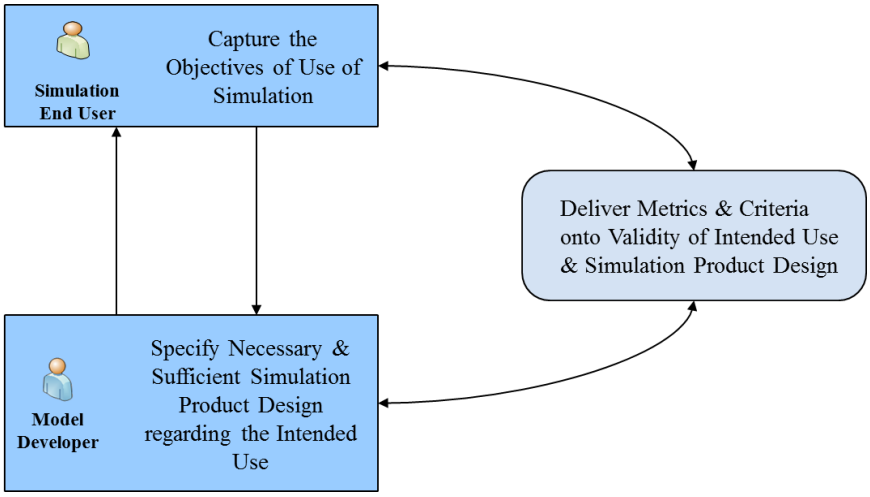


Figure 2.1: Proposed Approach

System designer
- Help formalize the system design in a standardized, interchangeable template.

The proposed approach to build the designed fidelity progressively all along the life of simulation products is illustrated in figure 2.1.

3. THESIS CONTRIBUTIONS

The overall contribution of the thesis towards an unified approach to the fidelity problem is essentially threefold, namely,

1. Formalization of fidelity problem as an inclusion problem, detailed further in section 3.4 of chapter II, in the established M&S notions of the experimental frame [Zeigler,2000].
2. A domain model approach in the semi-formal context to assess and define the simulation fidelity qualitatively.
3. A behavioural metric approach in the formal context to assess and define the simulation fidelity quantitatively.

In addition, a process oriented view for each of these approaches to simulation fidelity has been discussed in an industrial context. The major contribution of the thesis is the proposition of a domain model approach i.e. semi-formal and behavioural metric approach i.e. formal to this inclusion problem which is briefly presented in the following sections. The associated publications could be seen in the Publications section of this thesis.

3.1 DOMAIN MODEL APPROACH

The designed fidelity approach, in the semi-formal context, necessitates the collection of knowledge about the system to be modeled and scenarios under which it will be operated which normally involves interaction between system designers, testers and model developers. However, owing to the complexity of different domains of knowledge involved which are usually at different levels of abstraction, it is a tedious task to define the essential elements to be modeled for a given test. In order to alleviate this complexity and standardize the knowledge which could then be exploited, we propose a domain model approach based on ontologies in chapter III. This domain model is essentially a 'knowledge template' i.e. an ontology which captures the system design and test scenario knowledge into pre-defined, standardized concepts and relationships [Ponnusamy,2016],[Thebault,2015]. This approach has been chosen due to the flexibility in expressing different domain knowledge in a succinct and standard form through standardized language of OWL [OWL], in tools such as Protégé [Protégé] with query [SPARQL] and reasoning [Grosof,2003] capabilities. Ontologies, in general, have been widely used to tackle complexity in the field of artificial intelligence, semantic web, bioinformatics, information science etc. by standardizing and organizing domain knowledge. Though ontologies in the M&S were addressed in literature, albeit at high level, for example in [Fishwick,2004], [Oren,2014], [Kezadri,2010], they were not explored sufficiently in a MBSE context for simulation model development. In addition, a holistic application of ontology, especially to the problem of simulation fidelity, by leveraging the flexibility, scalability, reasoning and query capabilities of ontologies has not been studied adequately to the best of our knowledge.

In the teleological modeling of complex engineering systems, different ontologies such as Functional representation [Chandrasekaran,1993], Structure-Behaviour-Function [Gero,2004] have been discussed in literature. The classical Structure, Behaviour, Function (SBF) framework is one of the widely used and mature ontology to specify the system's function and the causal processes that result in them at multiple layers of abstraction [Goel,2009]. In this thesis, we extend this classical SBF ontology to the domain of simulation in section 4 of chapter III and introduce

additional notions of interface (I) and Operation (O) to describe interconnected system with different modes of operation [Ponnusamy,2016]. In the Operation ontology, we have proposed the concept of Operating Mode, based on the classical formalisms such as modechart [Jahanian,1994] and mode automata [Maraninchi,1998], but is believed to be more amenable to describe a system's modes of operation at higher levels of abstraction. In addition, our domain model comprises of different generic concepts (e.g.: Datatype), industry or domain specific concepts (e.g.: Criticality Level, Airbus internal standards) and among others, a test ontology to capture the test scenario knowledge. The domain model has been constructed based on the academic and industrial state of art such as SBF framework, common MBSE approaches such as SysML [SysML,2006] CAPELLA [Roques,2016], Airbus internal M&S processes and standards, interviews and discussion with the V&V stakeholders which is further elaborated in section 1 of annex.

In an industrial context, it is important to illustrate how such a domain model approach improves the existing processes with minimal disruptions. A process to utilize this domain model to define requirements on a simulation model is proposed in section 5 of chapter III. In particular, a three step process of capturing the system and test scenario knowledge through the domain model, using reasoning approach to check consistency and using queries to extract information is proposed in this thesis. The domain model is implemented in the Protégé tool [Protégé] and different query mechanisms were defined to verify and extract information at mulitple layers of abstraction to build a MR. The application of this approach is demonstrated with a real industrial case study of the aircraft Nacelle Anti-Ice System (NAIS) in section 1 of chapter V . The results from the case study discussed in section 1.2 and 1.3 of chapter V were highly promising especially in the context of knowledge standardization, reuse, archival and query capabilites. The challenges in practical implementation and outlook were discussed in section 6,7 of chapter III and section 1 of chapter VI.

In addition, we have also shown that the output of this MR construction approach could be used as an input to the MS construction process by automatically selecting a consistent model from a model library based on the recursive procedure proposed by Levy et al [Levy,1997]. In section 6 of chapter III we have given an operational perspective of this entire MR and MS construction process based on the existing industrial processes.

3.2 BEHAVIOURAL METRIC APPROACH

The designed fidelity approach, in the formal context, necessitates a component based design approach for developing a simpler representation of the constituent systems wherein each component must be adequately representative enough to perform V&V on the SUT. The key question in this approach is how to measure this fidelity i.e. how closely (or not) does the model simulate i.e. 'mimic' the system behavior [Ponnusamy,2016]? However, quantifying fidelity, especially in a formal manner, is often a challenging task since it requires real system behaviour to compare against the model behaviour. This post-priori measurement of fidelity happens often at detrimental cost due to over or under specification of models as mentioned in section 1.1. Instead, this fidelity needs to be measured a priori both globally and locally i.e. with respect to V&V objectives before integration with other models, SUT and deploying on the simulation platform as discussed in section 2.2.2. In order to formally quantify this fidelity between a component system model and a simulation model, we propose a formal approach in chapter IV which assigns a metric

to quantify this degree of similarity with respect to all possible or a subset of behaviours of the system based on the notions of game theory and formal verification principles.

Our approach is based on the quantitative extensions of classical simulation relations [Milner,1989] proposed in the context of discrete systems [Henzinger,2013],[Chatterjee,2015] and continuous systems in [Girard,2007], [Pappas,2003]. In the former, a distance notion based on two player game for automata [Cerny,2010] and interface automata [Cerny,2014] gives a transition-wise or path-wise distance in the context of implementation, coverage and robustness. In the later, an approximate bisimulation relation essentially giving a global error bound i.e. maximum degree of dissimilarity between two models at a given time instant is proposed. This is then formally verified by geometric over approximation of the reachability set through zonotopes [Girard,2005], ellipsoids etc. However, in the field of (discrete) simulation, such global bound is over-conservative since according to a scenario a model might still be valid locally despite its global error [Ponnusamy,2016]. Similarly, the distance notion proposed for untimed discrete systems in [Cerny,2010], [Cerny,2014] and timed discrete system [Chatterjee,2015] concerns only fidelity distance evaluated transition-wise for a particular path. This may not be adequate since not all such possible paths are explored. In other words, not all scenarios i.e. input combinations are considered. This necessitates finding such distance bounds on all possible paths evaluated over a positive real valued distance function. This generation of fidelity distance between *every* possible path of the system and simulation model for every possible input is also called as a quantitative reachability graph. An analysis of this graph will yield further insight into the adequacy of abstraction globally or with respect to V&V objectives. However, to the best of our knowledge, such a mechanism to quantify this distance for all possible inputs i.e. a superset of test scenarios between any two given models has neither been proposed nor been implemented especially in a fidelity context.

The behavioural fidelity metric approach is proposed for discrete systems in sections 5 of chapter IV whereas some theoretical results for linear continuous systems [Ponnusamy,2016] can be found in the annex. In the discrete systems case, our approach concerns both the open i.e. reactive to its environment and closed i.e. non-reactive to its environment modeled by automata and interfaces respectively [Alfaro,2003]. This is important since behavioural fidelity problem arise from a simulation model's internal structure (modeled as automata) as well as its environmental assumptions/guarantees (modeled as interfaces) and it is important to study the quantitative reachability approach for both such complementary paradigms. In particular, in the case of closed timed systems modeled as timed automata, we have proposed a turn based semantics specifically in the context of fidelity for the quantitative reachability graph generation in section 5.2.1 of chapter IV.

We have modeled this game based formal fidelity quantification for all such different class of systems in (Timed) Petrinet formalism. (Timed) Petrinets, is an extension of classical Petrinet formalism [Peterson,1981] with firing time for the events and an extension of it with data handling called Time Transition Systems [Berthomieu,2014] is used in our approach. The token based formalism of the Petrinets is amenable to model such turn based games which is explained in detail in chapter IV. In addition, the availability of state of the art and in house developed Petrinet analyzer tool called TINA [Berthomieu,2004] with its graphical editor and reachability generation capabilities renders it an attractive choice for our implementation which is discussed further in section 6 of chapter IV. Since the quantitative reachability graph generated by TINA is in textual form, we have also developed a parser to rebuild all the paths which will then be used to perform some analytics such as finding a path with least or maximum distance, distribution of fidelity

distance etc. In addition, our Petrinet implementation allows incorporating different fidelity distance metrics such as transition weighted error, absolute error, non-weighted error etc. according to the user need. The demonstration of our approach and implementation were presented in section 2 of chapter V for an (un)timed model of a buffer system, untimed interface model of NAIS with different types of abstractions. Despite the challenges of scalability of our explicit enumeration approach, initial results indicate even for a limited reachability exploration (for example $<10^6$ paths), our method gives valuable insights on the distribution of fidelity which could be then be used for deployment, model repair or simply archival.

In summary, despite the seemingly orthogonal solutions proposed in this thesis, namely domain model approach and behavioural fidelity metric approach, they both serve their purpose in improving fidelity albeit at different levels of abstraction. The domain model approach standardizes and exploits the often informal domain knowledge to build a semi-formal MR with adequate levels of fidelity. This would then serve as a baseline upon which the model developer chooses the model existing variants whose level of fitness for a given purpose i.e. to validate the SUT using simulation is given by our formal approach. Another key benefit of this unified approach is these two methods can either be used independent of each other or complementary to each other depending on the prevailing fidelity issues and user's need as seen from the thesis roadmap section. The plan of the thesis is given in the next section.

4. THESIS PLAN

The key challenge of the designed fidelity approach is to develop a mechanism to collect the fidelity requirements and then to evaluate the model against these requirements. This thesis is broadly focused on identifying the challenges in developing such a mechanism and solutions for mitigating them at multiple levels of abstractions followed by its demonstration on application case studies. In the unified fidelity framework context, the semi-formal and formal approaches are detailed in chapter III and chapter IV respectively with their application case studies presented together in chapter V. The intended key benefits of the approach listed above are evaluated in each such chapter and relevant conclusions are drawn which are further discussed in chapter VI. In this context, the chapters of the thesis are organized as follows,

Chapter II : The second chapter addresses the background and problem formulation of our designed fidelity approach through the established theory of modeling and simulation framework of experimental frame formalism and inclusion relations between them.

Chapter III : This chapter presents the semi-formal approach based on the principles of ontologies in building a domain model to capture, formalize and evaluate the knowledge of simulation fidelity requirements with respect to the system specification to build high level simulation specification with sufficient fidelity.

Chapter IV : The formal approach based on the principles of formal verification and game theory to quantify fidelity of simulation models with respect to their system specifications for different class of systems is presented in this chapter.

Chapter V : The results of applying the semi-formal and formal approaches to the case studies were detailed in this chapter.

Chapter VI : The last chapter focuses on the overall and specific outlook on our unified approach, challenges ahead, axes of future work and conclusion.

In addition, associated information not detailed in the aforesaid chapters such as pseudo code, methodology implementation etc. could be found in the annex. The bibliography section contains list of references used in the study.

CHAPTER II

EXPERIMENTAL FRAME & INCLUSIONS

In this chapter, the designed fidelity approach is discussed in the context of a unified perspective in building a simulation Experimental Frame (EF) through inclusion relations. In order to better understand the problem formulation especially in the context of chapter I, a brief overview of the context and background is presented in the following sections.

1. BACKGROUND

In this section, some generic definitions and background information on system V&V, M&S and the associated fidelity notions from the industry and academia are briefly discussed.

1.1 SYSTEMS VERIFICATION & VALIDATION

According to Brian Gaines, a *system* is what is distinguished as a system which essentially means that to distinguish some entity as being system is a necessary and sufficient condition for its being as a system [Gaines,1979]. A system is usually characterised by what belongs to it and what it doesn't belongs to it. The systems theory focuses on arrangement and interdependent relationships between the components of a complex system and distinguishes between a system's behaviour and structure. Such a definition is echoed by the Airbus definition of system as abstract entities, introduced by a standardization authority (ATA 100) defined as a set of equipment [FAA,2002]. In addition, according to INCOSE, System engineering, a multidisciplinary field, is defined as an iterative process of top down synthesis, development, and operation of a real world system that satisfies, in a near optimal manner, the full range of requirements for the system [INCOSE,2011]. More specifically, IEEE standard 1362 [IEEE,2007] defines a system as a collection of interacting components organized to accomplish a specific function or set of functions within a specific environment.

A system can be either closed or open depending on its reactivity to its environment. A key property of a *reactive* system which interacts with its environment is it can be controlled through variables called *input* that are generated from the environment to influence the system and observed through *outputs* which are variables generated by the system and influence its environment. Such an interpretation leads to a concise definition of system being a source of data. The process of extracting data from such a source i.e. system by exerting with input is called an *experiment*. An experiment could either be real e.g. flight tests or virtual e.g. simulation to perform verification or validation activity on the system under test.

In this context, **Validity** of a system is measured through *validation* activities *(*to answer *'*did I develop the right product?'*)* and **Correctness** through *verification* activities *(*to answer *'*did I develop the product right?'*)* [Brade,2004]. In addition, validity has another perspective with respect to product requirement, called requirement validity (to answer *did I ask the right questions*?). Hence product validity is given by its requirement validity and correctness. These V&V activities

are carried out according to some V&V plan formulated usually by the system designers or architects. Such an activity through an experiment is usually comprised of different sub-experiments called *test scenarios*, usually supplied by the test team, which essentially describes what is expected of an experiment and how it is done on the system built by the system designers.

1.2 MODELING & SIMULATION

In general, the process of describing the system as a model is called ***modeling*** and the process of experimenting the model is called ***simulation*****.** A ***model*** is essentially an abstract representation containing the essential structure of some object or event in the real world. More precisely, according to IEEE 610.12-1990 standard [IEEE,1990], a model is formally defined as an approximation, representation, or idealization of selected aspects of the structure, behaviour, operation, or other characteristics of a real-world process, concept, or system. Thus modeling is the process of generating abstract, conceptual, graphical or mathematical models of a real system whereas simulation is the imitation of the operation of real-world process or system over time [Cellier,1991]. A simulation generates an artificial history of the system behaviour and upon observation of that observation history, design decisions or analysis could be made for the real system. Hence, the simulation could be used as an analysis tool for predicting the effect of changes or as a design tool to predicate the performance of new system [Balci,1997].

Marvin Minsky defines a *Model* for a system as anything to which experiment can be applied in order to answer questions about the system [Cellier,1991]. Thus a model is always related to the tuple of system, S and experiment, E [Zeigler,2000]. Hence notations like validity, fidelity etc. of a model must be addressed in association with the system it represents and experiment which it intended to address.

1.3 FIDELITY

A brief survey on the notion of fidelity and its manifestations with respect to modeling and simulation are given in the following sections.

1.3.1 Definition

Fidelity is often used in different contexts both in scientific and non-scientific fields alike, however, it would in general, as a classical definition of Oxford dictionary imply, the degree of exactness with which something is copied or reproduced. A myriad of interpretations of fidelity, especially in the M&S community leads to inconsistency in the Verification, Validation & Accreditation (VV&A) activities and this necessitates a precise notion of this generic term. In the present thesis fidelity is defined as a notion of 'distance' to reality and by assigning a metric this distance could be measured quantitatively. Fidelity, henceforth, is defined as *the* ***distance*** *from the* ***simulation of a system*** *to the* ***simulated system.*** This definition is akin to widely accepted definitions such as the US Department of Defence (DoD) stating fidelity as the accuracy of the representation when compared to the real world. Simulation Interoperability Standards Organization (SISO) fidelity Implementation Study Group (ISG) formally defines fidelity in [SISO,2013] as

The degree to which a model or simulation reproduces the state and behaviour of a real world object or the perception of a real world object, feature, condition, or chosen standard in a measurable or perceivable manner.

In stating that fidelity should generally be described with respect to the measures, standards or perceptions used in assessing or stating it, SISO further defines it by,

The methods, metrics, and descriptions of models or simulations used to compare those models or simulations to their real world referents or to other simulations in such terms as accuracy, scope, resolution, level of detail, level of abstraction and repeatability.

Fidelity can thus characterize the representations of a model, a simulation, the data used by a simulation (e.g., input, characteristic or parametric). Each of these fidelity types has different implications for the applications that employ these representations. In addition, SISO emphasises the referent i.e. simulation fidelity requirements must be carefully defined in terms of how much is to be simulated (i.e., entities and characteristics) and what interactions are involved (i.e., relationships between entities in the referent). SISO identifies a key obstacle in acceptability of M&S methods as a tool to make design decision for real world problems is defining a fidelity metric which measures the simulation behaviour. In defining fidelity as a measure of distance to reality, abstractions in modeling could be seen as the cause of this distance. An abstraction level in complex engineering simulations such as for aircraft is a crucial factor in influencing resources deployed to use simulation as a means in system design and development. An incorrect or inconsistent choice of abstraction level of model will result in prohibitory complexity in overall simulation process and thereby its validity and fidelity. In the next section, the notions of fidelity and validity are discussed with respect to this modeling abstractions.

1.3.2 Fidelity, Validity & Abstraction

A unified approach to fidelity was done by Roza in [Roza,2004] where a mathematical formulation of fidelity and the fundamental concepts underlying its characterization and measurement were established, parts of which are comprised in SISO ISG report [SISO,2013]. The study establishes that Fidelity quantification and qualification *doesn't equate* to validity of simulation (Theorem 6, [Roza,2004]). It follows from the preceding result that fidelity is an intrinsic or absolute property of any model or simulation characterizing its degree of realism (Theorem 3, [Roza,2004]) and an absolute metric of fidelity could never be established owing to the inherent levels of uncertainty (Theorem 1&2, [Roza,2004]). In addition, Roza (Theorem 7, [Roza,2004]) and Zeigler [Zeigler,2000] states that model fidelity and simulation fidelity do not equate. Thus a formal metric relating the fidelity and validity of simulation becomes imperative and, this question of relation between the simulation behaviour and its objective could be addressed vis à vis the factors affecting the fidelity of simulation outcome with respect to the validity, namely, abstractions used in modeling. In framing a metric between the abstraction used in model development and its associated objective of use, a bridge could be made between the fidelity (degree of realism of model) and validity (degree of correspondence to objective of use). Brade et al [Brade,2004] emphasis this relation of fidelity with validity by defining fidelity as a measure to show that the model and its behaviour are a suitable representation of the real system and of its behaviour with respect to an objective of use of the M&S product. Since fidelity is one of the vital drivers in terms of development and deployment cost in simulation, an early quantification of fidelity helps in better simulation product development for system validity assessment.

1.3.3 Fidelity Classification & Metric

In the study by Roza [Roza,2004], fidelity has been classified into eight classes namely detail, resolution, accuracy, interaction, temporality, causality, precision and sensitivity. Fidelity was also classified as esoteric fidelity i.e. an ideal measure and practical fidelity with a proposition on a domain independent fidelity criteria evaluation. In evaluating this fidelity, quantitatively or qualitatively, the evaluation could be carried out within the ambit of these eight characterisations. In specifying the real world reference knowledge specification template, the real world was specified as a system of interacting subsystems in a hierarchical object oriented fashion.

Another classification of simulation fidelity from the end user perspective in terms of perceptive fidelity and behavioural fidelity was put forth by Robert Hays [Hays,1980] based on definitions of Kinkade, Wheaton, Farina et al. [Kinkade,1972]. The perceptive fidelity classification deals with end user experience of the simulator in the following terms:

Equipment fidelity (physical configuration) – e.g.: a flight simulator to evaluate flight control laws has gauges, dials, control sticks etc.

Environmental fidelity (duplication of environmental context) – e.g.: motion cues, dynamic representation of environment such as sky, ground etc.

Psychological fidelity – e.g.: level of psychological perception of the end user as realistic though it may turn out to be otherwise.

The behavioural fidelity also referred by the authors as task fidelity is akin to generic notion of simulation fidelity which is described in this thesis as a degree of representation of real world by the simulation. However, these definitions fell short of relating between the representation of real world and simulation objectives and its context of usage since a model is always developed with an objective behind. It may also be noted that these definitions are more to do with training simulators for mature systems rather than with simulators used in system design and development especially in the design verification phase of figure 1 of chapter I which this thesis is focussed on.

In addition to such classification, a metric on fidelity is useful to gauge the rigour of simulation models. The metric could either be qualitative (e.g.: low/medium/high) or quantitative [SISO,2013], though quantitative metrics are often overlooked, they could well be attributed to a related qualitative metric and vice versa. In practice, both qualitative and quantitative metrics are useful, since the former is amenable for subjective evaluation by human domain experts and the later for objective evaluation of some specific characteristics. However, even before ascribing a metric, qualitative or quantitative, we need a mechanism to improve the fidelity of models by choosing right level of abstraction which then can be compared against a given metric. In this thesis, our semi-formal approach, presented in chapter III, is used to qualitatively (yes/no) evaluate fidelity by answering *what are all the scenarios that can(not) be modelled based on system specifications?* Similarly, our formal approach, presented in chapter IV, is used to quantitatively (a real valued function) evaluate *what is the component model's fidelity with respect to the system specification?*

It may be noted that our designed fidelity approach is not intended to propose a metric per se, but to propose a mechanism to improve the fidelity of the overall process which can then be assessed with any given metric. Thus it is amenable for further extension or modification of the fidelity metric and in doing so the inclusion principles of neither the domain model approach nor the formal approach is expected to change. For example, current qualitative evaluation of domain

model approach could be extended quantitatively in future to answer what is the coverage of scenarios and what are its impacts on MR? What is the effect of abstraction on a scenario? etc. Similarly, our quantitative approach evaluates fidelity based on transition weighted error, further explained in chapter IV and this metric could be modified according to the user need without changing the underlying game based semantics.

A classification or definition of fidelity metric as discussed briefly in this section, though imperative in understanding the phenomenon better, may still be inadequate for the model development for V&V activities. This is predominantly due to their inability to explicitly address how to incorporate different such fidelity requirements through modeling abstractions to develop simulation model(s). Before discussing it, the concept of simulation product which is comprised of this set of simulation model(s), simulation platform, is briefly presented in the next section with some perspectives addressing the fidelity aspects of it. These perspectives are important to better understand the problems of measured fidelity approach presented in section 1.1 and our solution of designed fidelity approach presented in section 1.2 of chapter I.

2. PRELIMINARIES

In this ection, few preliminary definitions which will enable to better understad the problem formulation and context are presented.

2.1 SIMULATION PRODUCT

In this study, we define a *simulation product* as a simulation application i.e. a model or set of models, deployed on a simulation platform, and interfaced with the SUT as illustrated in figure 2.1. The models simulating the environment of SUT are shown in dark green, interface with SUT is shown in orange.

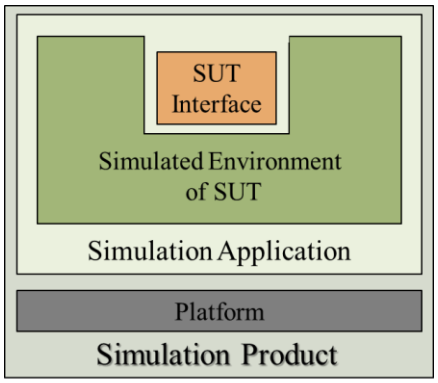


Figure 2.1: Simulation Product

The *simulation application* comprises a set of standard simulation models and associated configuration files which specify the connections between models, and their scheduling properties [Thebault,2015]. The *simulation platform* usually consists of an IT infrastructure and the simulation software. The platform schedules and monitors the execution of the models with respect to time constraints of logical or real time simulation. It enables communication between the models

and provides the end user with control and observation facilities to operate the simulation [Thebault,2014].

The simulation application development is based on the knowledge of the operational environment of the SUT, which is, in the avionics context, composed of equipment whose behaviour is governed by physical laws such as aircraft natural dynamics and other avionic systems. In order to enable such an architectural representation, an internal standard may exist in the industry to define a common understanding on how the simulator platform shall execute the simulation application.

Simulation products developed at Airbus are 'enabling products' which enables part of the complete life cycle of the aircraft in product development, testing and training. Simulation models represent all or part of aircraft system or equipment as well as the environment, in which the model is operating, and are used all along the aircraft development process, Integration, V&V and training purposes. The models are either developed by Airbus or requested, through contract, to suppliers and these activities are widely spread over different stakeholders. The V cycle for commercial aircrafts developed by Airbus with different phases of the product development and validation is illustrated in figure 2.2 [Airbus].

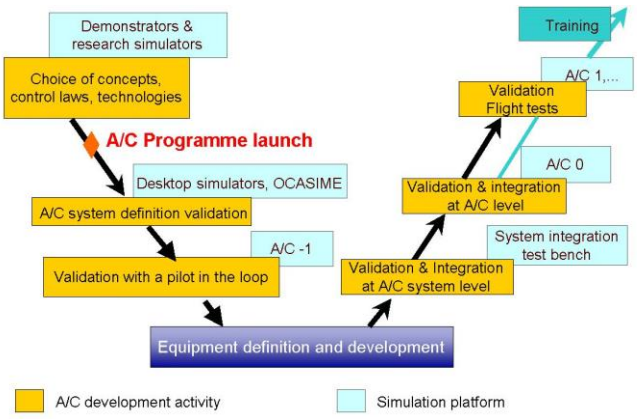


Figure 2.2: Airbus V Cycle [Airbus]

In addition, the correspondence with different simulation platforms such as system integration benches, functional integration benches and desktop simulators at each phase of the V cycle is also given. In the next section, two key perspectives on this simulation product, or more specifically simulation application, since our study does not concern platform or model of computation aspects, are presented.

2.1.1 Simulation Product: Twin Perspectives

In experimenting the model i.e. simulation, it is important to know and differentiate between the ability of the model defined by the simulation developer, and expectation on the model defined by the simulation end user. In general, there exist two perspectives on the simulation product [Albert,2009] namely,

SOU: *Simulation Objectives of Use* (SOU), the experimental frame that describes the way in which experimentation of the SUT will be performed. This specification will be paired with the SUT to

generate quantifiable expected results. The simulation user defines the SOU through some required abstractions on models, scenarios along with expected results and tolerances. The end user defines these expectations on the simulation product according to the V&V plan for the SUT. In this perspective, the simulation product is essentially a black box, with an assortment of configuration, software and hardware components. The SOU may be defined by a single or group of different simulation end users.

SDU: *Simulation Domain of Use* (SDU), the experimental frame that describes the usage domain of a simulation application, i.e. its properties and its limitations. The simulation developer defines this SDU through some implemented abstractions and therefore the usable scenarios of the simulation product. The SDU is the set of models which simulates the SUT environment to answer SOU questions on the SUT. In practice, the SDU is developed by different model developers and assembled by an integrator before deploying on the platform which in turn is developed by other stakeholders.

These two perspectives are given in the following figure 5.5. It may be seen that the simulation product i.e. SDU developed from its specification is evaluated against the SOU to assess the simulation product validity.

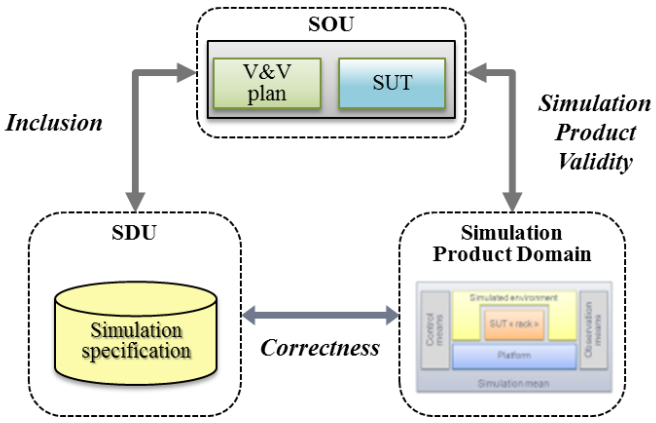


Figure 2.3: Simulation Product – SDU & SOU

Intuitively, a simulation product is deemed to have sufficient fidelity or simply the simulation product itself is valid, if its capabilities i.e. SDU includes expectations i.e. SOU in it. More specifically, in section 3, SDU and SOU are presented in the established M&S formalism of experimental frames and fidelity evaluated through verification of inclusion relations between these two experimental frames.

In the next section, current industrial practices of fidelity evaluation of such simulation product introduced in section 1.1 of chapter I is presented in detail.

2.2 SIMULATION PRODUCT DEVELOPMENT PROCESS

The simulation product development usually involves three stakeholders namely model developer, system designer and simulation user. The simulation user is usually the V&V task owner who defines the SOU derived in turn from the high level V&V objectives in terms of functional, non-functional and behavioural requirements. In practice, there is a model specialist who translates

simulation requirements and system specification into functional and behavioural requirements of the model to be built. The model developer builds the model based on this requirement using knowledge from existing library of abstractions. The built model assembled and then verified against their requirements by the simulation user according to the V&V plan. This is illustrated in figure 2.4 [Thebault,2015], where system V&V through simulation process is illustrated with associated stakeholders.

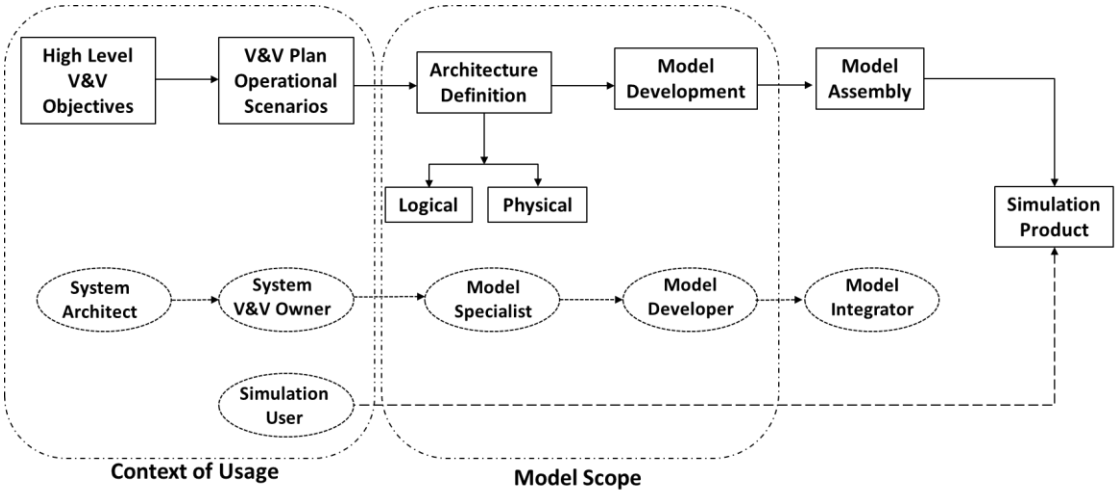


Figure 2.4: Simulation Product Development Overview

A simulation product needs to be updated continuously to follow each high level design change and also new simulation capabilities for V&V objectives to the end user. In general, this simulation application development process is performed by simulation platform teams, who consistently interact with the component system developers and simulation users. The simulation application development process is briefly illustrated in the following figure [Thebault,2014],

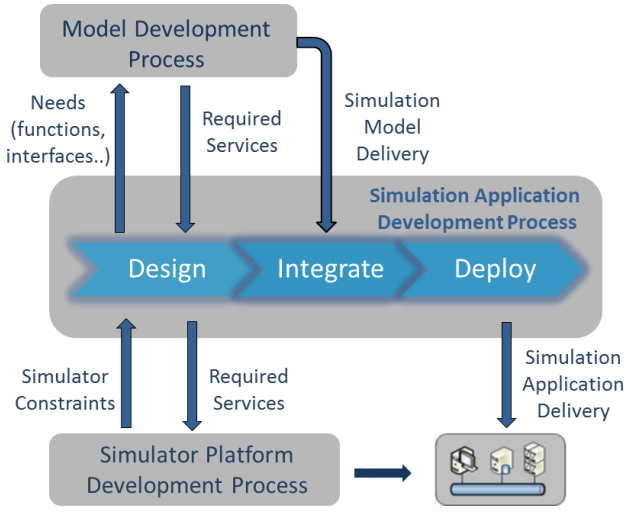


Figure 2.5: Simulation Product Development Process [Thebault,2014]

It can be seen that the process can be broadly classified into design, integrate and deploy. In the 'design' phase, functional and performance objectives of the simulation models are defined in addition to their interface definition. The second phase, 'integrate', is model assembly phase to ensure the consistency for its 'deployment' on the simulation platform. However, in practice, owing to the complex nature of this process, the context of usage is not always captured in accordance with modeling abstraction employed in the 'design' phase. Thus simulation product development is essentially an iterative process based on measured fidelity approach due to the challenges in complexity, methodology and continuous evolution of product requirements.

This iterative process often proves to be costly since a model might be too detailed i.e. over fidelity or too little detailed i.e. under-fidelity and this have implications in cost and time of the system development process. Thus, in order to mitigate these problems, the context of simulation product i.e. model must be taken explicitly into the 'design' phase. It may be noted that, despite the current approach of developing a model as detailed as possible, there exists no concept such as absolute validity of a model and any model, whatsoever its complexity may be, cannot satisfy all the possible requirements [Balci,1997]. In essence, as noted by Robinson [Robinson,1997], the objective of V&V activity is not to prove that the model is right, but prove that it is incorrect in sense that longer the model resists the notion of an incorrect behaviour, more the confidence. This interpretation has a direct relation with our notion of designed fidelity approach explained in next section where by relating the notion of fidelity i.e. distance to reality, with V&V plan i.e. SOU and model behaviour i.e. SDU, enough confidence in the M&S for V&V can be created or assured.

In the next section, the problem formulation is discussed in the established Theory of Modeling and Simulation framework [Zeigler,2000] which defines suitable notions of abstraction and validity onto the tuple of Model-Simulator-Experimental frame. A brief overview of the formalism and associated definitions are given, followed by a discussion on key perspectives and corresponding inclusion relations between them in the context of fidelity.

3. THEORY OF MODELING & SIMULATION FRAMEWORK

3.1 EXPERIMENTAL FRAME

In the context of studying a system (through simulation), the concept of EF [Zeigler,2000] is used to describe experimental scenarios under which the SUT will be used. An EF, in general, defines the controllability (input scenarios) and observability (expected outputs) means to stimulate and observe the SUT behaviour in addition to conditions of experimentation. This EF could be intuitively interpreted as the environment of the SUT supplying inputs and obtaining its outputs or alternatively SUT serves as the environment for EF. This composition of experimental frame with the SUT is illustrated in figure 3.1.

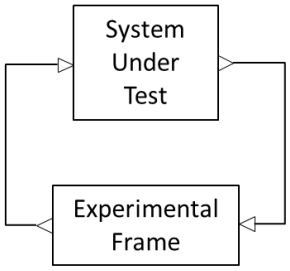


Figure 3.1: EF & SUT

In general, an EF could be made of systems or its abstractions i.e. models. We call the former as called system experimental frame (EF_{sys}) and the latter as called simulation experimental frame (EF_{sim}). This simulation i.e. virtual EF or simply an EF is an operational formulation of the objectives and needs of an M&S application. In other words, the EF aims to translate objectives in precise experimentation conditions and could be interpreted as a simulation product discussed in section 2.1. The following figure 3.2[1], [Zeigler,2000] better illustrate the relation between the system under test (System) under a given condition (Experimental Frame), its representation (Model) and its behaviour (Simulation).

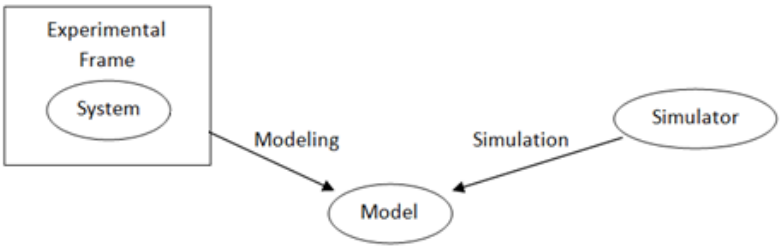


Figure 3.2: Experimental Frame, Model & Simulator, [Zeigler,2000]

In replacing a system by its abstraction i.e. model to build an EF, a model could be conceptual, formal or executable and are listed as follows in the order of development:

Conceptual Model: A conceptual model describes the abstracted and idealized representation of the real system and holds all concepts of the model (or the simulation), i.e., its decomposition into interacting subsystems, the representation of properties of interest in the form of attributes, the degree of abstraction and idealization, and the rationale and reasoning that led to the chosen representation of the real system in the language of the model's application domain. Conceptual Model serves as communication basis and helps in building an insight essential for comprehension and examination of the model as a representation of the real system. In our approach, it may be noted that the conceptual model could be interpreted as MR.

Formal Model: A formal model is the formalized description of the Conceptual Model, compliant with a well-defined modeling formalism, expresses the Conceptual Model quantitatively and unambiguously, and thereby prepares several methods for its solution. The Formal Model, being

[1] Reused with permission from Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, Bernard P. Zeigler, Herbert Praehofer, Tag Gon Kim, Chapter II, Figure 1, Page 26, Copyright Academic Press.

solution-oriented, implementation-independent and unmistakable description of the Conceptual Model, provides the basis for transformation of the model into the Executable Model. This formal model definition could be interpreted as MS in our study.

Executable Model: An executable model, also called implementation model, technically implements the formal model and provides the additional information such as memory allocation, model of computation, distribution aspects etc. that allows the model to be executed and operated on a computer or in a network of computers.

The following figure 3.3, [Albert,2009] illustrates the concepts and differentiates among the model hierarchy,

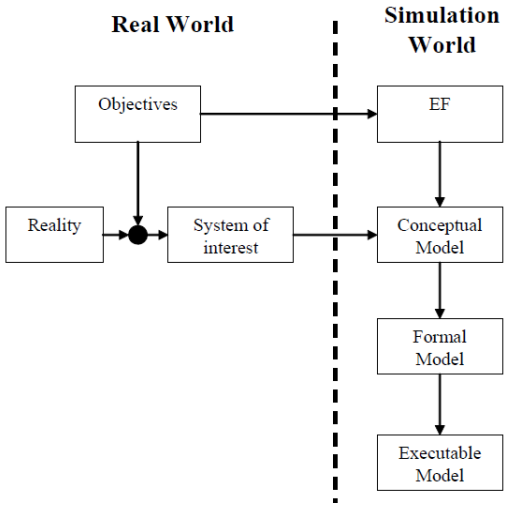


Figure 3.3: Real World vs Simulation World [Albert,2009]

In our study, the focus is on the formal and conceptual model since the transformation from formal to executable model is usually a correction i.e. verification problem. Thus, in this thesis we define the problem of building an EF (SOU or SDU) in two stages, first is the definition of its components and how it is built i.e. architecture which corresponds to conceptual model or MR definition and second is specifying the component itself i.e. defining behaviour which corresponds to formal model or MS.

An EF, in general, is essentially composed of a generator (G), a transducer (T), an acceptor (A) and some environmental models (env). The EF is denoted by $EF = \{M_G \cup M_T \cup M_{env} \cup M_A\}$, where M refers to the EF component which could either be system or their representations i.e. models. The components of EF are given as a tuple $M=<T, u, y>$ with inputs (u), outputs (y) over the time base (T). It may be noted that the components could be at different level of abstraction and the time could either be bounded or unbounded i.e. untimed.

A generator M_G is the stimulant for SUT whereas M_T serves to transform the SUT output into an observable form. The validity of SUT for this set of input and output is assessed by an acceptor M_A which yields a metric on the validity of test output. This is illustrated in the figure 3.4.

The acceptor output is called Degree of Validity (DoV) given either as a qualitative measure or quantitative measure by $f_A: u_A \rightarrow \mathbb{R}_{[0,1]}$. In general, there are varying degrees of validity and scope of test requirements coverage for a SUT and the classical valid/invalid notion may be inadequate

since it cannot differentiate between an EF resulting in more coverage of scenarios than the other EF which covers less scenarios. Instead, a quantitative notion of validity assessment is proposed by DoV notion to assign a real value as a measure of validity. Intuitively, DoV of 0 means invalid and value of 1 means valid with in between values corresponding to partial validity. In the following section, some associated definitions of EF are discussed.

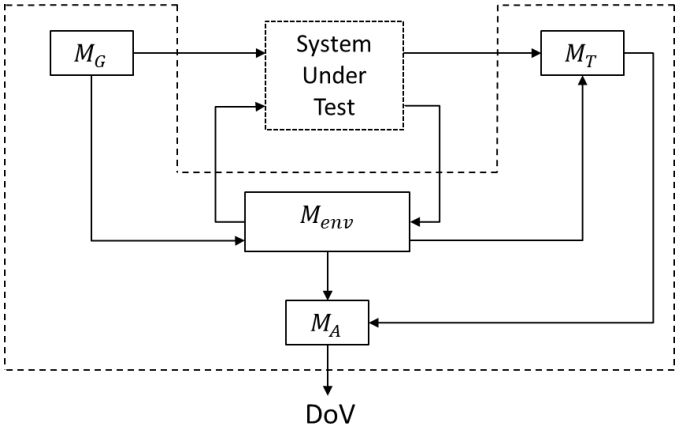


Figure 3.4: Experimental Frame

3.2 EF MORPHISM, APPLICABILITY & DERIVABILITY

The concepts of homomorphism, applicability and derivability were initially proposed by [Zeigler,2000] in the framework of M&S and we extend these definitions formally in the fidelity context. Informally, a morphism relation establishes behavioural equivalence between a concrete model and its abstraction. Applicability and derivability defines a compatibility criterion between a SUT and EF, and also between two experimental frames. A fidelity framework needs to address abstractions with respect to this morphism relation, answer whether the EF can meet simulation objectives and whether the SUT can work with the EF. The concepts are briefly introduced and such perspectives are discussed in the following sections.

3.2.1 Morphism

A morphism relation establishes correspondence between a concrete model, i.e. system specification in our case and its abstract version through abstraction operation. It may be recalled from figure 3.4 and chapter I, such abstraction operations are applied on the EF components resulting in a new EF. Abstractions are manyfold depending on the simulation objectives and hypotheses. We define abstraction operation as α over an abstraction class denoted by $c \in C$ where C is the set of abstraction classes. Such abstractions are related by binary relations forming a partial order. A partially ordered set or a poset is a set $P = (\preccurlyeq, S)$ with reflexive, transitive relation on a set S [Milner,1989]. The hierarchy of abstractions could be defined as a partial order relation over a finite lattice. An abstraction of EF into other is defined as follows [Ponnusamy,2014]

$$\alpha_c^j : EF_k^j \rightarrow EF_k^{j+1} \quad (1)$$

where j=1..n refers to the abstractions of EF as given in Eq. (1) which is essentially abstraction of EF component, $M_p^1 \in EF_k^j$ by an abstraction operation $\alpha_c^{n=1..N}$ resulting in a new abstract EF.

$$M_p^1 \xrightarrow{\alpha_c^1} M_p^2 \xrightarrow{\alpha_c^2} \dots M_p^n \tag{2}$$

Different abstraction operations may be feasible over such a finite lattice whose height is defined by a set N. The valid set of abstractions among them satisfying some properties of interest defined by acceptor, $\varphi_{j=1,2..}$ are defined by

$$\forall\, n \in N, \exists\, \{\alpha_c^n\} \vDash \{\varphi_1, \varphi_2 \dots\} \tag{3}$$

The abstraction operation, α denoted here is generic i.e. it applies to abstraction of model behaviour and architecture (number of ports, coupling, structure, data type at interface etc.). Such a definition is followed by an inclusion criterion further explained in section 3.4 that will help address the simulation validity with respect to the abstractions.

3.2.2 Derivability

In general, according to the V&V plan, different EF could be constructed representing the scenarios ($EF_i \mid i=\{1..N_{sc}\}$) for a given SUT where N_{sc} is the number of such scenarios. EF could be derived from a more general i.e. more capable EF and this operation of derivability is formally given by our following definition [Ponnusamy,2016]

Definition 3.1: Derivability refers to ability to derive an EF_{k+1} from another EF_k.

$$\beta_k : EF_k^j \rightarrow EF_{k+1}^j \tag{4}$$

where j=1..n refers to the abstractions of EF as given in Eq. (1).

It must be noted that this set of derivable EF are included in the EF defined by EF_0 as follows

$$\bigcup_{k=1}^{k=K} EF_k^j \sqsubseteq EF_0^j \tag{5}$$

In other words, a scenario not present in V&V plan could not be derived from the defined scenarios. The derivability is transitive due to the inclusion relation between them,

$$\left(EF_{k+1}^j \sqsubseteq EF_k^j\right) \wedge \left(EF_{k+2}^j \sqsubseteq EF_{k+1}^j\right) \Rightarrow \left(EF_{k+2}^j \sqsubseteq EF_k^j\right) \tag{6}$$

where $k \in \{1..K\}$ gives the limit of such operation.

3.2.3 Applicability

The relation between EF and SUT is given through applicability definition [Ponnusamy,2016] as follows

Definition 3.2: Applicability is an onto relationship between SUT and EF.

$$\gamma_k : EF_k^j \rightarrow SUT_m \tag{7}$$

where m= {1..M} refers to the different SUTs according to the V&V plan.

An EF is composed of different components and depending on abstraction level, different hierarchies of EF are possible for a particular scenario. Such a maximal applicable set of EF is given by

$$\gamma_l^{max} = \bigcup_{k=1}^{k=K_l} EF_k^j \tag{8}$$

where $K_l \leq K$.

The question then becomes, how far a model developer can abstract the components such that the resulting EF i.e. SDU is still applicable to SUT and the SOU can be derived from it. Similar to applicability, derivability can be described in figure 3.5[2] [Ponnusamy,2016], which is modified from [Zeigler,2000], where all these concepts are shown in a three dimensional lattice typology.

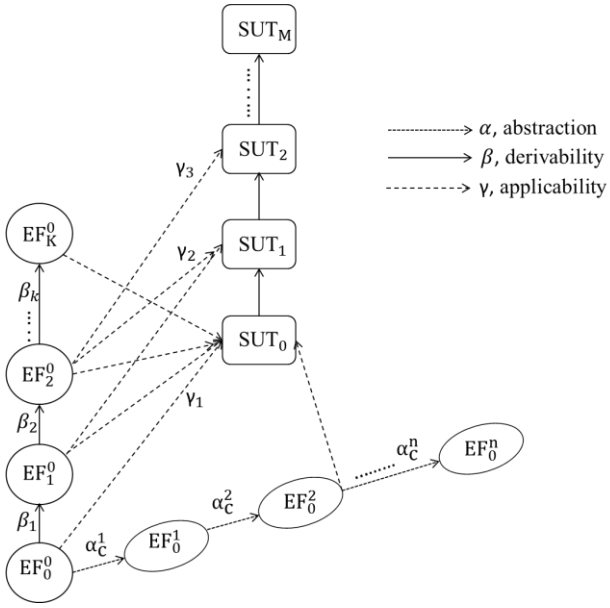


Figure 3.5: Applicability, Derivability & Abstractions - modified from [Zeigler,2000]

[2] Modified with permission from Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, Bernard P. Zeigler, Herbert Praehofer, Tag Gon Kim, Chapter II, Figure 5, Page 34, Copyright Academic Press.

For a particular EF, derivability shown vertically in EF lattice, is the set of all derivable EF of the same EF applicable to same set of SUT as the original EF gives the set of all valid abstractions of EF. In this case, all the derivable frames $EF^{0}_{0..K}$ are applicable to a given SUT_0 and the applicable abstractions (1..n) are the one which under abstraction still obeys this applicability. This applicability of abstractions can be seen for each derivable frame and abstractions must be chosen such that Eq. (7) is obeyed for the required scenario and given SUT. It can be noted that deriving from this EF upwards, the possible scenarios are fewer and the frames become more restrictive. In a sense, it is akin to abstraction in lattice i.e. the $EF_k \sqsubseteq EF_{k-1}$ (scenario inclusion) and $Y_{SUT_{m-1}} \sqsubseteq Y_{SUT_m}$ (SUT output inclusion).

Followed by the discussion of these generic properties of an EF is our formalisation of the problem of building an EF through two key perspectives presented in the next section.

3.3 EXPERIMENTAL FRAME - KEY PERSPECTIVES

In building an EF i.e. simulation application or product itself generally, among others, two key perspectives emerge which are not only functions of the level of abstraction or modeling formalism but also on the proposed solution methodology discussed in later chapters. They are broadly classified in our study as architectural and behavioural perspectives. Such a segregation is natural in terms of expressivity since the architectural perspective is usually expressed in non-formal natural language texts or semi-formal system engineering approaches whereas there exists rigorous mathematical basis for behavioural perspective in most of the cases. It is important to note that each approach is complementary to other and in practice, a formal model is always derived from semi-formal specifications and a semi-formal specification almost always implemented as a formal model. In addition, this segregation is important since in reality a (complex) system is built by multiple stakeholders with different perspectives which are often interrelated. In order to have a truly global approach in improving the fidelity of simulation, it is imperative to incorporate these two approaches. These two perspectives are informally introduced as follows,

***Architectural*:** An architectural perspective addresses how a simulation application is built. It refers to the capture and management of fidelity at higher levels of abstractions such as systems, functions, interfaces, operating modes, ports, data types etc. which helps in engineering an architecture of the simulation application. Some of the key questions addressed in this perspective are: what are systems being modeled, what are the functions needed and how they are allocated on systems? how the systems are interconnected? what are the equipment in each system? what is the granularity of each interface? etc. Essentially this perspective includes, among others,

Functional View : What the system need to do to answer the scenarios and how it will do?

Operational View: How the system is operated, when the system is operated?

Physical View : How the system is built?

In addition, there exist other views such as interaction view i.e. how the system interacts – with user and among themselves, stakeholder view, criticality etc. Such a multi-view modeling is inherent in any system engineering activity and for the sake of convenience these activities are labelled as architectural view i.e. defining a simulation application architecture such that the dynamic evolution is modeled further from this 'black box' view using a behavioural perspective.

***Behavioural*:** The behavioural aspect concerns how the underlying dynamics (temporal or atemporal) is represented and how the system does respond to a scenario. In the behavioural perspective, dynamic evolution of the systems is modeled through different formalisms (discrete, continuous, timed…) and behaviours are captured at the interfaces to study the fidelity of the simulation application being developed. It must be emphasized that this notion of behaviour (e.g. state or output trajectory) is modeled as a transition system.

These perspectives are further explained in the context of solution methodology in chapters III and IV. In the next section, the inclusion relations between EF are presented to address the simulation validity with respect to the abstractions.

3.4 EXPERIMENTAL FRAME'S INCLUSIONS

The architectural and behavioural perspectives of an EF can be attributed to SDU and SOU introduced in section 2.1.1. In other words, a SDU and SOU could be defined in an architectural perspective e.g.: components, connections, functions etc. and behavioural perspective e.g.: tolerance on behaviour etc [Ponnusamy,2014]. The abstractions made when the model i.e. SDU is built must match a set of acceptance conditions given by the SOU [Foures,2013]. An experimental frame typology could be thus found by having equivalence classes according to the system considered (system, equipment, type of system, software, etc.) and the system properties (performance, robustness etc.) targeted by the V&V activity [Albert,2009]. Thus, the objective will be to define a way of (in)formally quantifying the fidelity of a simulation and to define a methodology for finding and implementing the abstractions consistent with the simulation objectives. The problem then becomes how to abstract the EF components such that the distance or 'error' i.e. fidelity introduced by the abstraction operation results in EF consistent with user requirements i.e. SOU. Recalling a simulation product is said to be with sufficient fidelity if its capabilities represented as an EF i.e. SDU in architectural and behavioural perspective *includes* expectations represented as another EF i.e. SOU in the same perspectives, a generic inclusion relation is introduced in this context as follows,

$$\text{SOU} \subseteq \text{SDU} \tag{9}$$

This generic inclusion relation is further explained in terms of architectural and behavioural perspectives in the sections 3.4.1 and 3.4.2 respectively. The following figure 3.6 illustrates this inclusion relation along with the EF concepts introduced in sections 3.2.1 to 3.2.3. The figure could be better understood from the process perspective discussed especially in section 2.2. In general, from the definition of EF, the components $M_{G,T,A}$ corresponds to the scenario under which the SUT and its environment, M_{env} operates whose system specification, M_{sys} is given by the system designers. This system specification can be abstracted by an abstraction operation, α either by the simulation user or model developer. The model is called a reference model if the simulation user defines a set of acceptable abstractions, α_{SOU}. However, in reality such a reference model seldom exist as the SOU is usually described at higher levels operational perspective and in practice M_{env} itself. In practice, the model developer, based on system specification, M_{sys} develops models, M_{env}^{SDU} using abstractions, α_{SDU}. The scenarios under which this model could be used is given by the $M_{G,T,A}^{SDU}$. These scenarios must include the scenarios defined by the user i.e. $M_{G,T,A}^{SOU}$. In other words, there

must be a derivability relation between $\beta_{SOU} : M^{SDU}_{G,T,A} \rightarrow M^{SOU}_{G,T,A}$ such that the resulting EF i.e. M^{SDU}_{env} and $M^{SOU}_{G,T,A}$, can be applied to the SUT using the applicability relations, γ_{SDU}.

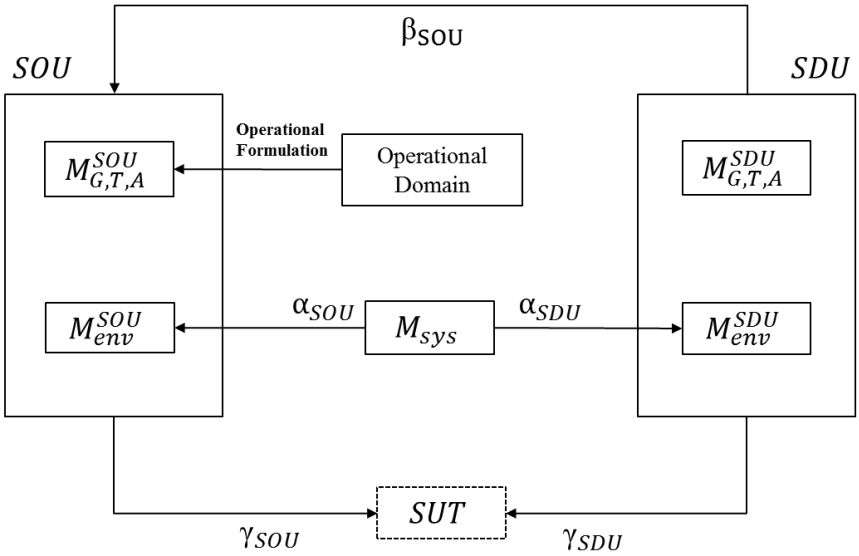


Figure 3.6: SOU & SDU - Applicability, Derivability & Abstractions

In addition, it may be seen that the SOU defined by the simulation user is an operational formulation of the SUT operational domain which in turn is defined by the operational end user.

3.4.1 Architectural Inclusions

The inclusion relations of Eq.(9) in an architectural perspective refers to inclusion of architectural elements required by SOU in the SDU. In practice, this is a two-step process, which is further elaborated in chapter III, with checking consistency and inclusion followed by design space exploration. The first step involves evaluating whether elements of an abstraction class, α^n_c required in SOU exist in SDU. The second step complements the first step since SOU is almost always described in higher levels of abstraction whereas SDU is finely but at times overly detailed. This necessitates exploration of design i.e. SDU to extract architectural elements with respect to SOU.

3.4.2 Behavioural Inclusions

The behavioural inclusion is defined in terms of inclusion relations between SOU and SDU input and output behaviour segments. Intuitively, the model and by extension the EF developed i.e. SDU must be capable of producing at least the same behaviour as expected by the SOU either exactly or within bounds defined by the SOU. The behavioural aspects of EF including the inclusion relations are discussed below starting with a formal definition of an EF.

Formally an EF is defined in the form of the following tuple [Traoré, 2010]

$$EF=<T, u_{EF}, y_{EF}, \Omega_{u_{EF}}, \Omega_{y_{EF}}, SU> \quad (10)$$

where $\Omega_{u_{EF}} \subseteq (T, u_{EF})$, $\Omega_{y_{EF}} \subseteq (T, y_{EF})$.

with T is the time base

u_{EF} are the input variable of EF

y_{EF} are the output variable of EF

$\Omega_{u_{EF}}$ are the set of admissible input segments for the experimental control

$\Omega_{y_{EF}}$ are the set of EF output segments

SU is the set of conditions, also referred to as summary mappings establishing relationship between inputs and outputs within a frame.

Similar to figure 2.1, this EF definition can be illustrated along with the SUT input segment, $\Omega_{u_{SUT}}$ and output segment, $\Omega_{y_{SUT}}$ as follows,

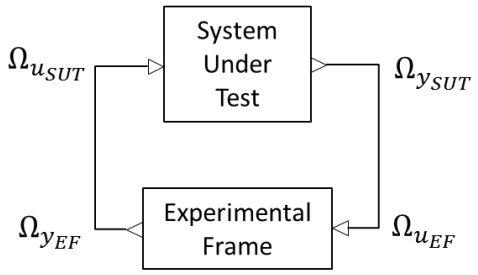


Figure 3.7: EF Definition & SUT

It may be noted the EF input, $\Omega_{u_{EF}}$ need not necessarily be SUT output, $\Omega_{y_{SUT}}$ alone, but also have some free experimental control inputs which is not depicted here. Let us denote the EF input, $\Omega_{y_{EF}}$ which serves as input to the SUT, $\Omega_{u_{SUT}}$ as simply Ω_y. Then, in the context of behavioural fidelity, following Eq.(9), the general inclusion relation between the admissible model segments with respect to its capabilities becomes,

$$\Omega_{y_{SOU}} \subseteq \Omega_{y_{SDU}} \tag{11}$$

The architectural and behavioural perspectives of EF as SDU and SOU are presented with respect to the abstractions in the context of a fidelity framework in the next section.

4. FIDELITY FRAMEWORK & INCLUSIONS

The implementation of the fidelity framework, as remarked in chapter I, is twofold, capturing and managing or assessing fidelity. *Capturing* fidelity needs refers to the collection of fidelity requirements from the SOU in terms of allowable abstraction or required abstraction precision. *Assessing* fidelity refers to quantitative assessment by a (in)formal abstraction compatibility criterion between allowable and implemented abstractions i.e. SOU and SDU respectively. Before addressing a methodology to implement these two aspects, some perspectives on this framework and notions on fidelity distance in the context of abstractions is discussed in this section.

From the definition of the EF specification in Eq.(10), let us denote the system specification (M_{sys}), its reference abstraction i.e. model (M_{SOU}) and implemented abstraction (M_{SDU}). Let us denote the property of a model by φ which generalizes the scenarios definition i.e. given inputs, M_G and expected outputs M_A in section 3.1. Then the V&V cycle in figure 1 of chapter I, can be illustrated in terms of such hierarchical abstraction and requirements. The figure 4.1 can be interpreted as follows, a system specification (M_{sys}) is said to be valid, denoted by $M_{\varphi_{sys}}$, if it satisfies the properties, φ_{sys} which are in turn needed to validate some properties φ_{SUT} of the SUT. Then, in an ideal validity assessment of system by simulation, the simulation user i.e. SOU defines a set of acceptable abstractions (α_{SOU}) resulting in a reference model behaviour representing the system, M_{SOU} and this becomes a necessary and sufficient model if it satisfies φ_{SOU}. Similarly, the model developer i.e. SDU, based on system specification, develops models implementing abstractions (α_{SDU}) resulting in a certain model behaviour, M_{SDU} satisfying φ_{SDU}. A faithful simulation replaces the $M_{\varphi_{sys}}$ by studying $M_{\varphi_{SOU}}$ derivability from $M_{\varphi_{SDU}}$ and thereby allowing to conclude about the SUT. Thus the objective is to develop M_{SDU} consistent with M_{SOU} to answer questions on the SUT. It may also be noted that, from section 2.1, the $M_{G,T}^{SOU}$ derivability comes from $M_{G,T,A}^{SDU}$ or M_A^{SOU} derivability from M_A^{SDU}.

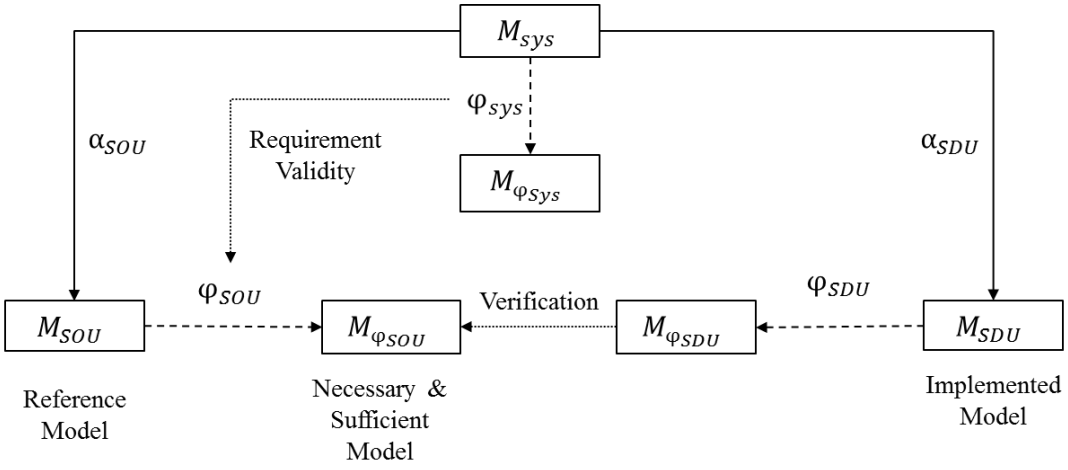


Figure 4.1: Abstraction in Modeling & Simulation

In addition, it must be emphasized that the question of requirement validation is not addressed here and it was assumed that the given simulation requirements are valid with respect to its system requirements and the system specification is valid with respect to φ_{sys} i.e. $M_{\varphi_{sys}}$ is simply noted as M_{sys}.

It is known that an abstraction operation, α is essentially a 'modeling rule' to reduce the complexity of a model and to have a simulation model with sufficient fidelity, the abstraction mandated by the SOU must be derivable from the one actually implemented by the SDU. However, in reality, the user requirements which are usually expressed in an operational perspective do not give α_{SOU} explicitly and even when available, it will be incomplete. Thus the SOU is seldom expressed as a reference model or $M_{\varphi_{SOU}}$, and is usually expressed in terms of some properties and specifications i.e. φ_{SOU} for behaviour and architecture. In particular, acceptable behavioural

abstractions are indirectly expressed as acceptable error tolerances and high level specifications for architecture. In the architectural perspective, often fidelity requirements are only expressed as necessary architectural elements and not as tolerance over them i.e. only as absolute sense. For example, a typical requirement says a component $M_{1,2}$ of type α_c^n and M_3 of type α_c^{n+1} is needed to validate a scenario but not as at least M_1 and M_3 needed for simulation. In addition, since SOU is often at higher abstraction level, it seldom says about composition of $M_{1,2,3}$ and usually it is left to model developer to develop this model specification. Thus in architectural perspective only inclusion is checked *qualitatively* whereas in behavioural perspective it is *quantitatively* checked vis à vis the fidelity tolerances on behaviour. Thus the key question in a fidelity framework is how to find a mechanism to address the issue of identifying and implementing abstractions in an architectural perspective and quantifying the adequacy of implemented abstractions in behavioural perspective.

In particular, in the architectural perspective, fidelity is simply evaluated qualitatively as a Boolean notion i.e. yes/no whereas in the behavioural perspective a quantitative metric is attributed. In general, two key perspectives for a fidelity assessment emerge, namely, verification & synthesis. In a verification perspective, a fidelity assessment method yields a fidelity distance i.e. a metric (qualitative/quantitative) for a given SDU abstraction. The key idea is: *are my abstractions compatible with a metric assigned on its compatibility?* Instead, in synthesis perspective, for a required fidelity defined by the SOU, a fidelity method gives a necessary and sufficient SDU abstraction. The key idea here is: *what are my compatible abstractions with respect to a metric?* However, owing to the fact most of simulation models are built by reusing existing model components and difficulties in fixing a realistic metric by the test team, the present study only concerns the verification aspects of fidelity. In this context, some generic distance notions on fidelity are briefly discussed in the next section. These distance notions correspond in general to both architecture and/or behaviour perspective.

4.1 FIDELITY DISTANCE

In this section, notions of fidelity distance are introduced from the perspectives of SDU and SOU. Let the model fidelity, ε_F, be defined as the distance of the M_{SDU} from the real system, M_{sys}. It must be noted that the simulation fidelity, S_F, and simulation model fidelity, ε_F, are different with the former being affected by factors such as the model of computation (C), execution platform (P) etc. which are not discussed in the scope of our study. Thus the simulation fidelity is the aggregation of all component fidelities, $S_F = \sum(\Sigma\varepsilon_F + P_F + \Sigma C_F + ...)$ [Ponnusamy,2014]. However, for the sake of simplicity the simulation model fidelity is called as simulation fidelity and implicit in this statement is the assumption that other factors such as P_F etc. are perfect. Though this may not be true in reality, most of fidelity issues arise from how the system is modeled i.e. ε_F which is the focus of this study.

Fidelity, resulting from abstraction, is based on both the SUT type and type of operation i.e. SOU. It is to be noted that the fidelity per se is an absolute realism measure of the SDU (what the model can do) *independent* of the SOU (how the model is intended to be used). However, an absolute definition of fidelity is neither feasible nor useful since a model is always abstracted with an objective behind [Gross,1999], [Brade,2004] & [Roza,1999]. A more pertinent question is what is the right level of abstraction for the SDU with respect to SOU? Or succinctly, how do we formally relate fidelity and validity?

Let ε be this distance notion, also referred as abstraction precision and there are two types of distance namely ε_{SDU} and ε_{SOU}. In particular, the absolute fidelity, ε_F^{abs} can be defined as,

$$\varepsilon_F^{abs} = \varepsilon_{SDU} + \varepsilon_X \tag{12}$$

where ε_{SDU} denotes the EF fidelity distance with an unknown additional distance, ε_X, implied by the implementation aspects such as model of computation etc. Since in our study $\varepsilon_F^{abs} \approx \varepsilon_{SDU}$, then δ_F, a fitness measure for *closeness of abstraction* between the SOU and SDU is introduced [Ponnusamy,2014] as,

$$\delta_F = \varepsilon_{SDU}/\varepsilon_{SOU} \tag{13}$$

If δ_F =1, then it is at the right level of abstraction and $\delta_F < 1$, then the abstraction is too precise (over fidelity) and vice versa. In the measured fidelity approach, ε_{SDU} is defined independent of the ε_{SOU} and instead of such an absolute measure, a relative measure called the relative fidelity, ε_F^{rel} which takes into account ε_{SOU} *apriori* need to be defined in the design fidelity approach such that Eq.(13) is improved. It may be recalled that these distances, both absolute and relative, are quantitative measures in the behavioural perspective and qualitative measures in the architectural perspective in our study. In other words, the architectural inclusion and the resulting relative distance is evaluated qualitatively whereas in behavioural perspective the distance is measured quantitatively, both as a global measure, ε_F^{abs} and as a relative measure, ε_F^{rel}. This approach is also similar to two stage fidelity quantitative metric proposed in [SISO,2013] where the first stage answers whether the fidelity is true(>0) or false (0) and the second stage answers if it is true, how far it is greater than 0 on a positive scale. In our case, the binary (true or false) question corresponds to the inclusion question (yes/no) and if included, then how far does the behaviour similar to system is evaluated on a quantitative scale.

In general, a SDU abstraction is valid if it is compatible with the SOU abstraction and this level of compatibility yields a measure of required abstraction. Consider a system dynamics of order i.e. state space size, $n_{sys} = 5$ abstracted to $n_{sdu} = 2$ with $n_{sou} = 3$. This is a case of over abstraction with respect to objective as $\delta_F > 1$. However if the objective is different, say $n_{sou} = 1$, then it is a case of under abstraction with $\delta_F < 1$. Thus the correct abstraction is subjected to the SOU definition i.e. a model may have low fidelity but still be valid.

Consider another simple example, let us assume an ideal system output of $Y_{sys} = 1°$ at interface of the SUT, which is abstracted by the SDU and SOU as range of values, an interval abstraction defined by [min max]. The abstraction is valid if the acceptable range is bigger than the available range and relative fidelity is high as the two ranges are closer.

Proposition 4.1: Let $\alpha_{\varepsilon_{SDU}}$ and $\alpha_{\varepsilon_{SOU}}$ be abstractions of SDU and SOU with distance ε_{SDU} and ε_{SOU} respectively, a simulation product is said to be faithful if the developer abstractions are more precise than user abstractions i.e. $\alpha_{\varepsilon_{SDU}} \preceq \alpha_{\varepsilon_{SOU}}$.

In the framework of fidelity, the abstraction inclusion can be interpreted in verification as well as a synthesis perspective [Ponnusamy,2014]. The SOU lays out the required rule in terms of allowed fidelity distance, ε_{SOU}. The verification problem is checking the distance of allowable abstraction, ε_{SOU}, against the abstraction implemented, M_{SDU},

$$f^{ver}(M_{sys}, M_{SDU}) = \varepsilon_{SDU} \quad (14)$$

Then, by definition of inclusion, a simulation model is valid when $\varepsilon_{SDU} \leq \varepsilon_{SOU}$. In other words, the unknown reference model $M_{\varphi_{SOU}}$ is verified against implemented model $M_{\varphi_{SDU}}$ through the precision of abstraction. In the other case, namely synthesis, for a given precision, ε_{SOU}, a corresponding user abstraction is found, M_{SOU}.

$$f^{syn}(M_{sys}, \varepsilon_{SOU}) = M_{SOU} \quad (15)$$

Consider first example, it is akin to asking what is n_{sou} (a modeling rule) for a given fidelity requirement. Then it is essentially a problem of correction i.e. implementation of reference model M_{SOU} as M_{SDU} in the environmental model.

By partial order relation, for abstraction M^{i}_{SDU} where $i = 1..n$ are different levels of model abstractions, if

$$\begin{aligned} M^{i}_{SDU} &\leq M^{i+1}_{SDU} \\ M^{i+1}_{SDU} &\leq M^{i+2}_{SDU} \end{aligned} \quad (16)$$

Then

$$M^{i}_{SDU} \leq M^{i+2}_{SDU} \quad (17)$$

The optimal abstraction is the one whose precision of abstraction is closest to the required precision.

In addition to abstraction of model behaviour, model interfaces are abstracted based on their syntax definitions and the behaviour they handle. The architecture (number of ports, coupling, structure) and behaviour of the EF and SUT interfaces must be applicable and defined in terms of a partial order relation. Such a definition followed by an inclusion criterion will help address the simulation fidelity with respect to abstractions.

5. CONCLUSION

In this chapter, the problem of capture and manage fidelity of our designed fidelity approach has been discussed as an inclusion problem in the EF formalism. This inclusion problem has been formalised in architectural and behavioural perspective and are assessed qualitatively and quantitatively. A method to verify this inclusion based on a domain model approach for architectural perspective is presented in chapter III and based on a formal approach for behavioural perspective is presented in chapter IV.

CHAPTER III

SYSTEM SIMULATION DOMAIN MODEL APPROACH

In this chapter, an ontology driven domain model approach for improving the fidelity of the simulation by developing models through the inclusion of simulation objectives for the system V&V is presented. The application of this domain modeling technique and semantic web principles to identify, extract and build a MR and then a MS is presented in the following sections.

1. INTRODUCTION

In the M&S of complex systems, recalling section 2.2.1 of chapter I, one of the key challenges in modeling is the lack of common understanding between the stakeholders, semantic inconsistency, and interoperability [Benjamin,2009]. The designed fidelity approach necessitates collection of knowledge about the system to be modeled and scenarios under which it will be operated called System Description (SD) knowledge i.e. M_{sys} and Test Requirements (TR) i.e. M_{SOU} in general and $M_{G,T,A}$ in particular respectively. This is then used to build the Model Requirements (MR), $M_{\varphi_{SOU}}$ through inclusion relations discussed in section 3.4 of chapter II such that the resulting Model Specification (MS) or SDU i.e. M_{SDU} when composed with the SUT satisfy φ_{SOU}.

In general, in building a MS it is imperative for the model developer to understand and incorporate only the essential elements needed for the test and usually this MR is given by the model specialist. However, owing to the complexity of different domains of knowledge involved which are often implicit and incomplete at different levels of abstraction, it is a tedious task to define this MR manually. This is compounded due to the lack of a consistent derivation of low level V&V requirements from high level V&V objectives as illustrated in the figure 2.4 of chapter I. In addition, since modeling can be interpreted as a 'reasoning' problem i.e. inclusion of relevant information about the system being modeled, it is important to identify, relate and organize this information from the domain knowledge [Ponnusamy,2015]. However, this is often a tedious task which necessitates a domain model approach with reasoning and knowledge exploitation capabilities.

An ontology helps to formalize this knowledge and build a domain model since a model can be interpreted as a set of concepts with some relationships between them. An ontology, as defined by ISO 15926, is a formal representation of a set of concepts within a domain and the relationships between those concepts [ISO15926]. A formal ontology is a controlled vocabulary expressed in an ontology representation language with a grammar to express something meaningful within a specified domain of interest [Noy,2001]. Such ontology could be used to build a domain model or a meta-model which is an explicit model of the constructs and rules needed to build specific models

within that domain of interest. As remarked by Pidcock in [Pidlock], a valid meta-model is an ontology, but not all ontologies are modeled explicitly as meta-models.

In the case of M&S, the flexibility of ontology in expressing different domain knowledge in a succinct and standard form could significantly improve the modeling activities by explicitly incorporating the model context of usage and thereby ensuring better simulation fidelity. Ontologies serve as an attractive option due to their standardization in terms of OWL [OWL] language, scalability, and availability of tools such as Protégé [Protégé], Topbraid Composer [Topbraid] etc. with SPARQL [SPARQL] query capabilities. An additional advantage of ontologies is its reasoning i.e. ability to infer knowledge which is otherwise hidden or scattered. The existence of plug-in reasoners with Protégé tool such as Fact++ [Fact++], Hermit [Hermit] etc. helps to draw inferences and check consistency. The inferred ontology can be queried for specific needs with SPARQL, a query language which is used to retrieve and manipulate data stored as Resource Description Framework, RDF, a semantic web standard [Sintek,2002]. In addition, as remarked in [Wagner,2012], [Jenkins,2012], ontologies could be used in conjunction with industry standard SysML [OMG] based MBSE and this will help practicing engineers to capitalize on the graphical syntax of SysML and reasoning capabilities of ontology.

2. STATE OF ART

In understanding and explaining complex systems, usually there exist multiple views of representation since a single perspective may not be adequate to represent the underlying complexity. The system engineering languages such as SysML [OMG], CAPELLA [Roques,2016] which supports MBSE activities addresses this need for multiple viewpoints at different layers of abstraction. However, despite its widespread use in system engineering due to its graphical interface and scalability capabilities, such general purpose languages have a closed semantics and not very flexible to build a domain model [Jenkins,2012]. On the other hand, since a system (or a model) could be considered as a representation of some concepts and relationships between them i.e. knowledge, ontologies could be useful in building such a domain model. Ontologies for system engineering were explored by Graves et al [Graves,2008] to standardize knowledge exchange between stakeholders during product development and utilize the reasoning capabilities for consistency evaluation. The benefits of such an approach in representing static properties such as product structure and challenges in representing dynamic properties were discussed for air systems engineering. This study was then extended to leverage the formal semantics, logical reasoning of ontologies with standard and graphical system engineering approaches using SysML by model transformation between them in [Wagner,2012], [Jenkins,2012]. Though such novel approaches were interesting in building a rigorous MBSE framework, it does not concern the development of ontology per se for the M&S domain.

The interest of ontologies in the M&S domain has been discussed in [Fishwick,2004] [Miller,2004], [Lacy,2004] and an ontology based dictionary of generic M&S terms has been given in [Oren,2014]. In particular, Miller et al [Miller,2004] discusses the potential of leveraging the query, navigation and web-based exchange capabilities [Miller,1997] for M&S. In [Durak,2016], the authors presented an ontology for simulation systems engineering, where the contribution of system engineering to simulation product development has been discussed. Similarly, an ontology for system V&V has been proposed in [Kezadri,2010] with various formalisms and techniques for

the purpose of knowledge sharing between stakeholders. However, all these studies focus on high level knowledge standardization in terms of formalisms, definitions, and taxonomy of M&S. A holistic application of ontology in simulation model development for system validation especially in an industrial process context has not been explored adequately to the best of our knowledge. In addition, using reasoning and query capabilities of ontologies to check for completeness and consistency between two different knowledge bases i.e. SDU and SOU to build a MR has not been explored and this is a particular contribution of our approach. This study envisages such an integrated approach which consolidates knowledge capture via domain model and exploitation techniques to not only build a MR but also build a modeling abstraction library and automated assembly of model for near seamless deployment.

The domain model proposed in this study is based on academic and industrial state of art, interview and questionnaires with stakeholders, Airbus internal V&V processes, documentations and standards. In order to build a domain model with multiple viewpoints, we have used the classical system teleological ontology of Structure, Behaviour, Function (SBF) proposed in [Gero,2004]. The SBF ontology is based on the standard definition of system [Simon,1969], and Functional representation ontology [Chandrasekaran,1993] and specifies the system's function and the causal processes that result in them at multiple layers of abstraction [Goel,2009]. Applications of this SBF framework include automated design and problem solving in the fields of mechanical design [Deng,2002], construction [Clayton,1999], computer aided design [Colombo,2007]. Originally intended for design science, this framework is used to represent designing as a set of processes linking structure, behaviour and function together [Gero,2004]. Such representation is amenable for our designed fidelity approach since a V&V activity is essentially focused on functional and non-functional (e.g.: performance) validation. In particular, the connection between behaviour and function in this ontology is useful for example, to capture knowledge such as a behaviour i.e. causual process such as 'push throttle' on a structural element 'Throttle lever' results in a function 'propel aircraft'. However, these notions, originally intended for teleological modeling of complex systems, could be restrictive and too abstract in expressing the test scenarios in the V&V context which is usually expressed in detail from an operational perspective (e.g.: use cases in SysML or operational layer in CAPELLA) over some architecture or high level behaviour. In order to enhance the classical SBF framework we have proposed the concept of Interface and Operation into a SBFIO ontology. The Operating Modes formalism proposed in this approach is similar to mode automata [Maraninchi,1998], mode chart [Jahanian,1988] but is lightweight and believed to be more flexible and amenable to ascribe functional or system behaviour at higher levels of abstraction. Such a less formal notion is needed to extract a firsthand knowledge on a system, its mode of operation and associated dependent modes without resorting to a detailed modeling to build a MR.

In addition to such standard ontology, a viable domain model approach needs to incorporate the underlying processes of simulation product development discussed in section 2.2 of chapter II. This is especially true when the V&V activities in general and M&S activities in particular are geographically and organizationally scattered. There has been very few studies in this regard [Monceaux,2007], however, of late there has been growing focus on leveraging the potential of ontologies especially in aerospace system engineering [Zayas,2010], [CRYSTAL,2014] and V&V applications. Such studies have broader scope as they concern the overall V&V process whereas our problem is more specific to V&V by simulation. In addition, our approach not only attempts to

build a domain model per se but also demonstrate its viability in MR and MS construction with use case studies.

In addition to building a MR, ontologies could also be used to classify different abstractions used in modeling by the model developers to build a model library, aids in model selection and assembly. A long term benefit of such approach will be to capitalize on the validated abstraction class for a particular scenario which will reduce the modeling time followed by model integration. Modeling abstraction classifications have been made by [Frantz 1994], [Albert,2009] etc. though an ontology implementation has not been done adequately. In addition, the problem of manual model selection from a library is also a time consuming error prone approach. Levy et al in [Levy,1997] proposes a recursive procedure to extract a consistent model from the library according to some requirements. However, a limitation of this approach is it presumes the library is well documented in terms of abstractions employed. However, this is seldom the case since abstractions employed by model developers are not formalized adequately. Hence, in our approach we have implemented the abstraction ontology proposed in [Albert,2009] and then we leverage the reasoning capabilities of ontologies to build and fill the model abstraction library. This would then serve as an input for the recursive procedure implementation to find the consistent model automatically. Similarly, SPARQL query capabilities of the ontology approach for the simulation model assembly were discussed in [Novk,2011]. However, it may be restrictive as it would match model interfaces absolutely whereas in reality there exist a hierarchy. Hence, we extend the concept of abstraction hierarchy defined over lattice in formal verification [Cousot,1992] and semantic annotation [Lickly,2011] to the V&V domain and an informal distance notion is ascribed to the elements of lattice to improve the model assembly query results.

The overall ontology based approach to simulation model development and the system V&V by simulation ontology concepts are elaborated in section 4. This MR and MS construction based on the domain model approach is discussed in a process oriented perspective in section 5,6.

3. PRELIMINARIES

3.1 LANGUAGES & FRAMEWORKS

The concepts and relationships of the ontology are expressed in Web Ontology Language (OWL) from the W3C consortium in the form of classes, individuals and properties [OWL]. This language is originally intended for semantic web where information in the web is given explicit semantics and thereby enabling machines to process information. The OWL extends the Resource Description Framework (RDF) [RDF] and RDF Schema (RDFS) which represents information about resources in a graph form, with the expressive and reasoning power of the description logic.

3.2 LOGICS & REASONING

In the formal knowledge representation, Description Logic (DL) [Baader,2005] based on the predicate logic such as first order logic, describes a domain in terms of concepts i.e. classes, roles i.e. properties or relationships and individuals. This logic focuses on tractable reasoning such as satisfiability, subsumption, consistency etc. which can be verified by reasoned and thus complex concepts can therefore be built up incrementally out of simpler concepts. Reasoners infer this relationship by reification, a concept in logic where an instance of a relation is made the subject of

another relation [Grosof,2003]. The OWL language, based on this DL, capitalizes the well-defined model theoretic semantics of DL whose properties such as complexity, decidability are better understood to represent and exploit knowledge in a formal and consistent manner.

3.3 QUERY

The concepts and relationships of the ontology are expressed in RDF form of a triple with a subject, predicate and object. Let us denote this by a tuple $< \mathcal{C}_s, \mathcal{P}, \mathcal{C}_o >$ where $\mathcal{C}_s$ and $\mathcal{C}_o$ are subject and object concepts with $\mathcal{P}$ being the predicate i.e. property associating them. For example, the triple $< Quantity, Characterizes, Function >$ could be interpreted as a Quantity e.g. 'height' *characterizes* a function e.g. 'Display Altitude'. In this 'height' and 'display altitude' are the individuals i.e. instances of the class 'Quantity' and 'Function' respectively. This information can be queried using the Simple Protocol and RDF Query Language (SPARQL), which is a SQL-like language for querying RDF data [SPARQL]. Queries are constructed in triple pattern of subject, predicate and object using TURTLE syntax with conjunctions, disjunctions and optional patterns such as to filter, sort etc. [TURTLE]. Sample queries can be found in the annex.

4. SIMULATION FIDELITY DOMAIN MODEL

In developing a domain model it is important to incorporate different viewpoints in the system teleological perspective such as Structure, Behaviour, Function (SBF) ontology [Garo,2004]. However, these notions could be restrictive in expressing the test scenarios in the V&V context and we have extended them with the notation of interface (I) and Operation (O) to describe interconnected system with different modes of operation. Together, the ontology is called a SBFIO ontology and in addition it has different generic (Ports, Variable, etc.) and domain specific concepts (ATA chapter, System ID, Siglum, etc.).

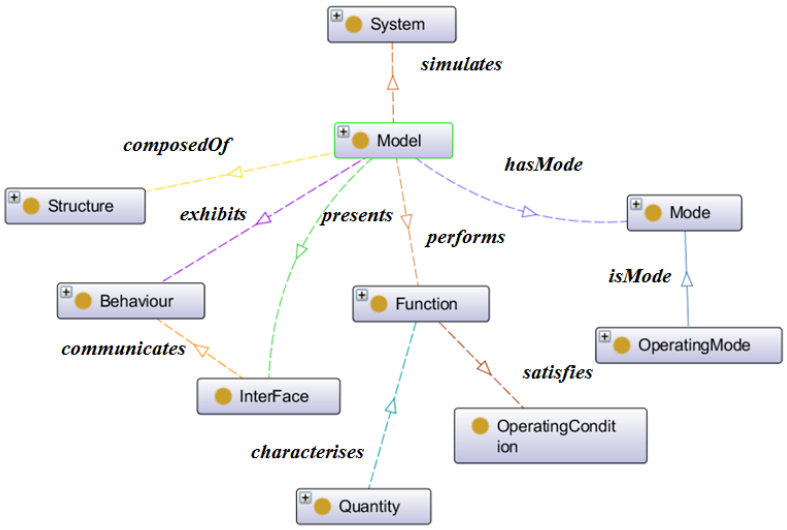


Figure 4.1: SBFIO Domain Model

This ontology and its underlying rules are used to build the system simulation domain model which is implemented in the Protégé tool [Protégé]. The SBFIO part of domain model is illustrated in figure 4.1 [Thebault,2015], [Ponnusamy,2016]. For the sake of brevity not all the concepts are discussed in this section and the focus is only on the important concepts of the domain model. A major part of the domain model is listed in the section 1 of annex for reference.

4.1 SBFIO ONTOLOGY

The key concepts of the SBFIO ontology covering such a perspective are briefly given as follows:

Structure: In addition to classical architectural descriptions of how the system is built (e.g.: composition) [Garo,2004], [OMG], spatial information is included in our domain model. Besides ensuring geometric consistency aspects, the spatial information could be related to the corresponding physical phenomenon and the interaction between the systems.

Behaviour: A system behaviour is the temporal evolution of the system when subjected to some scenario and behavioural abstraction will be briefly discussed in section 4.3. However, this concerns only high level description of behaviour whereas a formal approach to low level concrete behaviour is presented further in chapter IV.

Function: Function describes the system objectives and how they are achieved [Roques,2016]. We also define a system's function as essentially an energy flow manipulation and ascribing domain specific laws to such flow type the phenomenon can be modeled. For example, an aircraft actuator's function is to move the control surface according to pilot's command which involves electrical to mechanical energy conversion. Based on such abstract information the associated laws can be inferred from the library developed by the domain experts.

Interface: Interface refers to how the systems interact among themselves (e.g. I/O ports) or with the external user (e.g. push button). We define interface as the system boundary that can have different attributes such as range, precision etc. It may also be seen as a manifestation of the observable behaviour and is essential in ensuring the consistency at interconnection and composition.

Operation: Operation generally refers to the concepts of 'operating modes' and 'operating condition' of the EF which are introduced in detail in the following section.

4.1.1 Operating Condition

Operating Condition (OC) implies the conditions of environment of the SUT and is used to ascribe assumptions behind models especially at higher abstraction level. In other words, it refers to the assumptions of the EF components and is used in succinctly expressing and identifying operational domains and dependencies. For example, an operating condition of a flight control system at '*takeoff*' phase implies associated assumptions for the engine performance model at this phase. In the next section, operating modes are explained.

4.1.2 Operating Mode

Operating Modes (OM) extends the classical notion of mode-charts [Jahanian,1994], and is similar to mode automata [Maraninchi,1998] but is believed to be more flexible and amenable to

ascribe functional or system behaviour at higher levels of abstraction. Mode charts is a specification language for real time systems whose semantics are given by Real Time Logic (RTL) [Jahanian,1988], a logic for reasoning about the absolute timing of events. Modes are essentially partition of a system's state space and a system can have different possible modes (e.g.: Switch-On/off, Engine-Start/Stop). Then our definition of OM is based on a simple causality relation for interconnected systems with interdependent modes (e.g.: Switch-On THEN Engine-Start). This definition is amenable to ascribe functional behaviour or a semantic behaviour vis à vis the system description. In contrary to the rigorous but less flexible formalisms such as mode automata [Maraninchi,1998], our definition refers to the operational manifestation of a model under a given scenario from a static perspective and eases the understanding of Test Requirements (TR) and System Design (SD) which are usually at different levels of abstraction. In other words, the effects of a component's mode on other components can be observed statically and this helps in better understanding the necessary elements to be modeled whose real dynamic behaviour will be analyzed later using established formalisms such as mode automata.

Definition 4.1: Let us denote a system component by C^i having modes $M_j^i \in M^i$ where i and j refers to the component id and the corresponding mode respectively. The dependency between modes are given by mode inter-connection $I_i^k: M_j^i \rightarrow \bigcup_j M_j^k$ i.e. a mode of a component, C^i may affect one or more modes of other component, C^k such that $I_i^k \subseteq \{M^i \cup M^k\}$. The OM then becomes a tuple [Ponnusamy,2016],

$$OM^{ik} = < C^i, I_i^k, C^k > \tag{1}$$

where the connected modes of C^i are called guards i.e. causative and that of C^k are called states i.e. resultant. Transitions between modes occur whenever the guard mode changes. The transition $T_{n \rightarrow n+1}$ defines transition from one operating mode, OM_n to another, OM_{n+1} when the guard conditions changes i.e. becomes true denoted by $\vdash$ symbol. This is given by,

$$T_n: M_j^i \times \tau \rightarrow M_{j'}^i \mid \tau : M_{j'}^i \vdash \tag{2}$$

where $M_j^i \in OM_n$ and $M_{j'}^i \in OM_{n+1}$. The mode description and its transition is represented in the following figure [Ponnusamy,2016],

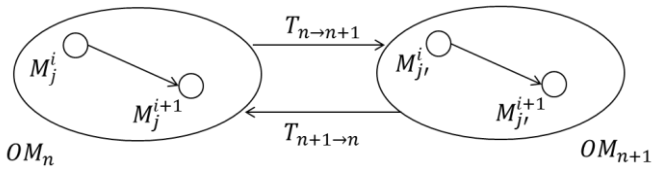


Figure 4.2: Operating Modes & Transition Diagram

For example, consider a system with four components, $C^{i=1..4}$ each having different modes. The dependencies in between them are shown as dotted lines in figure 4.3, for example, the mode M_1^1 affects M_1^2 which in turn affects M_1^4 i.e. components C^4 is dependent on C^1. These modes can be interpreted as reachability space where each state is a possible operating mode of the block and preceding state is the guard operating mode. In other words, when the guard condition is true the

entailment relation between these two modes result in final mode. The evolution from guard to the mode involves dynamics and at each mode too there could be an associated behaviour.

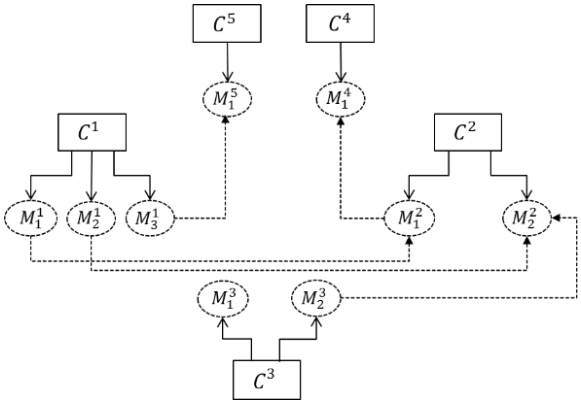


Figure 4.3: Mode Dependency Example

Such dependencies could be illustrated using OM in the following figure 4.4, which could be then reasoned and queried to find implicit information such as modes (un)affected by a particular mode or its attributes (e.g.: type of system, associated designer etc.). In practice the system designer need only gives the component and its dependent modes and the link between different such pairs are extracted automatically. This is useful since the designer usually knows the causality relation only few components upstream and downstream and it is thus important to relate between all such information to have holistic view before modeling the system. In other words, this helps in capturing each component's operational environment assumptions in terms of modes. In the figure below, the causality relation in mode is denoted by solid arrow line and the transition between modes by dotted arrow lines. In addition, transition can be constrained, for example, once mode M_3^1 is activated it cannot be switched to other modes of the component and hence the end state will always be M_1^5. Thus the transitions can be primary i.e. affects other OM or secondary i.e. does not affect other OM e.g.: OM_5.

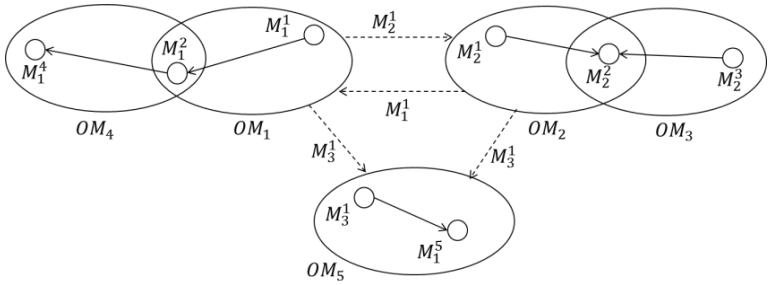


Figure 4.4: Operating Modes

From such illustration, queries can be made on the instantiated domain model for applications such as identification of the transitions between modes and the necessary dependencies to be modeled. For example, reachability notions such as the mode M_1^4 can be reached from M_2^1 by

changing the mode to M_1^1 can be queried. Similarly there are two ways of reaching M_2^2 and associated (or the shortest) path can be queried.

This description will also be useful in high level functional failure mode and effect analysis. A failure could be interpreted as the inability of the system mode to transit in response to its associated causality conditional i.e. guards change. Consider a TR stating simulate C^2 failure and this requirement necessitates inclusion of components associated to C^2 such that any mode change in upstream component i.e. guards does not have effect on C^2 since it is already failed and effect of downstream components i.e. states with respect to it. From SD it is known that C^2 can fail at M_1^2 or M_2^2 and in case of failure at M_1^2 it can be easily infered that M_2^1 will not have any effect and it must be included to check the effect. In addition, OM_4 can be abstracted for simulation of OM_5 since it does not have any transition associated. Similarly recovery procedures such as in case of M_1^2 failure to respond to transition M_2^1, M_2^2 can be reached through OM_5 if there exists a transition i.e. guard change M_3^1 to M_2^1 can be seen. It may be reminded that all such inferences are static i.e. from instantiated domain model through queries and this helps in inclusion of necessary abstractions to be implemented for a given test requirement before dynamically simulating.

4.2 TEST ONTOLOGY

In addition to the SBFIO ontology, a set of concepts to capture the TR in terms of the scope of test, test mean i.e. simulation platform etc. are defined in our domain model approach. Some of the constituent concepts include test condition which describes the conditions enabling a test in terms of SBFIO, test procedure and expected outcomes. In addition, other concepts such as class, criticality, cluster, stakeholder etc. are defined to capture the associated attributes of the test required (refer section 1 of annex).

In the next section, similar to ontologies which capture the system design and test knowledge from system designer and simulation user respectively, an ontology to capture the modeling knowledge from the model developer is presented in the context of building a MS through component model selection and assembly.

4.3 MODELING ABSTRACTION ONTOLOGY

In the M&S of complex systems where the legacy models are usually reused to incrementally build and assemble to form larger, even with a consistent MR, quite often it is difficult to select a model from the library and assemble them. This is predominantly due to the absence of standardization of the modeling mechanism i.e. abstraction employed which is usually mastered by the engineers but not formalized. The problem of choosing an abstraction to represent a phenomenon i.e. modeling, in essence is a reasoning problem as posed by Levy et al in [Levy,1997] since a model developer reasons about a given physical system at different levels of abstraction. Similarly, in the field of Artificial Intelligence (AI), qualitative simulation has been proposed by Kuipers et al which is based on qualitative reasoning about systems [Kuipers,2001]. This reasoning over different abstractions available as a library presumes every model is well documented in terms of abstraction employed. However, this is seldom the case and this incompleteness of the library is due to the lack of formalized description of different types of abstractions and relations between them. In the next section, based on the works of Albert [Albert,2009], a domain model of this modeling abstractions on four axes of scope, computation, data and time is implemented in Protégé and the reasoning capabilities were exploited to build and fill the model abstraction library. In addition, an algorithm

based on [Levy,1997], has been implemented as SysML activity diagram (refer section 3 of annex) to select an abstraction consistent with requirements from this library.

4.3.1 Classification of Abstractions

A model is built to represent one or more system viewpoint described in section 4.1 via abstractions. There exist different taxonomies for abstractions employed in M&S by [Frantz 1994]. In our approach, modeling abstractions are broadly classified into four classes namely, architecture, data, computation and time [Albert,2009] They are briefly described below,

Architecture: Architecture is a structural notion describing the scope and topology of the model using techniques of omission and functional aggregation.

Computation: Computation refers to the modeling and capture of system evolution i.e. dynamics. This behavioural notion includes concepts of I/O relation, accuracy, mathematical precision and dynamic interaction. Abstraction techniques such as equilibration and exogenisation (simplification) are used under this class.

Data: Data dimension refers to the representation of dynamics as data. This includes concepts of data type, unit, domain, resolution and precision with associated abstraction techniques such as type coercion.

Time: Time class refers to the temporal granularity of event ordering described in the simulation scenario and its compatibility with the model in terms of wall clock time, simulator time and mathematical time.

A brief illustration of this model abstraction taxonomy [Albert,2009] implementation in Protégé tool in figure 4.5. For the sake of brevity, the class definitions are not discussed in detail since the focus is, for a given a class description, how to reason and select corresponding modeling abstractions.

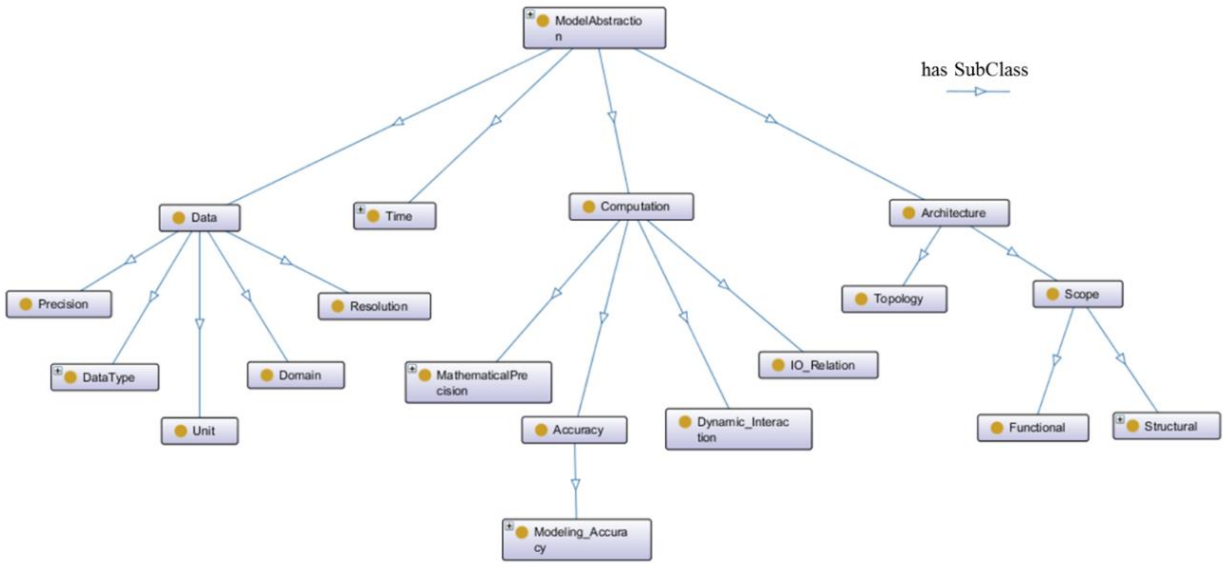


Figure 4.5: Modeling Abstraction Taxonomy

In general, the abstraction classes are identified $c \in \mathfrak{M}$, where $\mathfrak{M}$ is the domain model shown in figure 4.1. Consider a model M_i defined by an abstraction operation α_c^i, where α_c^i is a member of the abstraction class α_c set as described above. This model definition is valid for a certain condition called Operating Condition defined in section 4.1.2. For example, an aerodynamic model with abstraction of only laminar flow is valid for a range of Reynolds number, $Re<Re_{limit}$. Recalling that the hierarchy of abstractions is related by binary relation forming a partial order ($\preccurlyeq$) as follows.

$$M_0 \overset{\alpha_1^1}{\rightarrow} M_1 \overset{\alpha_1^2}{\rightarrow} \dots M_N \tag{3}$$

where M_0 refers to concrete model and n=1..N are possible abstractions.

The model abstraction library lists the models and their corresponding abstraction and operating conditions as described in the table 4.1 [Ponnusamy,2015]. The abstractions defined manually by the developer or user are indicated by the '*' sign and those which are inferred then by reasoning capabilities of the ontology to complete this table to the extent possible, are denoted by '+' sign.

Table 4.1: Model Abstraction Library

Model	Abstraction				Operating Condition
	α_c^i	α_c^{i+1}	α_{c+1}^i	α_{c+1}^{i+1}	
M_0	*			*	OC_1
M_1			+	*	OC_1
….					OC_1
M_N	+	*	*		OC_1

* defined, + inferred

The models described by such a partial relation forms a lattice. Lattice or Hasse diagram is a mathematical diagram of this partial order relation. Such models described over lattices are grouped based on the abstractions. Since a valid abstraction is an operation from a concrete model to an abstract model, where, whatever is true about the concrete model is true in the abstract model but the reverse is not necessarily true, the properties can be inferred from such inheritance relations. From Eq. (3) for models M_n and M_{n+1} and their requirements φ defined over some temporal logic such as Linear Temporal Logic (LTL) or Signal temporal Logic (STL) [Donze, 2014], if

$$M_{n+1} \vDash \{\varphi_{p=1..P}\} \Rightarrow M_n \vDash \{\varphi_{p=1..P}\} \tag{4}$$

Thus for an abstraction belonging to the same class $\alpha_c^{i=1..n}$ arranged over the lattice, implementation of an abstraction α_c^{i+1} also mean the implementation of abstraction α_c^i due to partial order relation $\alpha_c^i \preccurlyeq \alpha_c^{i+1}$. The model abstraction library is thus filled based on these inheritances and dependencies identified by reasoning over the partial order relations. These inclusion relations are exploited to fill the modeling abstraction library and this approach is illustrated with a battery example in section 5.2.1. In the next section, a process oriented view of utilizing this domain model to build the MR and then to build a MS is briefly presented.

5. PROCESS OVERVIEW

The domain model presented in section 4 serves two purpose, namely, to build a MR and then to build a MS. The MR construction is based on the concepts of domain model discussed in sections 4.1-2 and the resulting MR will be then be used to select component models and assemble them to build a MS based on abstraction ontology discussed in section 4.3. The two processes, namely MR construction and MS construction, are briefly presented in the following sections.

5.1 MR CONSTRUCTION PROCESS

The MR construction through the domain model approach essentially has three different steps namely,

1. *Formalization*: Formalize text based SD and TR into domain model instances as SDU and SOU respectively.
2. *Verification* : Verify the inclusion of SOU in SDU
3. *Extraction* : Extract the necessary abstractions from SDU to build MR

5.1.1 Formalization

In the formalization phase, the text based SD and TR are translated to the domain model instances by the system designer and the simulation user respectively. However, this process of translating natural text into domain model is manual at present and a brief discussion on automated translation of this text into instances using Natural Language Processing (NLP) techniques is briefly addressed in the section 1 of chapter VI.

5.1.2 Verification

In the verification phase, the resulting TR and SD models are checked for the consistency, for example, to check whether instantiations mutually are consistent. In the second step of verification, these two models are overlaid, for example by making the instance of SUT required in TR and the corresponding element available in SD as identical. The resulting model is evaluated by the reasoned, for example by Fact++ [FACT++], to find implicit information and queries can be made to check the consistency between them. An example could be querying whether the SUT required by TR and the corresponding element available in SD are of same type or does same function or have same interfaces etc.

5.1.3 Extraction

The extraction phase corresponds to the SD design exploration where the required instances of the SD are extracted based on the TR instances. This could either be done manually or through queries. For example, a query could be written to extract only the equipment connected to the SUT and this can be further filtered according to the equipment specified in TR. Some of the sample queries are listed out in the annex.

The process is illustrated in the following figure. In addition to the sequential process to construct a MR, the archival and reuse capabilities are also illustrated. The approach could help in creating repository of MR, SD and TR which will be a significant value addition for enterprise in terms of knowledge capitalization and reuse. An example of using the inference and query capabilities of this domain model approach to identify and justify abstractions consistent with the test scenarios is illustrated in section 1.2 and with a failure mode case study in section 1.2.3 of chapter V.

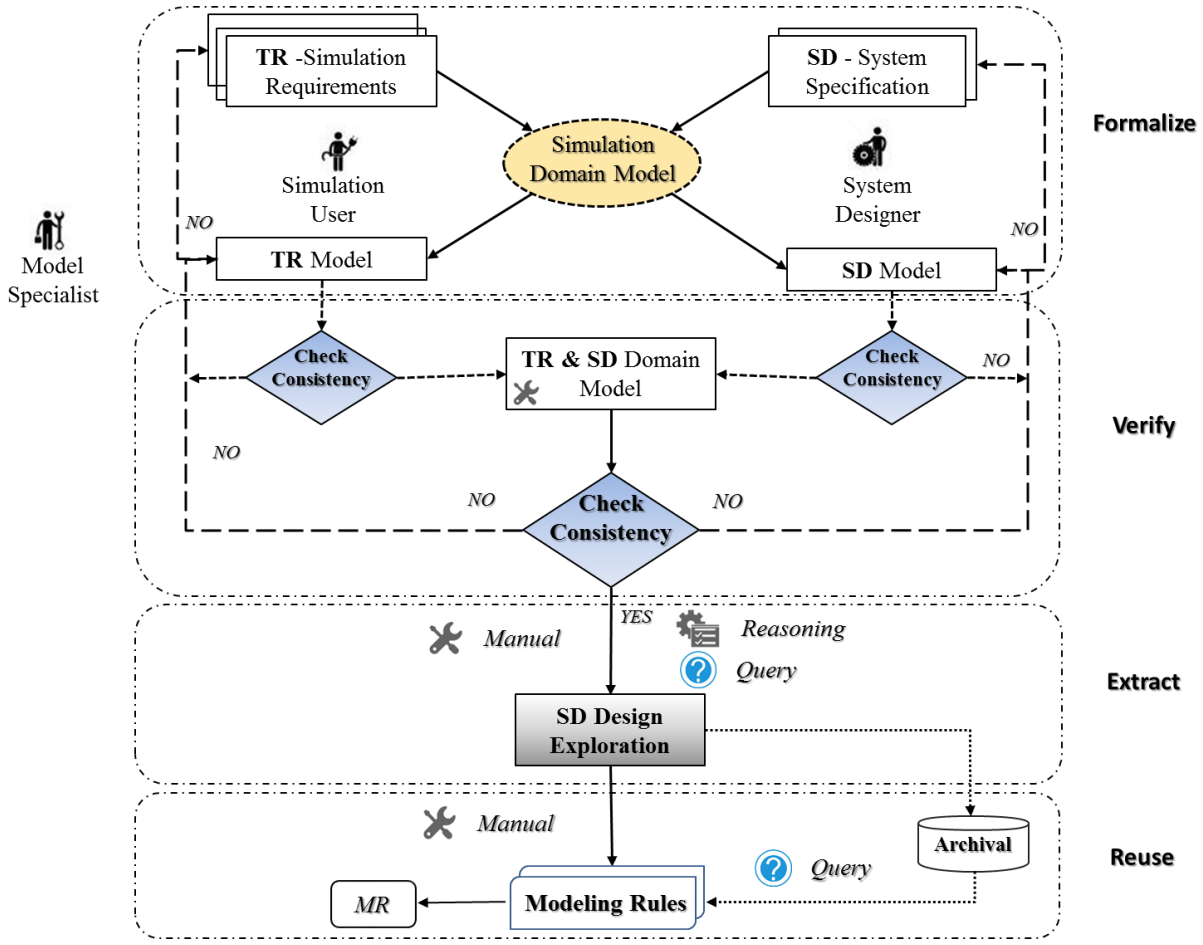


Figure 5.1: MR Construction Process Overview

5.2 MS CONSTRUCTION PROCESS

The MR construction by the model specialist is followed by MS construction by the model developer and integrator which involves three steps, namely,

1. *Model Abstraction Library Construction* : Formalize abstractions used to build component model specifications into abstraction ontology instances.

2. *Automated Model Selection* : Select the simplest yet consistent model according to the MR from the library using the automated model selection process.
3. *Automated Model Assembly* : Extract the models with compatible interfaces for assembly.

The process of MS construction is illustrated in figure 5.1 in conjunction with MR construction process. It can be seen that the MR construction phase is followed by model selection using the abstraction ontology described in sections 4.3 and selection process to be discussed later in section 5.2. In the next sections, the abstraction library construction using the abstraction ontology described in section 4.3 is explained with an application case followed by model selection based on the implementation of recursive procedure proposed by Levy et al [Levy,1997]. Though the main application case for the domain model approach in MR construction is presented in chapter V, this relatively simple application case is illustrated here to demonstrate the synergies of reasoning aided domain model approach in automated model selection with the MR construction.

5.2.1 MODEL ABSTRACTION LIBRARY CONSTRUCTION, MODEL SELECTION: APPLICATION CASE

The application case is a battery system similar to the one described in [Levy,1997]. The battery is connected to a solar panel of a satellite and the function of the battery is to provide power to the panels when the satellite is at the far end of earth without the sunlight. It is known that a phenomena exhibited by the system can be modeled in different ways. Thus the battery can be modeled in different perspectives (e.g.: model voltage phenomena, charge level or a combination of both) and for each perspective it can be modeled in varying granularity of details (e.g.: voltage is independent or dependent of charge level). Every such model may correspond to different operating condition and the challenge is to find an abstraction consistent with the required operating conditions and phenomena.

5.2.1.1 Model Abstraction Library Construction

The model abstraction library based on table 4.1 [Ponnusamy,2015] for this application case has models with Voltage (V) as output with different abstractions on ChargeLevel (CL), time (t), Temperature (T) is given in table 5.1. The model ids are given by the following set, $M_{i=1..6}$ = {*Constant Voltage, Binary Voltage, Normal Degrading-1, Normal Degrading-2, Charge Sensitive, Temperature Sensitive*}. The Operating Condition (OC) corresponds to state of *damage* and *rechargeable* conditions. For this case, there are only two conditions namely {*not damaged*} and {*not damaged, rechargeable*} denoted by OC_1 and OC_2 respectively.

Let us denote a class, c and its instance by a notation $\mathfrak{C}$ and $\mathfrak{I}$ respectively then an instance of class is denoted by $\mathfrak{C}{:}\mathfrak{I}$. Consider a sample model M_5, *Model:ChargeSensitive* which describes the evolution of voltage as function of charge level and time under a condition *not damaged*. The *Quantity: ChargeLevel* and *Quantity:Time* is defined to *characterize* the battery function, *Function:Recharge.* An instance could be defined or inferred and the objective is to minimally define these instances and infer the rest. For example, a model with *OperatingCondition: rechargeable* upon inference becomes *not damaged* too. This rule is encoded in ontology through a *subsumes* relation such that if OC_i *subsumes* OC_j then $OC_1 = OC_1 \cup OC_2$. Similarly, other domain

specific rules could be implemented by domain experts and such template will be useful for other stakeholders to find the dependencies through inference.

Table 5.1. Battery Model Abstraction Library

M	Abstraction					OC
	α_{IO}^{1}	α_{IO}^{2}		α_{IO}^{3}	α_{IO}^{4}	
		α_{1}^{comp}	α_{2}^{comp}			
M_1	$V^{*}=C_0$					OC_1
M_2		$V^{*}=f_0(CL)$				OC_1
M_3		$V^{*}=f_1(t)$				OC_1
M_4		$V^{+}=f_1(t)$	$V^{*}=f_2(t)$			OC_1
M_5				$V^{*}=f_2(t,CL)$		OC_2
M_6					$V^{*}=f_2(t,T,CL)$	OC_2

* defined, + inferred

In addition, queries can be made on the instances to extract required data or match related data. For example, models could be grouped under an assumption classes based on the output quantity, Voltage (V) in this case. Then, using SPARQL queries, all models having same outputs can be extracted and grouped. Similarly, instances of a class *ParameterDependancy* defining the quantities characterizing the function under an operating condition can be queried to answer teleological questions such as listing functions which depends on same parameters etc.

In the following section, only a few abstractions for each class are explained and this method can be extended for others too, provided a hierarchy can be built with binary relationship between them as described by Eq.(3).

5.2.1.1.1 Architectural Abstractions

The architecture relations such as system-subsystem-equipment-component are expressed through *Structure_Composed_of* relationship. For example, the battery system is composed of component such as terminals, switches etc. An instance *Model:Binary_voltage_Model* with the relation *Structure_Composed_of* to another instance *Structure_part: Binary_voltage_Model_Terminal* which in turn related to other instance such as port etc. Intuitively, a simulation user requirement of simulating a battery port implies simulation of its parent system.

5.2.1.1.2 Data Abstractions

A hierarchy of data types could be created using *data_part* property similar to architecture. A simulation model data type abstraction is deemed valid if the data type is at least less abstract than required by the user. For example, describe Data Types (DT) as Float ≼ Int ≼Boolean, and the simulation user required data type DT_{user} as Int and that of model developer, DT_{dev} as float. It is inferred that 'int' is also a 'float' and hence the data type abstraction is deemed valid. These lattice declarations could be extended to other concepts in the context of static model analysis for mitigating model composition errors [Lickly,2011].

5.2.1.1.3 Computation Abstractions

Consider a type of computation abstraction such as accuracy which is the difference between exact solution and approximate solution due to modeling abstractions of the behaviour. One such abstraction is the Model order which refers to the degree of freedom or in other words the ability of model to capture the rate of change of the dynamics. The model dynamics defined with same input quantities could be related with 'Model_Order_part' relationship with the dimension of its space i.e. the complexity of the model. Let the order be defined as, $\mathbb{O}$:M→$\mathcal{N}$, where $\mathcal{N}$ is set of natural numbers. If $(M_2)\preccurlyeq(M_1)$, M_2 is more capable than M_1 and it intuitively implies the former model captures the dynamics of the later as well. Hence the model abstraction at higher order infers the model simulates lower order dynamics too.

In the battery example which models the output voltage as function of different parameters based on their Input-Output (IO) relations. The abstraction hierarchy α^{i}_{IO} corresponds to the number of inputs for the function, f_m where $m \in \mathcal{N}$ is the order of function. For the models of same IO inputs, the hierarchy can be further decomposed on the model order. In the Normal Degrading-1 & 2 case, the second order model, M_4 also simulates first order behaviour given by the model M_3. Similar such reifications i.e. information enrichment can be done for members of other classes such as architecture etc.

The next task after completion of the model abstraction library is to select the model consistent with requirements which necessitates the construction of the lattice which will be explored by the recursive algorithm. The lattice structure can be generated by a lattice plug-in or Formal Concept Analysis (FCA) tools [Ganter,2005] such as Lattice Miner where the abstraction library is given as input in the form of objects and attributes. Similar to the lattice described in [Levy,1997], the generated lattice for models with Voltage as output is shown in the figure 5.2. The objects i.e. models are noted in red and attributes are noted in blue and the inclusion hierarchy can be seen. For example, the *Model:Temperature Sensitive* is modeled by temperature, CL and time whereas the *Model: ChargeSensitive* does not model temperature effect. In other words, the latter model is an abstraction of the former or lower the lattice element higher is the complexity.

Similar lattice can be generated for other consequence quantities or any other assumption classes. This would allow to complete the model abstraction library and this can be done in a hierarchical manner especially. Upon completion of such a library, the next task is to select a model which best suits the requirements and this implementation is discussed in the next section.

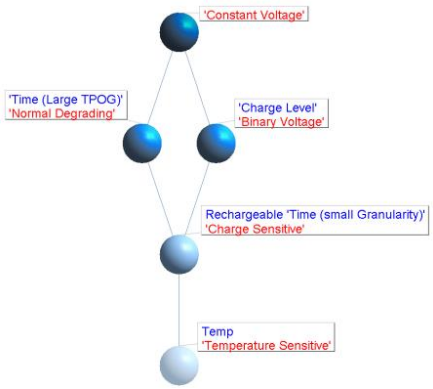


Figure 5.2: Lattice for Voltage Assumption Class

5.2.1.2 Automated Model Selection

In this section, an implementation of the recursive algorithm [Levy,1997] to identify a necessary and sufficient simulation model is presented [Ponnusamy,2015]. The algorithm is executed over the instantiated domain model i.e. model abstraction library previously constructed in section 5.2.1. The resulting output is a selection of consistent model with necessary and sufficient abstraction which is built in the form of parametric diagram to be directly simulated.

Then the model selection problem is to find a necessary and sufficient consistent model called *scenario model* i.e. model attribute of *ModelSelection* class, from the given input of *domain theory*, i.e. a set of model abstraction from the library, called *assumption classes*, and a *query*. A query is characterized as follows

- a list of quantities, *quantity* whose value to be predicted by simulating the system,
- a list of exogenous quantities, E_{input} whose elements are assumed to be given and to be outside the scope of the simulation for which scenario model is constructed.

A domain theory is characterized by a set of assumption classes. An assumption class is a set of models which describe the same phenomenon, i.e. having the same *output quantity* in their consequence based on different and often contradictory modeling conditions. *Quantity,* as described in section 4.1, is an atomic expression denoting time dependent attributes associated with the participants in a model instance. On the other hand, *Consequences* are statements that are true whenever the phenomenon represented by the models takes place. Consequences can also be any other logical assertions that are true in a state in which an instance of the model exists. *Activation conditions* are statements that indicate when the phenomenon represented by the model takes place by specifying constraints on the participants of the model and on its quantities. The conditions include both structural constraints on the participants as well as constraints on the ranges of quantity values. Models are related to each other by a refinement/generalization relationship *Rel.* A model can be related to zero or many other models which are simpler i.e. more abstract or complicated i.e. less abstract. It is assumed that every assumption class has a single most complicated model and a single simplest model. In other words, the lattice is finite with a minimum and maximum.

These concepts were implemented in SysML [OMG,2006] which is detailed in section 3 of annex and the lattice structures are instantiated according to this implementation. The selection algorithm implemented as activity diagram, which is could be found in the annex, is then used to recursively find the consistent yet simplest model. The results for the models which correspond to the query 'Voltage' for conditions *not damaged* is given in the figure 5.3. Informally, the algorithm starts with simplest model and progressively adds the assumption according to the requirement until all the necessary assumption classes are added out of which a simplest model is chosen. In this case, the final scenario model is {*battery-damaged, charge-sensitive, accumulation-with-ageing*} and each selection is highlighted in grey at the end of each iteration in the figure 5.3.

Though the results differ in CL assumption class at the third iteration [Fig 11, Levy et al] the objective is not the algorithm implemented as an activity diagram and its results per se but a description of model library and further model selection in graphical system engineering notion such as SysML for better standardization and understanding of the underlying semantics of the process coupled with ontologies. This is further discussed in section 1 of chapter VI.

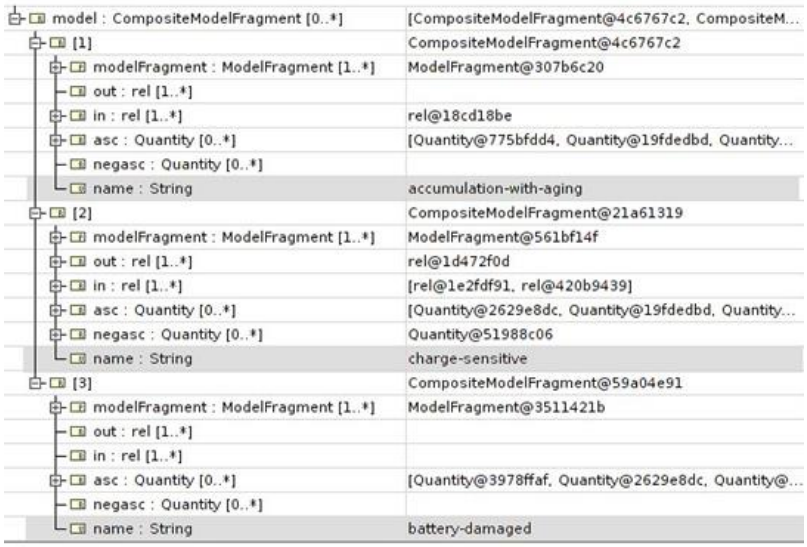


Figure 5.3: Model Selection Results

In the next section, an automated mechanism for assembling different such models is briefly explained.

5.2.2 Automated Model Assembly Implementation

In a component based design framework, the assembly of components is an important but often ignored aspect and many integration problems arise due to interface compatibility. In assembling i.e. connecting two models, compatible models are selected from a library of models by matching their input and output parameters of their interfaces. In [Novàk,2011] this task is discussed via queries of ontology but in an absolute sense i.e. two models are compatible only if the output of first model is same as the input of second model. This could be true for matching parameters, units etc. but for conditions such as matching data types etc. it could be stringent. Consider an example where a battery model (M_1) modeling voltage is connected via an electrical circuit to an antenna model (Ant). The battery output datatype could be '*int'* whereas the antenna model input datatype is '*float'*. A boolean type checking gives an error despite a *float* is also an *int* datatype. In our ontology, when such an instance occurs, the connection is deemed compatible as shown in figure 5.4(b), since in the datatype lattice described in section 5.2.1.1.2, *Float* $\preccurlyeq$ *Int*. This is evaluated by simply measuring the length of its relative position in the lattice chain (e.g.: *int* is located lower than *double* hence it has higher length and only elements with lower length is chosen for input type compatibility).

Let us consider an example where the engine model is connected to accelerometer (*Acc*) model to measure the acceleration, a, induced by the thrust, F. The acceleration can be calculated either as function of force or mass or both and from the set of candidate models shown in figure 5.4.(d) [Ponnusamy,2016] it is evident that second model cannot be used here. From the two available models the first one is chosen for its higher precision if the output datatype is same (or better).

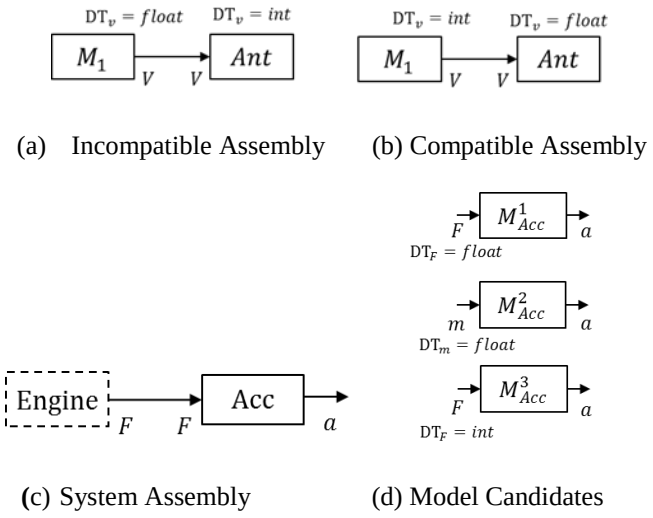


Figure 5.4: Model Assembly

The associated pseudo-queries for this example are given in the section 2.1 of annex. Similar such queries can be written to match or extract other system attributes. The model assembly phase by the model integrators can be seen in figure 5.1 to build a final model specification, MS i.e. simulation product which will then be deployed on a platform for validation activities.

In the next section, an operational perspective of both the MR and MS construction process is presented.

6. OPERATIONAL PERSPECTIVE

In an industrial setting an essential prerequisite for any method proposed to improve the model fidelity is that it must be amenable for integration into the system development process and also need to be user friendly for the practicing engineers. It is thus important to illustrate how the proposed method will be operated and quantify its effect on the 'as is' process. In this context, the operational perspective of the proposed domain model is presented in the figure 6.1 [Thebault,2015]. It can be seen that the proposed approach replaces text with domain model concepts and aided by reasoning over implicit information to make them explicit and evaluate their consistency with respect to each other. This is followed by model selection process and the selected model is instantiated in a classical simulation tool such as Modelica [MODELICA] etc. The phases of MR construction, component model selection and assembly along with respective stakeholders can be seen from the figure 6.1. The simulation domain model described in section 4 for MR and MS construction and the implementation of model selection algorithm are denoted in dotted oval.

This process can be easily integrated in the standard M&S process in industry and it can be seen that this is a non-intrusive method for the engineers since building and exploiting abstraction library is intended to be automated with minimal effort. However, as with any domain model approach in industry, initial effort will be high for tool development, workforce training, process management and deployment. But as several studies demonstrate MBSE is an important enabler in system development especially due to rapid and complex evolution of corpus of engineering knowledge in an organization and the need to capture systematically this engineering knowledge

for standardization and exploitation. The practical challenges in development and deployment of such an approach is discussed in chapter V and VI.

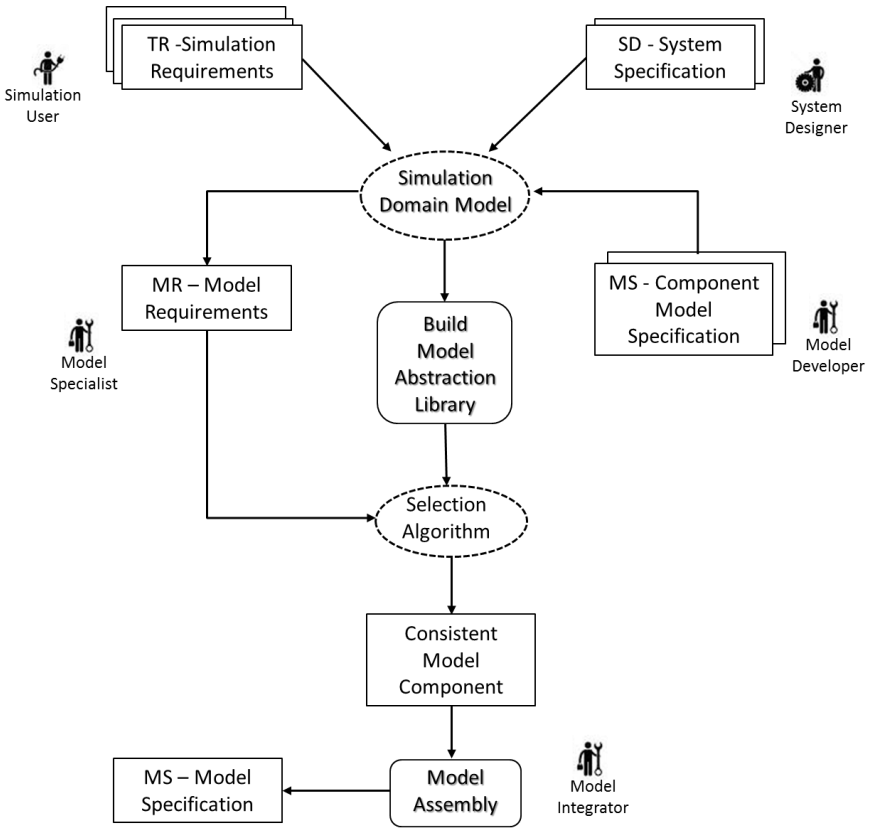


Figure 6.1: Operational View of M&S domain model

7. CONCLUSION

The ontology driven domain model approach presented in this chapter helps ensuring traceability between different abstraction layers and ensures viewpoint consistency and thus enables seamless integration of models and deployment. Such ontology aided simulation design process will enable different stakeholders in simulation to define, solicit and manage knowledge usable for M&S in a consistent way. It helps the test team to optimize the test scenario through inclusion principles and the modularization of ontologies helps in test independence to reduce redundant test combinations. It alleviates the general difficulty of the lack of synchronization and standardization between system development and testing by incrementally and iteratively improves the systems design and testing knowledge with the program schedule. This not only helps in modeling knowledge archiving and reuse for streamlined development of system variants but also for better coordination and decision making in the program development. Realization of such an objective will help

improve the level of confidence in simulation results for the system V&V and help better utilization of simulation resources by selecting the best available resource according to the test objectives.

CHAPTER IV

BEHAVIOURAL FIDELITY METRIC

In this chapter, a formal approach in behavioural fidelity quantification for different class of simulation models is addressed. The problem of formally quantifying the fidelity i.e. distance between the simulation model behaviour vis à vis the system model behaviour is presented through simulation relations and its approximations, game theoretic notions and reachability theory.

1. INTRODUCTION

In the V&V of complex engineering systems, it may be recalled from section 1 of chapter I that the ability of models to replace systems by faithfully reproducing their behaviour is called 'fidelity'. This effectiveness of simulation in reproducing the reality is measured by quantifying the distance between a system and its simulation behaviour *formally* i.e. for all possible behaviours in order to have acceptable degree of confidence in this V&V process. This problem is presented as the problem of quantifying the fidelity of the EF components in this chapter. The current study does not concern the fidelity of the system specification i.e. design model but the simulation model i.e. a subset of design model for the purpose of V&V.

In this chapter, a behavioural fidelity metric for different class of dynamic systems is discussed based on the quantitative simulation relations proposed in the literature, for example in [Henzinger,2013],[Chatterjee,2015],[Girard,2007]. In this study, the term dynamic systems imply systems modeled as state transition systems whose evolution is a function of events and/or time. The broad objective is, given two dynamic systems, one being a system specification and other being an abstraction i.e. a model or possibly a legacy model, how to quantify the degree of fidelity between them for *all* possible behaviours. In other words, how close (or far) does the model matches the events and/or event timings of the system for all possible sequence of events. In the following sections, an informal notion of behavioural fidelity is introduced followed by a formal quantification using the concepts of quantitative simulation functions, reachability theory and its implementation.

2. BEHAVIOURAL FIDELITY

In the M&S of complex systems, especially for the purpose of V&V, one of the fundamental questions in using models to represent a dynamic system is how closely does the model simulate i.e. 'mimic' the system behaviour?. Simulation or Model Fidelity, also called representativity or faithfulness, is this ability of a model to do whatever the system it intends to represent does [Ponnusamy,2016]. In other words, under similar environmental assumptions i.e. inputs, a model with higher fidelity reproduce as many behaviours as the system. This fidelity could be interpreted

as a distance to reality from the fidelity distance definitions in chapter II and there needs to be a mechanism to quantify this distance with respect to all possible behaviours of the system, ε_F^{abs} or a subset of behaviours i.e. with respect to SOU, ε_F^{rel}, before its deployment on a simulation platform.

In this section, it is assumed that EF architecture is defined for example by a MBSE process described in chapter III. The problem then becomes how to ensure the EF behaviour i.e. aggregate behaviour of EF components is consistent with user requirements or more specifically, how to quantify the EF component abstractions such that the resulting composition as an EF will result in a behaviour at the EF-SUT interface consistent with user requirements.

The behavioural fidelity problem can be posed as the compatibility of EF behaviour at the interface to SUT from the perspectives of SDU and SOU. In other words, relative fidelity is the distance between the output of an expected EF i.e. EF_{SOU} and available EF i.e. EF_{SDU} . Recalling Eq.(11) of chapter II, inclusion relation between two such frames whose outputs are denoted by $\Omega_{y_{SOU}}$ and $\Omega_{y_{SDU}}$ respectively, the fidelity of an EF is said to be sufficient i.e. EF is representative if

$$\Omega_{y_{SOU}} \sqsubseteq \Omega_{y_{SDU}} \tag{1}$$

The problem then becomes how to abstract the EF components such that the distance or 'error' introduced by the abstraction operation results in EF behaviour at its output interface consistent with user requirements. In general, a key point in such fidelity quantification for a dynamic system is the origin of the fidelity distance i.e. how a model is built since there exist different ways of modeling. Some of the commonly employed abstraction mechanisms as listed in [Albert,2009] are state aggregation, omission, linearization etc. In the current study, the abstractions are structural i.e. omission of a particular transition. However, the general fidelity quantification technique remains the same irrespective of the abstraction mechanism employed. An abstraction operation over state space of size $n_{s_i}^j$ is defined as follows,

Definition 2.1: Let the abstraction, $\alpha_i^j : R^{n_{s_i}^j} \rightarrow R^{n_{s_i}^{j+1}}$, be a surjection, mapping a model M_i^j to its abstract version, M_i^{j+1}, where $n_{s_i}^j > n_{s_i}^{j+1}$. The hierarchy of abstractions are related by a binary relation forming a partial order. The height of the lattice, $\mathcal{L}_{\alpha_i}$ is given by the size of j={1..n} and the position at the lattice corresponds to abstraction level.

Then the valid set of abstractions among the different set of abstractions for a given model i=n is defined by

$$\forall\, n \in j, \exists \{ \alpha_i^n \} \vDash \{\varphi_1, \varphi_2 \ldots \varphi_z\} \tag{2}$$

where $\varphi_{i=1..z}$ are the requirements defined in formalism such as temporal logic.

The EF fidelity problem as defined in section 2.2.2 of chapter I and section 3.4.2 of chapter II, can then be represented as problem of finding (synthesis) or checking (verification) abstraction(s) $\{ \alpha_i^j \}$ of the system specification, M_i such that the model behaviour is bounded by the required fidelity distance, ε_F, with respect to the expected behaviour at the EF interface to SUT.

$$\alpha_i^j \mid \Omega_{y_{SDU}} \sim_{\varepsilon_F} \Omega_{y_{SOU}} \tag{3}$$

The distance notion can be attributed at the EF interfaces and where the required fidelity is usually given as a distance i.e. tolerance at the SUT input interface. However, in practice, the distance requirements are usually not cascaded top down to individual components which are in turn designed by different stakeholders which necessitates boundedness of this distance under composition. A directed metric [Alfaro,2009] on this behavioural distance, both global and relative, will ensure this boundedness based on the triangular equality principles where the distance between M_{sim}^1 and M_{sys}^1 does not increase when composed with M_{sys}^2. Assuming a sequential composition of models such that, $M_{sys}^1 \oplus M_{sys}^2 \oplus M_{sys}^3$ the distance becomes

$$\varepsilon(M_{sim}^1 \oplus M_{sim}^2) + \varepsilon(M_{sim}^2 \oplus M_{sim}^3) \leq \varepsilon(M_{sim}^1 \oplus M_{sim}^3) \tag{4}$$

In addition, this boundedness under composition is helpful in top down fidelity distance specification as well where this distance i.e. net error $\varepsilon_{EF} = \sum_{i=1}^{N_c} \varepsilon_i$ can be decomposed and cascaded down for each model (M_i, ε_i) to be developed. The procedure is iterative and assignment of tolerance to each component is made in collaboration with the system designer and test team. The next section briefly explains how to define the maximum error tolerance i.e. fidelity specification at the EF or component interfaces.

An informal description of our approach to fidelity quantification for timed systems is briefly presented before a formal description for timed and untimed systems in section 5.

2.1 FIDELITY & ITS QUANTIFICATION – AN INFORMAL INTRODUCTION

Let us consider a V&V activity where some properties of the SUT, φ_{SUT} are evaluated by stimulating and observing this SUT in conjunction with its environment. In this V&V by simulation, these environmental systems M_{sys} are replaced by their models, M_{sim} through some abstraction operation, α such as state omission or aggregation. Such abstractions create distance with respect to the real system's behaviour called fidelity, δ_F and it needs to be quantified for all possible behaviours. This is illustrated in figure 2.1. This quantification is absolute if it is done independent of test cases i.e. some subset of all possible stimulants and relative if it is done with respect to the test cases. An absolute fidelity metric is the (set of) distance measure(s) over the simulation model for all possible scenarios of the system. By contrast, a relative measure is scenario driven i.e. it focuses more on the trajectories related to a given scenario than the others. However, prior to quantifying this fidelity vis à vis its test scenarios, the global measure i.e. for all possible scenarios, needs to be addressed. This intuitively means, how far the model 'mimics' *all* the possible transitions of the system? In this study, absolute fidelity is first presented which would intuitively mean that all possible inputs, the simulation model behaves (within some bounds) same as that of the system and thus the SUT could not see differentiate among them. The relative fidelity distance quantification is essentially a variant of the absolute fidelity distance quantification method discussed in this chapter.

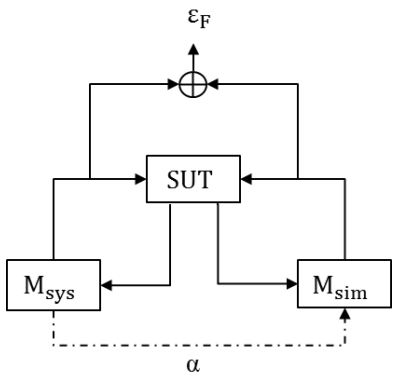


Figure 2.1: Simulation Fidelity

Let us consider a system specification, M_{sys} given by the system designer and a candidate simulation model, M_{sim} as shown in figure 2.2. The dynamics are modeled as a finite labeled timed transition system where for example, from initial state upon receiving a label 'a' (e.g. an input) the system moves to the next state in 2 time units. Alternatively, it may move from initial state to a final state upon receiving a label 'c' in 1 time unit [Ponnusamy,2016].

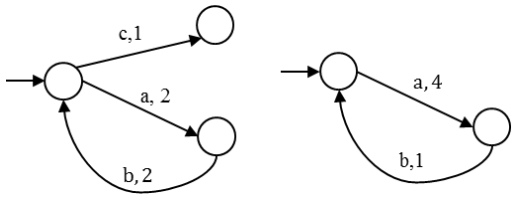


Figure 2.2: System & Simulation Model

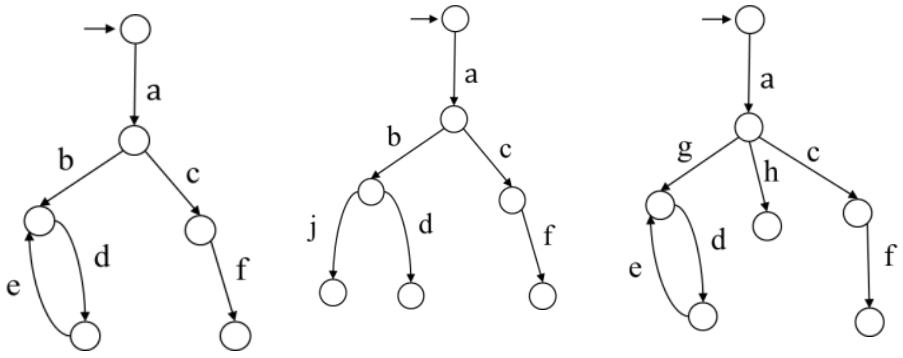
Consider a scenario where the simulation user requires a simulation model which is at least 80% representative i.e. it is required to capture the transitions with 80% (timing) accuracy. For the sake of simplicity, consider the labels of two models are same and they differ only with the time. A model developer, who is tasked with developing or reusing an existing model needs to quantify the model vis à vis this system specification before integrating with other model fragments and deploying on a platform. The objective in this case is to measure the timing difference for each transition and doing for all possible combinations yields a formal fidelity measure. Recalling fidelity is the ability of a model to match every move of the system to the desired degree of accuracy, a two player game can be played between them. Such two player games were widely discussed in the context of software verification, for example in [Kupferman,2000], [Henzinger,2013],[Chatterjee,2015]. In this game the first player also called an *attacker* plays the role of system whereas the second player also called *defender* plays the role of simulation model. A model is said to be with sufficient fidelity (or representative) if the defender wins the game with an acceptable degree of accuracy. In other words, every move of attacker is matched by the defender by the corresponding move if it exists. If the label exists but the timings different, the defender can still make the move albeit incurring a penalty for cheating so and this measure of cheating intuitively corresponds to how far the simulation model captures the transition timings of the system model. The concept of 'cheating' in alternating game has been used in the fidelity

context and it has a literal connotation as well since lexically cheating is the opposite of fidelity which translates to degree of faithfulness. In the given example, first the attacker makes a move with either label 'c' at 1s or 'a' at 2s. For the 'c' move, there exists no counter move by the defender and the game is lost. On the other hand, for the 'a' move by attacker, defender responds with same label in 4s and the time cheat is 2s. For the next move of attacker with 'b' label at 2s, defender's response is 1s and the time cheat is -1s. The net timing error is then 1s at the end of two transition and this error increases linearly for every loop made by the attacker on system model. The resulting errors are evaluated against the user requirement at the end to determine the model adequacy.

2.2 NEED FOR FORMAL QUANTIFICATION

In playing the game as described in previous section, the players are often confronted with different choices and hence there exists different *strategies* at each play. One of the challenges in playing this game is the choice of the strategy. Though different types of strategies have been discussed in literature [Chatterjee,2005], [Chatterjee,2012] most of them are in the context of playing a game on the system vs environment to reach a specific objective such as safety. However, in our case, the objective is to capture how close the game between simulation and system is, such that, they both allow same conclusion to be drawn for an evaluation against a specific V&V objective. In other words, a system may or may not satisfy a particular V&V objective, but, the objective of the simulation model is to faithfully reproduce whatever the system may choose to do. Hence it is important to evaluate all *possible* strategies i.e. a reachability graph. In addition, such an exhaustive exploration needs to quantify the degree of fidelity in every possible path i.e. a quantitative reachability graph, $\mathcal{R}_{\varepsilon}(\mathrm{M}_{\mathrm{sys}}, \mathrm{M}_{\mathrm{sim}})$. This would not only give a path-wise fidelity measure for all possible paths but also help in analyzing the global fidelity as well. This global fidelity could be interpreted as a mean fidelity measure.

Let us illustrate the need to formally quantify this distance by a simple example. Consider the game between the system, $\mathrm{M}_{\mathrm{sys}}$ and some (legacy) simulation models, $\mathrm{M}_{\mathrm{sim}}^{1,2}$ as shown in figure 2.3.a and 2.3.b,c. In general, it may be noted that the problems of behavioural fidelity come from two sources, namely, *un-modeled* dynamics and *incorrectly* modeled dynamics of the system. The former refers to the missing transitions whereas the latter refers to the incorrect transitions. For example, the transition *e* is not modeled in the simulation model (Figure 2.3.b) and the transition *b* is incorrectly modeled i.e. it is modeled as label *g* in the simulation model (Figure 2.3.c). Such information can be quantified via these games. Now, let us play this game informally with player 1 choosing label *a* in the system model. This label is matched by player 2 playing on the simulation model and the error is 0 for both $\mathrm{M}_{\mathrm{sim}}^{1}$ and $\mathrm{M}_{\mathrm{sim}}^{2}$. Now the player 1 chooses *b*, then player 2 does not cheat in the case of the first simulation model, $\mathrm{M}_{\mathrm{sim}}^{1}$. But in the case of the second, it cheats by playing on transitions *g*, *h* or *c* and the error is 1 (or ½ in case total transition weighted). This continues and in fourth play, when player 1 chooses *e*, player 2 playing on first simulation model has no more moves and the game is lost. On the other hand, the other path i.e. {*a*,*c*,*f*} of player 1 can be matched exactly by both the simulation models. For the sake of illustration, total transition weighted errors associated in the quantitative reachability is given in the table below for the first four plays. Thus it can be seen clearly that exploring all the paths of models in this turn base game gives significant insight into the fidelity characteristics of the simulation model [Ponnusamy,2016].



a. M_{sys} b. M_{sim}^{1} c. M_{sim}^{2}

Figure 2.3: System and Simulation Models

Table 2.1: Quantitative Reachability Graph

Play	$\varepsilon(M_{sys}, M_{sim}^{1})$	$\varepsilon(M_{sys}, M_{sim}^{2})$
1	0	0
2	{0,0}	{0.5,0.5,0.5,0}
3	{0,0}	{0.67,∞,0.67,0}
4	{∞}	{0.67, ∞}

In generating such a quantitative reachability graph, how the error is measured could be different depending on the user requirement. However, such an exhaustive approach independent of V&V objectives mean evaluation of all possible behaviours of a system specification against a model i.e. absolute fidelity. In practice, only a subset of the system's state space is explored based on a V&V plan and only such trajectories need be reproduced by the model with adequate accuracy i.e. relative fidelity. This could be factored in our approach by relatively measuring this distance with respect to the trajectories which are part of the V&V plan and this is briefly discussed in section 5.1. These perspectives are discussed in detail for different class of dynamic systems in sections 5,6. It may be noted that a truly absolute measure of fidelity is with respect to the reality which is neither feasible nor useful [Roza,2004] and hence in our study specification is assumed correct and approximated to be the real system. In the following section the scope of our fidelity quantification study is presented followed by the current state of art in behavioural quantification.

2.3 SCOPE OF STUDY

Broadly, the dynamic systems are classified based on the temporal and/or state aspects i.e. the evolution of a system could be a function of time or state or both. The following table concerns the autonomous system i.e. without input but holds true for input driven systems as well. The dark green highlighted classes of systems. It may be noted that this classification does not consider stability or non-determinism e.g. probabilistic systems. In addition, hybrid systems encompass both continuous and discrete dynamics [Tomlin,2003].

Table 2.2: Dynamic System Classification I

Parameter	Stateless	Discrete State	Continuous State
Untimed	Static System	Untimed Automata (UA)	n/a
Discrete Time	Discrete Timed System	Timed Automata (TA)	Discrete Time System (DT)
Continuous Time	Continuous Timed System	Discrete State System (DSS)	Continuous System (CT)

However, such a modeling paradigm is limited to a component perspective i.e. it is a *closed* world environmental assumption where a system is presumed to receive compatible inputs from its environment i.e. other models. In other words, the assumptions which a component makes on its environment are not explicitly considered [Alfaro,2003]. The notion of interfaces has been widely used especially in the component based software design and has been increasingly discussed in the component based system (or model) design as well [Benviste,2012][Alfaro,2005]. In this approach, also called contract based design, an interface implements a component i.e. environmental assumptions are explicitly modeled through a formalism similar like classical automata called interface automata [Alfaro,2001]. An interface automata models the input and output behaviours of a component at its interface. More precisely, it captures the input assumptions and out guarantees and thus amenable to model and reason about the environment of a model. Such assume/guarantee frameworks are an active research area in the context of 'contract based design' [Benviste,2012] for managing complexity, heterogeneity in systems design and V&V. Similar to (un)timed automata formalism modeling discrete state evolution with/without time a classification for interface automata can be presented in table 2.3.

Table 2.3: Dynamic System Classification II

Parameter	Classical Automata	Interface Automata
Untimed	Untimed Automata (UA)	Untimed Interface Automata (UIA)
Discrete Time	Timed Automata (TA)	Timed Interface Automata (TIA)

It must be emphasized that in reality, since cyber physical systems and avionics systems in particular are modeled at different layers of abstraction, different fidelity quantification needs to be developed to ensure adequate levels of fidelity through the various phases of system development and V&V. For example, a flight management system could be modeled as an (un)timed automata whereas the aircraft performance is modeled through differential equations. These systems then interact with other systems built by different stakeholders and it is important to address behavioural fidelity issues in a unified perspective. Our study concerns both these open and closed world environmental assumptions since behavioural fidelity problem could be posed at these two levels of abstractions for dynamics. In case of closed assumptions it becomes, *under closed environmental assumptions how far does the simulation model behaviour different from the system*? In case of open assumptions, it becomes, *how far does the simulation model differs from*

the system in terms of its environment i.e. input assumptions and output guarantees? It can be seen that, both these approaches are often complementary with first question aids in defining behavioural fidelity of an internal structure of component and second question aids in defining its composition. The fidelity quantification study has been presented for (un)timed automata and untimed interface automata along with tool implementation in sections 6.1 and 6.2 whereas continuous systems were discussed section 6 of annex.

3. STATE OF ART

In the formal verification of dynamic systems, automata theory [Clarke,2000] has been extensively studied and used, especially in the discrete world such as software verification. In this paradigm, the dynamic system is modeled as Kripke structure or labeled state transition system which consists of states and transitions which are labelled between those states. In this modeling, the behaviour of a system could be interpreted as a sequence of letters (labels) representing observable events collected as a language which can be checked against its requirement, both specified as ω automaton [Thomas,2002]. This linear view of checking language, also called language inclusion is PSPACE hard for finite state machines [Henzinger,2013]. On the other hand, in a branching time view where the behaviours are captured through tree automata, the algorithmic complexity is only polynomial time. This is based on the concept of simulation relations, introduced in [Milner,1989], which relates two systems based on this branching view and gives a sufficient (but not necessary) condition to check the language inclusion between them. The classical notions of simulation preorders and simulation relations essentially states two models are (bi)similar if every transition of one model is matched by the other (and vice versa). In this context, game theory, in particular two player games since has become an important enabler for many of such verification or synthesis problems [Grädel,2002]. Game theory, in general is a framework for decision making where two or more players take some decisions to achieve a goal either in a collaborative or adversarial manner [Myerson,1991]. However, all these simulation relations are boolean in nature i.e. either the model is exactly similar to the other or not and such boolean notions are too restrictive for practical purposes [Henzinger,2013]. It is not possible to distinguish between a more similar model and less similar model among the set of non-similar models. In our designed fidelity approach, this is akin to choosing a component simulation from a library of existing models to replace a system design model.

Quantitative extensions of these classical boolean notions [Alfaro,2005] were proposed for different class of systems, for example discrete systems in [Cerny,2010][Van,2006], continuous systems in [Girard,2007] etc. These quantitative approaches have been applied to the design of safety controllers, formal verification, model reduction etc. for continuous, discrete and hybrid systems. In this study, such relations are used in the context of simulation fidelity i.e. quantify the degree of similarity between the system and simulation model. Though intended for software verification where a program implementation is compared against a specification, and progressively studied in the context of cyber physical systems [Henzinger,2013], it is natural to extend this paradigm to the domain of simulation where a model could be interpreted as an implementation of a system specification.

In the field of (discrete) simulation, this quantitative distance notion based on the two player game proposed for automata [Cerny,2010], timed automata [Alur,1994] [Chatterjee,2015] gives a

transition-wise or path-wise distance in the context of implementation, coverage and robustness. Such a 'simulation' wise bounds are not adequate not all possible paths are explored and this necessitates finding bounds on all possible trajectories i.e. a quantitative reachability. However, to the best of our knowledge, such a mechanism to quantify this distance for all possible inputs i.e. a superset of test scenarios has not been implemented. This requires generation of a quantitative reachability graph where every possible transition of system is evaluated over a positive real valued distance function. An analysis of this graph will yield further insight into the adequacy of abstraction globally or with respect to V&V objectives as these global distances could be over-approximate since a model could be locally valid despite its poor global fidelity.

In the continuous systems, approximate bisimulation relations were proposed in [Girard,2007] and for linear systems it essentially give a global error bound i.e. maximum degree of dissimilarity between two models at a given time instant and this can formally be verified by geometric over approximation of the reachability set through zonotopes [Girard,2007], ellipsoids [Maler,2002] postpriori etc. However, this approach, as pointed in [Chatterjee,2015] does not take timing information into account. On the other hand, the studies on quantification of timed systems in [Henzinger,2005],[Chatterjee,2015] etc. has neither been discussed in the context of (multi-formalism) M&S especially for V&V nor has there been any method to explore all the player strategies.

A formal approach, as discussed in section 2.2, need cater not only to different formalisms of dynamic systems but also to its interpretations such as closed or open system in order to holistically assess fidelity at multiple layers of abstraction. Since behavioural fidelity problem arise from a simulation model's internal structure (modeled as automata) as well as its environmental assumptions/guarantees (modeled as interfaces) it is important to study the quantitative reachability approach in the context of interfaces too. Alfaro, in [Alfaro,2001], proposed the formalism of interface automata to specify temporal aspects of system interfaces whose transitions are modelled as automata. The simulation pre-orders for such systems were given by alternating simulation relations by Alur et al in the context of open systems as a two player game between a model and its environment [Alur,1998] and its quantitative extensions in [Cerny,2014] similar to quantitative simulation games. This approach too is focussed on different metrics but lacks a formal mechanism to explore all player strategies. In addition, in all the game theoretic frameworks, for open and closed assumptions, there has not been any tool implementation especially with capabilities to perform some analytics to quantitatively assess different simulation models and their fitness for use.

An important enabler of such an approach especially in an industrial context is the availability of user friendly tools especially in the system simulation perspective. There exists plethora of sophisticated formal verification tools such as NuSMV for finite state systems [Cimatti,2005], ABSINT [Cousot,1992] etc. especially for software, UPPAAL for timed systems [Bengtsson,1996], UPPAAL-TIGA for timed games etc. [Chatain,2009]. However, our study needs a modelisation and an explicit reachability enumeration for analysis which are not available in current tools to the best of our knowledge. Hence, we chose to model this game based formal fidelity quantification for all such different class of systems in (Timed) Petrinet formalism. (Timed) Petrinets, is an extension of classical Petrinet formalism [Peterson,1981] with firing time for the events and is widely used in specification and verification of time dependant systems. An extension of it with data handling called Time Transition Systems [Berthomieu,2014] is used in our approach. The token based formalism of the Petrinets are amenable to model such turn based games in

addition to the availability of state of the art and in house developed Petrinet analyzer tool called TINA [Berthomieu,2004]. The tool has a graphical editor and reachability generation capabilities which renders it an attractive choice for our game semantics modeling and quantitative reachability graph generation. This game based fidelity quantification has been implemented in the ProDEVS [Vu,2015] tool in conjunction with TINA which is presented in detail in section 6.

This approach has been presented in sections 4-6 for different class of discrete transition systems. In case of continuous transition systems this abstraction is presented as a controlled invariant problem based on works of Pappas [Pappas,2003], [Pappas,2000] in the context of EF as in section 6 of annex.

4. PRELIMINARIES

4.1 AUTOMATA & SIMULATION RELATIONS

Let us briefly define some preliminaries before describing the game theoretic fidelity notions [Cerny,2014]. It may be recalled that dynamic systems (with finite states) are modeled by a Finite State Automata (FSA) in automata theory. A FSA is defined by a tuple, $T =< \Sigma, X, x^0, \delta, R >$, where Σ is a finite non-empty set of alphabets or labels, X is the finite non-empty set of states, $x^0 \subseteq X$ is the initial non-empty state set, $\delta: X \times \Sigma \rightarrow 2^X$ is the (nondeterministic) transition function and $R \subseteq X$ is the set of accepting states. An accepting run of T over a finite word $\omega = w_0 w_1 \ldots \in \Sigma$ is the sequence of states $x_0 x_1 \ldots \in X$ such that $x_n \in R$. Then the language of T, $\mathcal{L}(T)$ is the set of words accepted by T.

Let us consider two transition systems, $T_1 =< \Sigma_1, X_1, x_1^0, \delta_1 >$ and $T_2 =< \Sigma_2, X_2, x_2^0, \delta_2 >$, with $\tau_1 \in \delta_1, \tau_2 \in \delta_2$, then T_1 simulates T_2 is denoted by $T_1 \preccurlyeq_S T_2$ and it holds if there exists a binary relation $f \subseteq X_1 \times X_2$ such that if $(x_1, x_2) \in f$ then

- $\forall (x_1, \tau_1, x_1') \ \exists (x_2, \tau_2, x_2')$ such that $(x_1', x_2') \in f$ (5)

and it becomes bisimulation, $T_1 \approx_{BS} T_2$ when

- $\forall (x_2, \tau_2, x_2') \ \exists (x_1, \tau_1, x_1')$ such that $(x_1', x_2') \in f$ (6)

It may be noted that the game theoretic approach assumes formalization of the knowledge about the labels i.e. transitions of the system and simulation model. In other words, the homomorphism relation is established between the labels i.e. equivalence of labels. This assumption is reasonable since the two models being developed by different stakeholders needs to have coherency in labels (ex: labels *job* and *j* refers to the same input event i.e. an incoming job) before establishing the simulation relation and quantifying the error between them.

These simulation relations, and in addition alternating simulation relations were extended to quantitative game graphs and this notion is used in the next section to quantify the distance between system and simulation model behaviour for different class of systems in section 5.

4.2 TWO PLAYER GAME: SYSTEM VS SIMULATION

Game theoretic notions have been used in verification as well as synthesis perspectives in the formal modeling and analysis of systems [Henzinger,2013]. In this section, a two player game is briefly introduced followed by the game between the system and simulation model in the context of quantifying its degree of similarity i.e. fidelity.

A (finite) game graph is a tuple, $\mathfrak{g} = < X, X^1, X^2, E, x_0 >$ where X a finite set of states is partitioned as X^1 and X^2 for the first and second player respectively, $E \subseteq X \times X$ is the set of edges, x_0 is the initial state of the play [Cerny,2010]. The dynamics of the transition system described by its states and transitions are interpreted as nodes i.e. states and edges of this game. The game starts with a move the first player followed by the second player and this continues until one wins. The strategy of the player to choose each move may or may not depend on the history of previous moves and in this study we employ the memory-less strategy. The set of visited states in the game is called a play which is denoted by $\rho = \rho_1\rho_2 \ldots$ and this is akin to the path of a transition system or trace if there is a propositional evaluation at each such state.

In the context of fidelity where the game is played between the system model and the simulation model, the latter is deemed representative if the defender wins. However, this necessitates all the moves i.e. transitions of the attacker must be matched. This is too restrictive and infeasible at times and hence the notion of 'cheating' similar to the one introduced in [Cerny,2010] is used. Then this degree of cheating (or alternatively accuracy) can be measured by a weighted error function, ε such as limited average for number of play, n_p , in the game between $T^{1,2}$ with the error function, e comparing labels at the end of each play. It is defined as follows,

$$\varepsilon(\rho) = \liminf_{n_p \to \infty} \frac{1}{n_p} \sum_{i=0}^{n_p - 1} e(\rho_i, \rho_{i+1}) \tag{7}$$

For example, an error of 0.3 means 30% of transitions are 'cheated' or alternatively the model is 70% representative. The error function satisfies the reflexivity and triangular inequality i.e. for all $T^{1,2,3}$, $\varepsilon(T^1, T^1)=0$ and $\varepsilon(T^1, T^3) \leq \varepsilon(T^1, T^2) + \varepsilon(T^2, T^3)$ respectively [Cerny,2010]. It is easy to see that lesser the propensity of the simulation model to cheat, the higher the fidelity will be i.e. cheating is opposite of fidelity.

In addition, from Eq.(7) which gives a path or trajectory wise fidelity measure, the mean fidelity for all possible such trajectories whose size is N_b, at the end of a play, n_p is given by

$$\varepsilon_{n_p}^{avg} = \frac{1}{n_p} \sum_{j=0}^{N_b - 1} \varepsilon(\rho_i) \tag{8}$$

As remarked in [Henzinger,2013], an error could be measured transition wise or moving average etc. and in this study the error is calculated as weighted sum with respect to transition.

Based on the different modeling formalisms listed in table 2.2 and 2.3, these games can be broadly classified as

i. Untimed

ii. Timed

iii. Untimed Interface

iv. Timed Interface

These can also be interpreted as component (timed and untimed automata) and interface (timed and untimed interface automata) perspective as described in detail in section 5.3. The semantics of each game is different especially in the context of simulation fidelity. In the following sections each game is presented along with the tool implementation except for the timed interface. In addition, the relative approach is presented for each class of systems briefly.

5. FORMAL FIDELITY QUANTIFICATION

In this section, our formal fidelity quantification approach is presented for differnt class of discrete systems. The approach for untimed and timed systems with closed environment assumptions (component perspective) are given in section 5.1 and 5.2 resepctively and for untimed system with open environmental assumptions (interface perspective) is given in section 5.3.

5.1 UNTIMED SYSTEMS

In this section, systems whose evolution is a function of only the event also known as label is considered i.e. time abstract. Formally, the game between system, M_{sys} and simulation model, M_{sim} denoted by $\mathfrak{g}(M_{sys}, M_{sim})$ with state space $X_{sys} \times X_{sim}$ is defined as follows [Ponnusamy,2016],

$$\begin{aligned} &\text{Player 1 move} : (x_{sys}, \tau_{sys}, x_{sim}) \longrightarrow (x'_{sys}, \tau_{sim}, x_{sim}) \\ &\text{Player 2 move} : (x_{sys}, \tau_{sim}, x_{sim}) \longrightarrow (x'_{sys}, \tau_{sys}, x'_{sim}) \end{aligned} \tag{9}$$

In this game, player 1, also called as attacker, plays on the system model and player 2, also called as defender, plays on the simulation model. Informally, the game is played as follows,

1. Player 1 plays on system model and hands back the token to player 2.
2. Player 2 plays on the simulation model, matches if the same label exists or cheats with the existing label and hands back the token to player 2
3. The play is over and the error is calculated, for example using Eq.(7).

The next play begins and this continues until any one player wins or the play itself is terminated externally, whichever is earlier. For every move of the attacker, the defender matches the move or cheats over the move and incurs a penalty. The attacker wins if the defender is not able to match his move and the defender wins if it matches every move of the attacker or attacker has no more moves. The game is a perfect information game i.e. the defender has full visibility on the attacker's move. This game is played in such a way that, the defender plays only the attacker's label if it is available in simulation model and if not, it plays all the possible choices. In particular, simulation relation exists if player 2 always has the winning strategy.

5.1.1 Relative Simulation Fidelity Distance

Similar to quantifying the global fidelity, the untimed games can be extended to quantify the relative fidelity as well. Let us consider the example in figure 2.3 and consider a V&V scenario informally (or formally via some temporal logic) stating whenever the user gives the label *a* and then *c*, *f* should always be the output with no error. This scenario is satisfied by both the simulation models. On the other hand, consider another scenario, stating whenever a user gives the label *a* and then *b*, *d* should always be the immediate output. In this case M^1_{sim} does better with error 0 than M^2_{sim} with error 0.5. If the scenario is, given the label *a* and then *b*, eventually the user must observes *e*, then M^2_{sim} is better than M^1_{sim} where the game has been lost. Thus, such a local notion helps in replacing system models with simulation models locally or 'relative' to the objectives. In other words, globally a simulation model could be far from representing the system but it may be adequate to represent the system for a particular V&V scenario. This relativeness vis à vis scenarios could be taken into account through relative weighting i.e. penalizing more the cheats on labels associated to the scenarios and less the cheats on other labels.

Let us denote the actions of interest on system model, M_{sys} by $\tau^{\varphi} \subseteq \tau_{sys} \in \delta_{sys}$ and whenever the defender cheats on these actions it incurs higher penalty than when it does not. The error weighting function is given by $e: \delta \times N \to \mathbb{R}_0^+$ where $\delta = \delta_1 \cup \delta_2$ and N refers to number of transition. It may be recalled from section 4.2 before that the two player game with turns m=1,2 the distance is calculated at the end of every defender move i.e.$\forall 2n$ where $n \in N$ is the number of transitions. The two different weights are denoted by w_1 and w_2 respectively which could either be a simple positive number or a function of transition $w_{1,2}(n)$. Let the label and state of a transition $\tau \in \delta$ be ξ and x such that $\xi \in \Sigma, x \in X$, then [Ponnusamy,2016]

$$\begin{aligned} \forall \tau_{sim} \in \delta_{sim}, \{\xi_{sim} \neq \xi_{sys} \wedge \xi_{sys} \in \tau^{\varphi}\} &\Rightarrow \varepsilon^{\varphi} = w_1(\mathcal{E}) \\ \{\xi_{sim} \neq \xi_{sys} \wedge \xi_{sys} \notin \tau^{\varphi}\} &\Rightarrow \varepsilon^{\varphi} = w_2(\mathcal{E}) \\ \text{else} \qquad\qquad &\varepsilon^{\varphi} = 0 \end{aligned} \tag{10}$$

In assigning weights to the 'cheating' transition, more weight w_1 is given to transitions related to V&V requirements called 'primary' transitions and less weight, w_2 is given to other transitions called 'secondary' transitions. This relies on the discounting principle that models cheating on primary transitions are penalized more and the earlier the cheat, more will be the penalty. In contrary, secondary cheats are penalized more with increasing time. Intuitively, models erring earlier on primary transition are viewed pessimistically whereas models erring earlier on secondary transitions are viewed optimistically on the assumption that they will eventually correct themselves. An example of the discounted and differential weighting is illustrated in following figure 5.1 where w_1,w_2 refers to primary and secondary weights.

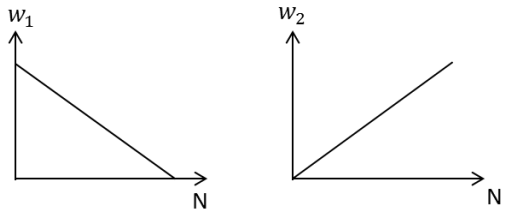


Figure 5.1: Linear Weighting

There are other possibilities of weighting such as quadratic or band limited etc. and is a design parameter in model verification or synthesis process. It may be noted that the transitions made by attacker are given a weight 0 whereas the defender transitions are weighted according to the attacker transition.

For the sake of illustration, consider for every cheating move, $n_c \leq n_p$, let the weight varies in steps of -0.1 for primary weight i.e. $w_1 = (1 - 0.01n_c)$ and +0.1 for secondary weight with each transition i.e. $w_2 = 0.1n_c$ such that $|w_1 + w_2|<1$. Let us consider two simulation models shown in figure 5.2 with the corresponding system model M_{sys} being illustrated in figure 2.3.a. Let the scenario be, whenever *a* (or *a* and then *b*) is given there is possibility to get at least four *e*'s at the end of ten transitions.

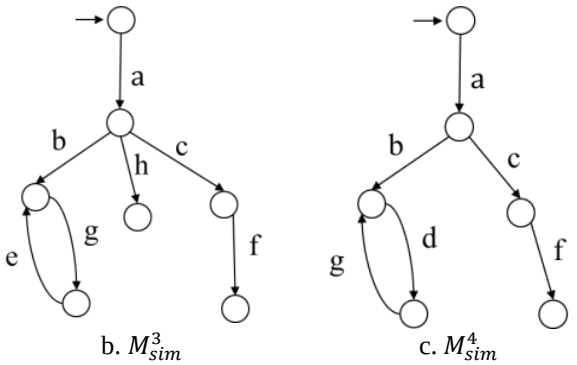


Figure 5.2: Simulation Models Relative Fidelity

In general, a scenario independent error quantification will yield a global value of 0.4 at the end of tenth play for both the models. Instead, the labels in scenario 'a', 'e' are given more weightage during cheating and intuitively one can see that, M^3_{sim} is better than M^4_{sim}. At the end of tenth play, n_c is 4 for both the models and, the relative error becomes $\varepsilon(M_{sys}, M^3_{sim})$=0.16 and $\varepsilon(M_{sys}, M^4_{sim}) = 0.384$. Such relative weighting can be integrated in the quantitative reachability graph generation. However, the weighting needs to be chosen carefully, a too stringent weighting may not show much difference with absolute error calculation and a too lenient weighting leads to spurious results. Further work is needed in this direction which is discussed in section 7 in this chapter and section 2 of chapter VI.

5.2 TIMED SYSTEMS

The fidelity quantification for systems whose dynamics is influenced by both event as well as time is described in this section. Timed automata [Alur,1994] is a classical formalism used to represent real time systems and this formalism extends the classical finite automata with clock variables constraining the system behaviour. Similar to Eq.(9), the formal definition of timed automata and the simulation relations between them are presented as follows.

Let the time domain be $\mathbb{T}$ with non-negative set of reals $\mathbb{R}_+$ and over this time domain the timed automata is defined by $\mathcal{T}=<\Sigma, X, T, x^0, \delta, R>$, where Σ is a finite non-empty set of alphabets or labels, X is the finite non-empty set of states, C is a finite set of clocks, $x^0 \subseteq X$ is the initial non-empty state set, $\delta: X \times \Sigma \times T \rightarrow 2^X$ is the transition function and $R \subseteq X$ is the set of accepting states.

An accepting run of $\mathcal{T}$ over a finite word $\omega=w_0w_1 \ldots \in \Sigma$ is the sequence of states $x_0x_1 \ldots \in X$ such that $x_0 \in x^0$. Then the language of , $\mathcal{L}(\mathcal{T})$ is the set of words accepted by $\mathcal{T}$.

Let us consider two transition systems, $\mathcal{T}_1 =< \Sigma_1, X_1, T_1, x_1^0, \delta_1 >$ and $\mathcal{T}_2 =< \Sigma_2, X_2, T_1, x_2^0, \delta_2 >$, with $\tau_1 \in \delta_1, \tau_2 \in \delta_2$, then T_1 simulates T_2 is denoted by $\mathcal{T}_1 \preccurlyeq_S \mathcal{T}_2$ and it holds if there exists a binary relation $f \subseteq X_1 \times X_2$ such that if $(x_1, x_2) \in f$ then

$$\forall\ (x_1, \tau_1, x_1')\ \ \exists\ (x_2, \tau_2, x_2') \text{ such that } (x_1', x_2') \in f \tag{11}$$

and it becomes bisimulation, $T_1 \approx_{BS} T_2$ when

$$\forall\ (x_2, \tau_2, x_2')\ \ \exists\ (x_1, \tau_1, x_1') \text{ such that } (x_1', x_2') \in f \tag{12}$$

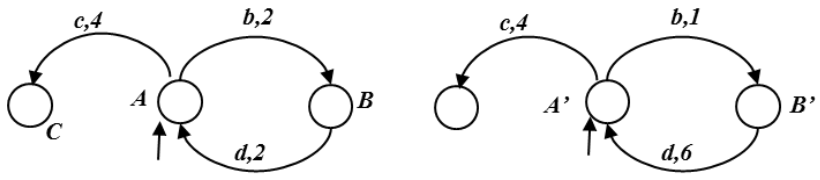
These simulation relations are usually boolean i.e. a simulation model either simulates the system or not. Quantitative extensions of these boolean notions are based on finite-state turn based two player game graphs [Chatterjee,2015], [Henzinger,2005]. These games are informally presented in the next section followed by formal explanation.

5.2.1 Timed Simulation Games

It may be recalled from section 5 that in the untimed game starts from state $x_0 \in X$ with a player 1 making the move to $x_1 \in X^1$ to which the player 2 counters by making a move $x_2 \in X^2$. The first play is over now and the game is started again. At the end of first play, if the player 2 cannot match player 1's move it is allowed to *cheat* and in doing so incurs a penalty and there are different ways of measuring this cheat such as weighted mean etc. Every move on the system model by the first player is followed by the second player on simulation model and this continues until one wins. However, in timed game, the turn based semantics of the game does not strictly hold true due to the temporal nature. The evolution of player 1 is independent of the player 2 since the objective of player 2 is to match player 1 timings. In other words, player 2 is not allowed to win by infinitely blocking the player 1's turn whereas it wait until player 1 finishes its turn [Ponnusamy,2016].

Proposition 5.1: Player 1 can block time of player 2

Let us assume a system and simulation model in the figure 5.3



(a) M_{sys} (b) M_{sim}^1

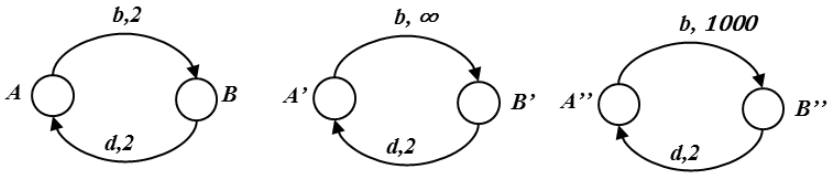
Figure 5.3: Blocking Game

In this case, without blocking, player 2 label 'b' is fired earlier then if player 1 moves 'c' instead, there is a cheat whereas in reality the player 2 does not cheat for 'c' transition. The

blockage of time helps to avoid this problem. Intuitively, a simulation model has to mimic system model so it has to see what the system does first or else it may end up in cheating if a way not to cheat is possible.

Proposition 5.2: Player 2 cannot block time of player 1

This assumption, also found in literature [Chatain,2009], could be explained with the following example of game between a system and simulation model in the figure 5.4. In this case, the third model is a better approximation of the first model than the second. However, if the game is played for <1002 time units both the simulation models are deemed unfit and the system model cannot move further from state B. This can be mitigated by segregating the evolution of system model from that of simulation model. In such case, the time difference is 998 time units for the third model and ∞ for the second model.



(a) M_{sys} (b) M^1_{sim} (c) M^2_{sim}

Figure 5.4: Non-blocking Game

Then, formally, the game between system, M_{sys} and simulation model, M_{sim} is denoted by $g(M_{sys}, M_{sim})$ with state space $X_{sys} \times X_{sim}$. Let $\sigma_i^{1,2}$ be label and $t_i^{1,2}$ be associated transition time and of player 1 and 2 respectively at play *i*, $\tau_{sys} \in \delta_{sys}$ and $\tau_{sim} \in \delta_{sim}$, player actions of selecting a transition from one model and handing over the turn to other player i.e. enabling transition of the other model are denoted by $p_1: \tau_{sys} \longrightarrow \tau_{sim}$ and $p_2: \tau_{sim} \longrightarrow \tau_{sys}$. For a given play of positive integers, $i \in \mathbb{I}_+$, player 1 move is defined as follows,

$$(x_{sys}, \tau_{sys}, x_{sim}) \xrightarrow{p_1} (x'_{sys}, \tau_{sim}, x_{sim}) \tag{13}$$

with the transition time of simulation model

$$t_i^2 = t_i^2 + t_{B_i} | t_{B_i} = t_i^1 \qquad \text{if } t_i^1 > t_i^2 \tag{14}$$

where $t_B$is the blocked time for player 2. Then the player 2 move is defined as

$$(x'_{sys}, \tau_{sim}, x_{sim}) \xrightarrow{p_2} (x'_{sys}, \tau_{sys}, x'_{sim}) \text{ if } \begin{cases} \sigma_i^1 = \sigma_i^2 \\ t_i^2 \leq t_{i+1}^1 \end{cases} \tag{15}$$

The play is terminated if $\sigma_i^1 \neq \sigma_i^2$ regardless of their transition times and the player 1 is deemed won. In all other cases, the next play, *i*+1, is started with player 1 move if $t_i^2 > t_{i+1}^1$. At the end of

each completed play, the time difference between the corresponding transitions i.e. labels, Δt_i is calculated using [Ponnusamy,2016],

$$\Delta t_i = \left(t_i^2 - \sum_{n=1}^{i} t_{B_i} \right) - t_i^1 \tag{16}$$

It may also be seen that such error function being a directed metric [Chatterjee,2015] satisfies the reflexivity and triangular inequality i.e. for all, $\Delta t\,(\mathcal{T}_1, \mathcal{T}_1)=0$ and $\Delta t\,(\mathcal{T}_1, \mathcal{T}_3) \leq \Delta t(\mathcal{T}_1, \mathcal{T}_2) + \Delta t(\mathcal{T}_2, \mathcal{T}_3)$ respectively. This helps in incremental model development and assembly with bounded timing error on the resulting composition.

The timing error quantification through this game based approach can be extended to system and/or simulation models whose transition timings are not defined precisely but in an interval as well. Let us define such interval for the system and simulation model as $[t^{1,2^{lb}}\ t^{1,2^{ub}}]$ where *lb* and *ub* refers to lower and upper bounds on transition timings. In this case, intuitively the interval difference is the timing difference and Eq.(14) becomes,

$$t_i^{2^{lb}} = t_i^{2^{lb}} + t_{B_i} | t_{B_i} = t_i^{1^{lb}} \qquad \text{if } t_i^{1^{lb}} > t_i^{2^{lb}} \tag{17}$$

where t_B is the blocked time for player 2. In other words, the transition of player 2 is enabled once player 1's lower bound transition time is enabled. Then the interval timing error, $[\Delta t_i^{lb}\ \Delta t_i^{ub}]$ is calculated as,

$$\Delta t_i^{lb} = \left(t_i^{2^{lb}} - \sum_{n=1}^{i} t_{B_i} \right) - t_i^{1^{lb}} \tag{18}$$

$$\Delta t_i^{ub} = \left(t_i^{2^{ub}} - \sum_{n=1}^{i} t_{B_i} \right) - t_i^{1^{ub}}$$

However, such interval error quantification needs to be further studied and is not yet implemented in our tool and only transitions fired at punctual time i.e. $t^{1,2^{lb}} = t^{1,2^{ub}}$ is considered in this study.

In discussing fidelity quantification through such game based approach, one of the key difficulties as discussed earlier is exploring the player's strategies. In this context, a reachability graph generation which explores all the player's strategies to quantitatively determine the corresponding transition timings is presented in the next section.

5.2.2 Timed Quantitative Reachability

In the timed games generating a reachability set is an exhaustive exploration of all the player strategies similar to section 2.2. However, in contrast to untimed games, continuous evolution of time for the attacker and blocking for the defender need to be taken into account in the play and error quantification as well. Consider the example, initially proposed in figure 2.2 of section 2.1,

in this case, the game is played with blocking as explained in section 5.2.1. In this example, the defender is blocked for 'a' or 'c' move after which there is no blocking since the second transition of simulation model is faster. The pseudo reachability graph of this game is given for the purpose of illustration in figure 5.5. The transitions of attacker and defender are given in solid and dotted arrows respectively. The first play is over at 6s and the second play is over at 7s and it can be seen that the blocking time is 2s. Since there is no matching transition for attacker move on 'c' label the game is locally lost on this path. Then the timing error is calculated as $\Delta t_1 = (6-2) - 2 = 2$ and $\Delta t_2 = (7-2) - 5 = 0$ and so on. It can be easily seen that the pair wise timing error can be deducted from this aggregate time, for example, the timing difference for second turn is -2 time units. Such evolution can be analyzed and visualized for better understanding of the model fidelity. This is further demonstrated in the application case in section 2.3 of chapter V and implementation is presented in section 6.1 [Ponnusamy,2016].

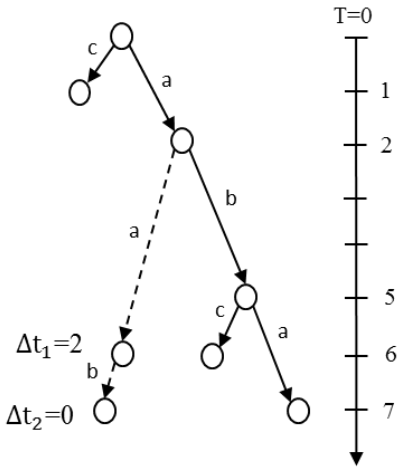


Figure 5.5: Quantitative Timed Reachability Graph

5.3 BEHAVIOURAL FIDELITY QUANTIFICATION AT INTERFACES

In system modeling, the classical assumption of a system being closed i.e. does not react to it is environment or where the assumptions on its input environment are closed is inadequate for representing embedded systems whose behaviour is influenced by its environment. The closed system's assumption of known inputs i.e. environment provides proper inputs is too strong an assumption since it assumes an ideal world where each system developer has same environment assumptions and interconnection between them poses no problem [Alfaro,2005]. But this is not the case as each system is developed with its own environmental assumptions and it needs to be captured for correct composition. In contrast, the open system's behaviour is influenced jointly by its internal structure modeled as a transition system and its environment. The open system dynamics can be described by a two player game between environment (choice of input) and internal structure (choice of output) [Alfaro,2003]. This game based notion helps in refinement and composition of systems i.e. compatibility (each component is prepared to receive any request that the other may issue, if not it results in implementation violating spec i.e. no more behaviours included).

5.3.1 Interface & Fidelity

In the M&S, as discussed in section 2.3, the fidelity problem can be posed as environmental assumption problem. The user requires SUT be tested at some environment called scenario i.e. SUT requires its environment to provide proper input to produce some behaviour that will be validated according to some V&V criteria. On the other hand, the designer supplies EF with his own assumptions. Intuitively, the required assumption i.e. scenario must be a part of supplied system's environmental assumption for the sake of compatibility and fidelity arises out of this compatibility problem i.e. mismatch of assumptions. At higher level these can be captured by ontologies or SysML [OMG,2006] but it must be captured at lower behavioural level too which necessitates a formalism. In this context, Interface automata are useful here since it says two interfaces are compatible if there exists at least one environment in which they can work together.

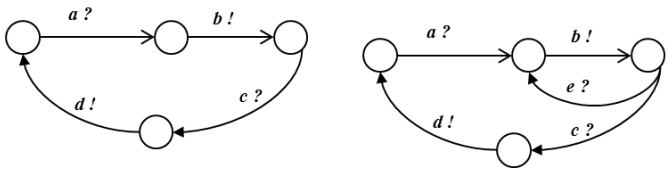
In V&V, a model replaces the system and composed with SUT besides other systems. Let the model and system be generically called component implementing some interfaces [Alfaro,2005]. Fidelity, recalling definitions in section 2, is nothing but a measure of a model's ability to replace a system such that SUT cannot differentiate between them. Such behavioural fidelity issues naturally come from two problems. First, how the model component is modeled i.e. internal structure: assuming the environment provides proper inputs does the transition system adequately produce the real system component behaviour? Second, does the environment provide proper input such that if the component is modeled right it produces the real system behaviour? Interface automata capture theses environmental assumptions, assert that the environment provides proper inputs and captures the I/O behaviour of a component. Refinement relations establish this replacing of system components with model components without any compatibility issues with other components and it can be approximate. The level of approximation is fidelity measure i.e. measure of violation in SUT environmental assumption by models. In the next section, notions of refinement are introduced informally with respect to abstraction.

5.3.1.1 Refinement & Fidelity Quantification

It may be recalled from section 4 that the relation between systems behaving similarly is given by classical simulation preorder relations. In [Alfaro,2001], Alfaro extends these relations to open systems through the game perspective for system refinement and composition via alternating simulation preorders. Alternating simulation is defined as the relation between states of two systems A and B such that, at related states, all the outputs that can be generated by A can be generated by B and all the inputs that can be accepted by B can be accepted by A [Alur,1998]. Alternating simulation preorders helps in establishing refinement relations between systems. Intuitively a system A is said to refine system B when B can be replaced by A. In other words, refinement relations refer to behaviour containment and this game theoretic notion leads to a uniform framework in the synthesis or verification of transition systems.

In general, refinement, which is a top down approach of interface based design approach is opposite of abstraction, which is a bottom up approach of component based verification. It may be recalled that, a simulation model component is an abstraction of system component and can replace systems if it can reproduce all the system behaviour. Refinement relations are better suited to establish this relationship between two transition systems than trace inclusion or simulation relations as shown in [Alfaro,2001]. Considering a relation between a system model and its representation i.e. a simulation model, the notions of trace inclusion states all behaviours of

simulation model are included in the behaviour of system model. This assumes all input behaviour of simulation model are subset of the system model and this notion is restrictive for reactive systems which interact with its environment. Instead, a contravariant refinement with respect to inputs and outputs via alternating simulation relations which plays the same role of simulation between transition systems. Thus, instead of classical refinement relations between a specification and its implementation discussed in the literature, this study extends them into M&S i.e. between simulation model and the system model. Let us consider a simple example below where the system model and simulation models are given.



(a) M_{sys} (b) M_{sim}^{1}

Figure 5.6: Trace Inclusion vs Refinement principles

It can be seen that the trace inclusion relations do not hold. However, the simulation model refines the system model as the simulation model accepts as many input behaviour as system and for same input behaviour for two systems simulation model produce a subset of system model output behaviour. Such relations are useful if the V&V requirement is to observe output d! followed by input *a?* and *c?* which both models satisfies. On the other hand, if the requirement is only to observe *b!* followed by *a?* input then the simulation model can be abstracted by removing *c?* and *d!* while still holding the alternating trace containment limited to the validation objectives though not universally. This local refinement or trace containment notion helps in replacing system models with simulation models locally or 'relative' to the objectives. The degree of refinement is given based on a distance notion between system and simulation models and this distance notion is dependent on the requirements. In other words, globally a simulation model could be far from representing the system but it may be adequate to represent the system for a particular requirement.

This approach is better illustrated with the following example. Consider a V&V activity on a simple controller whose function is to supply fluid to a hydraulic system by opening or closing valves. It has a push button interface which can either be 'on' or 'off' and a light interface which is illuminated only when the button is pushed 'on' to indicate the status of controller. The completed or receives a message *nack?* and reports *FAIL!* in case of failure. For the sake of redundancy there is a backup valve that will be used only if the valve is failed to open and this is commanded with *open backup valve!* The system model is illustrated in figure 5.7.

Let us consider three candidate simulation models already available or supplied by the vendors as illustrated in figure 5.8. Consider a test scenario where the objective is to verify the time delay between the user action on the push button and the corresponding status of the light. Intuitively, the model M_2 seems to be highly representative of the system. However, for the given objective, this model is too detailed whereas the third model is poorly representative as it does not have output corresponding to the light. But the first model is adequately representative though it is poor in absolute terms.

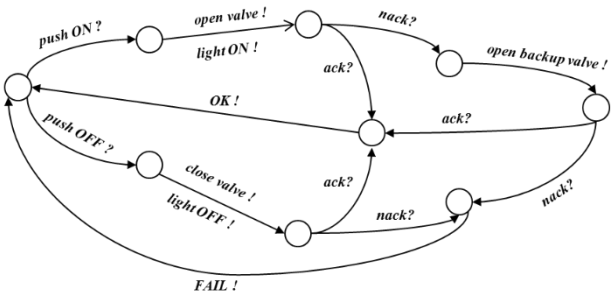
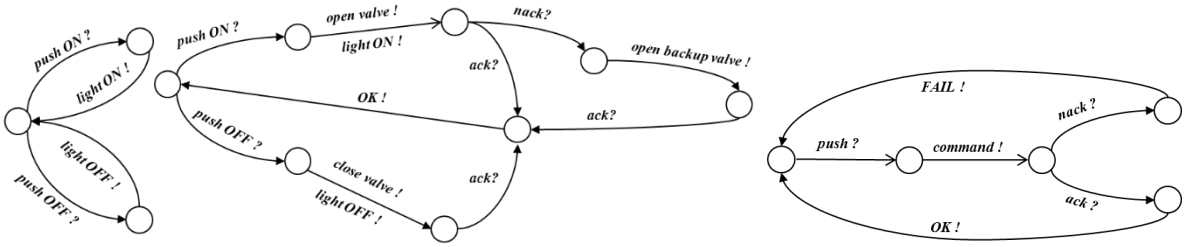


Figure 5.7: Controller System model



(a) M^1_{sim} (b) M^2_{sim} (c) M^3_{sim}

Figure 5.8: Controller Simulation Models

Let us consider another scenario where the objective is to verify the failure output in case of receiving *push OFF?* and *nack?*. Clearly the first model is not useful, the second model does not model the failure output whereas the third model does not differentiate between the open and close scenarios. The third model is better with respect to this requirement, however the requirement does not explicitly state failure in valve closing or opening. In this case, the second model is better though it does not model the failure output. In simple words, fidelity of simulation depends on the intended use of model and this intention of usage is imperative to perform V&V activities on the SUT. This example can be illustrated better with a distance notion on interfaces proposed by Cerny et al in [Cerny,2014].

In this simple example, the SUT i.e. the controller does not have an environment and the question is on the granularity of the model vis à vis its requirement. However, one can imagine this question becomes further complicated with the presence of environmental models whose granularity needs to be designed such that it is representative enough for the SUT validation. In other words, to validate this controller for a scenario, what level of abstraction needs to be chosen in its environmental models such as models of valves, switching logics, communication channels etc. at their respective interfaces? In this case it may be reminded that the environmental models are abstracted with respect to the V&V objectives of the SUT.

Thus, in addition to the component based design (or verification) perspective which is the existing norm, an interface based design perspective which is increasingly discussed in literature [Alfaro,2004],[Benviste,2015] especially in the embedded systems design ensures fidelity by capturing the degree of environmental assumption congruence between different interacting models. Thus the perspectives of verification and refinement could be seen as complimentary to

each other in the V&V process. Despite the abundance of academic work in this area of *contract based design*, such a paradigm is still in its infancy especially in the industry. In the following sections, the problem of fidelity quantification is extended to this formalism as a two player game between the system and its model formalized as an untimed interface automata. Refinement relations between system and simulation experimental frames were presented based on alternating simulation relation [Alfaro,2004] and its approximations for interface automata [Cerny,2014]. The distance notion of interfaces is presented in the context of simulation fidelity and a simulation objective dependent weighting is proposed to measure this distance similar to section 5.1. This game-theoretic distance notion is extended to generating quantitative reachability between the system and simulation model specified as untimed interface automata in the ProDEVS/TINA tool similar to sections 5.1-2.

5.3.2 Interface Automata & Experimental Frame

An experimental frame, recalling from section 2 of chapter II, is composed of different components and in such a component based design framework it is important to characterize the behaviour at the input and output interfaces of the components. Interface automata proposed in [Alfaro,2001] is one such formalism used to capture the temporal aspects of software component interfaces. In this study, it is extended to EF components in the context of simulation fidelity through refinement relations between two interface automata. An interface automata is a deterministic labeled transition systems in which the labels correspond to input and output actions. In other words, unlike classical automata an interface automata segregates and models the component's visible and internal behaviour explicitly.

5.3.2.1 Interface Automata & Alternating Simulation Relations

Formally, an interface automata [Alfaro,2001] extends a finite state automata with input and output actions and is defined by the following tuple, $T^{IA} =< X, X^0, \tau^I, \tau^O, \delta >$, where X is the finite nonempty set of states, $X_0 \subseteq X$ is the initial nonempty state set, two disjoint sets $\tau^O$and τ^I referring to output and input actions, transition function is defined by $\delta: X \times \tau \rightarrow 2^X$ with $\tau = \tau^I \cup \tau^O$. In general, denoting an action as $\sigma \in \tau$, a transition from state $x_1 \in X$ to a state $x_1' \in X$ can be written as (x_1, σ, x_1').

Alternating simulation relations extends simulation relations for alternating transition systems. For two transition systems described by interface automata, $T_1^{IA} =< X_1, X_1^0, \tau_1^I, \tau_1^o, \delta_1 >$ and $T_2^{IA} =< X_2, X_2^0, \tau_2^I, \tau_2^o, \delta_2 >$, alternating simulation of T_1^{IA} by T_2^{IA} by $T_2^{IA} \preccurlyeq_{A/S} T_1^{IA}$ holds if there exists a binary relation $f \subseteq X_1 \times X_2$ such that if $(x_1, x_2) \in f$ then

- $\forall\ (x_1, \sigma_1^I, x_1')\ \ \exists\ (x_2, \sigma_2^I, x_2')$ such that $(x_1', x_2') \in f$ (19)
- $\forall\ (x_2, \sigma_2^O, x_2')\ \ \exists\ (x_1, \sigma_1^O, x_1')$ such that $(x_1', x_2') \in f$

In the next section, we present EF in interface automata formalism as it communicates with SUT through interfaces. Alternating simulation relations for interface automata is then presented to establish refinement relation between EF in the context of V&V by simulation.

5.3.2.2 Experimental Frame & Refinement

Let us denote the EF interface automata as

$$EF = < X, X_0, \tau^I, \tau^O, \delta > \tag{20}$$

where X is the finite nonempty set of states, $X_0 \subseteq X$ is the initial nonempty state set, two disjoint sets $\tau^O$and τ^I referring to output and input actions, transition function $\delta: X \times \tau \rightarrow 2^X$ with $\tau = \tau^I \cup \tau^O$. The EF is input deterministic and internal actions omitted in the definition which follows the broadcast interface automata proposed in [Alfaro,2001]. The input and output actions are given with the '?' and '!' sign respectively.

The idea of refinement discussed in previous sections can well be extended to EF, consider the composition of experimental frame with the SUT, recalling the illustration in section 3 of chapter II,

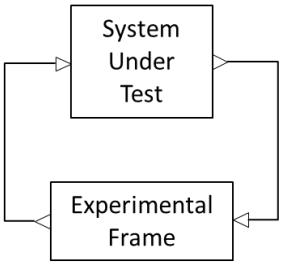


Figure 5.9: EF & SUT Composition

Extending the notions of refinement to EF, the objective is to replace the original EF made of real systems with the simulated EF made of models of the systems. In other words, simulation EF refines the system EF and we define this refinement relation in terms of alternating simulation relations as described in section 5.3.2.1. In this context, the following definitions are presented.

Definition 5.1: A simulation is said to be representative when $EF_{sim} \preccurlyeq_\varepsilon EF_{sys}$ *where* $\varepsilon \leq \varepsilon_\varphi$

Following the refinement relation definition in section 5.3.1.1 it is straight forward to see the connection between fidelity and refinement.

Definition 5.2: A simulation model is said to be representative when $M_{sim} \preccurlyeq_\varepsilon M_{sys}$ *where* $\varepsilon \leq \varepsilon_\varphi$

The simulation is said to be representative i.e. with sufficient fidelity if the value of the game described similar to the one described in section 4.2 is less than the fidelity tolerance, ε_φ.

Definition 5.3: *The simulation experimental frame refines the system experimental frame is denoted by*

$$EF_{sim} \preccurlyeq EF_{sys} \tag{21}$$

if the following holds true

$$\tau^I_{EF_{sys}} \subseteq \tau^I_{EF_{sim}} \tag{22}$$

$$\tau^{o}_{EF_{sys}} \supseteq \tau^{o}_{EF_{sim}}$$

and there exists $EF_{sim} \preccurlyeq_{A/S} EF_{sys} \mid x^{0}_{EF_{sim}} \preccurlyeq_{A/S} x^{0}_{EF_{sys}}$

This comes from the following applicability definition which states a precondition for using a simulation EF in place of system EF,

Definition 5.4: An EF_{sim} is said to be applicable to SUT if it refines EF_{sys}

From applicability conditions of Eq.(7) of chapter II, for inputs and outputs to be compatible with SUT, the EF_{sim} must accept as many inputs as EF_{sys} and when subjected to same scenario i.e. input it must produce a subset of output behaviours of EF_{sys}. From the perspective of SUT, it receives only what is being produced by EF whereas the EF receives what is produced by SUT. This compatibility conditions ensures EF produces fewer outputs than the SUT can accept and this compatibility applies to simulation EF as well.

Definition 5.5: An EF_{sim} refines EF^{2}_{sys} if $\beta(EF^{1}_{sys})=EF^{2}_{sys}$ and $EF_{sim} \preccurlyeq EF^{1}_{sys}$

This statement refers to scenario inclusion since derivability is relation between different EF's i.e. scenarios. It means a simulation EF can replace a new system EF provided it is derived from an EF which is refined by simulation EF. In other words, the same simulation EF can be used instead of system for a scenario which is shown to be a subset of the original scenario.

5.3.2.3 Alternating Simulation Games for System Model Refinement

From section 3 of chapter II, an EF is composed of different components $EF = \sum\{M^{i}\}$ where $i=1..N_c$ be number of components. Let us recall the notions for them by a generic word M^{i} and if the component is real system it becomes M^{i}_{sys} and its representation for simulation is M^{i}_{sim}. Conditions of compositionality are given in [Alfaro,2001] and due to the composition, the refinement relation extends to the component models as well and it becomes,

$$M^{i}_{sim} \preccurlyeq M^{i}_{sys} \tag{23}$$

The refinement relation given above could be characterized by two player alternating simulation on the system model and simulation model. Let those two models be represented by an interface automata as follows

$$\begin{aligned} M_{sys} &=< X_{sys}, X^{0}_{sys}, \tau^{I}_{sys}, \tau^{o}_{sys}, \delta_{sys} > \\ M_{sim} &=< X_{sim}, X^{0}_{sim}, \tau^{I}_{sim}, \tau^{o}_{sim}, \delta_{sim} > \end{aligned} \tag{24}$$

The simulation model refines system model, $M_{sim} \preccurlyeq M_{sys}$ when the following holds true

$$\begin{aligned} \tau^{I}_{sys} &\subseteq \tau^{I}_{sim} \\ \tau^{o}_{sys} &\supseteq \tau^{o}_{sim} \\ M_{sim} &\preccurlyeq_{A/S} M_{sys} \mid x^{0}_{M_{sim}} \preccurlyeq_{A/S} x^{0}_{M_{sys}} \end{aligned} \tag{25}$$

Then, let the game be defined as follows,

$$\mathfrak{g}_{M_{sys},M_{sim}} =< X, X^1, X^2, E, x_0 > \tag{26}$$

where $X = X^1 \cup X^2$ and let some $\{x_1 x_2\} \in X_{sys}$, $\{x_1' x_2'\} \in X_{sim}$ and $\sigma_{sys}^{I,o} \in \tau_{sys}^{I,o}$ and $\sigma_{sim}^{I,o} \in \tau_{sim}^{I,o}$
Then the attacker can choose either input from system or output from model given by:

- Input from system such that : $x_1 \xrightarrow{\sigma_{sys}^{I}} x_2 \in \delta_{sys}$
- Output from model such that : $x_1' \xrightarrow{\sigma_{sim}^{o}} x_2' \in \delta_{sim}$

whereas the defender can choose either input from model or output from system given by:

- Output from system such that : $x_1 \xrightarrow{\sigma_{sys}^{o}} x_2 \in \delta_{sys}$
- Input from model such that : $x_1' \xrightarrow{\sigma_{sim}^{I}} x_2' \in \delta_{sim}$

This refinement relations can be extended for composition as well, considering that two models are composed together to build a EF, $EF = \sum\{M_{sys}^{1,2}\}$ such that input of M_{sys}^{1} is the input of EF and output of M_{sys}^{2} is the output of EF. Denoting the composition between the models by composition operator $\oplus$, then $M_{sys}^{1} \oplus M_{sys}^{2}$. The compositionality principle [Cerny,2014] states,

$$M_{sim}^{1} \oplus M_{sim}^{2} \text{ if } (M_{sim}^{1} \preccurlyeq M_{sys}^{1}) \wedge (M_{sim}^{2} \preccurlyeq M_{sys}^{2}) \tag{27}$$

Thus an EF can be composed of simulation models replacing system models if the refinement relations hold between them. In the next section, instead of exact refinement an approximate refinement is explained with approximate alternating simulation in the global fidelity quantification as well as in a V&V context.

5.3.2.4 Experimental Frame Approximate Refinement

In [Cerny,2014], similar to section 5.1-2, boolean notions of interface refinement through alternating simulation preorder is improved to a quantitative notion where the distance between interfaces called interface simulation distance has been proposed. This directed metric is based on alternating simulation game and properties such as triangle inequality, notions of over or under abstraction were discussed. These approximate refinement relations are denoted by $\preccurlyeq_{\varepsilon}$, where ε be the degree of approximation such as limited average in Eq.(7,8). It can be seen that when $\varepsilon = 0$, it becomes exact alternating simulation relation and for increasing $\varepsilon > 0$, the alternating behavioural inclusion becomes lesser. Similar to section 4.2, the two player game perspective where the players cheat and thus incurring a penalty is extended to refinement of systems modeled as interface automata in which the game is played on the system and its abstraction. In this game, the attacker can choose either the output transition of simulation model or the input transition of system model. On the other hand, the defender can choose either the output transition of system model or the input transition of simulation model. The error quantification semantics of this turn based game is similar to the timed and untimed classical automata games and hence not discussed in detail here. The quantitative reachability graph is generated similarly as well and the implementation is discussed in sections 6.2.

Similar to section 2.2, let us illustrate the need for a differential weighting alone by initially applying the absolute weighting method by assigning a positive but equal weight of 1 for every cheating transition of the defender and measure the value of game by a limit average of errors. Consider game between system model in figure 5.7 and simulation model M_{sim}^{1} in figure 5.8, the attacker takes input *push ON?* and moves to next state. The defender responds by taking the same input and the error is 0. For the next move, the attacker chooses *light ON!* for which the defender responds by same action and the error is still zero as the defender matches every attacker's move. However, for the third move attacker chooses input *nack?* for which the defender has no choice but to cheat and moves to next state. This continues and the error keeps accumulating with the net error shown at the end of one full iteration is shown in table 5.1. On the contrary, consider the second model, M_{sim}^{2}, the cheating occurs only at the fourth transition when the defender fails to respond to the choice *nack!* of attacker. Again the defender cheats when attacker chooses and the net error becomes 0.5. For the first requirement mentioned in section 5.3.1.1, both models are valid, however the first model is simpler despite its poor absolute fidelity.

Table 5.1: Equal weighted error of models

Step, *i*	$\varepsilon(M_{sim}^{1})$	$\varepsilon(M_{sim}^{2})$	$\varepsilon(M_{sim}^{3})$
1	0	0	1
2	0	0	1
3	1/3	1/3	2/3
4	2/4	2/4	3/4
5	3/5	2/4	4/5
6	4/6	2/4	5/6

Consider the second requirement of failure output on system, again the second model fares better than the third model in terms of representativeness. However, it can be seen that the key requirement of *FAIL!* output is not present in the second model though it is present in the third model. The second model is essentially insufficient to represent the required phenomena i.e. failure output and cannot replace the system. In contrast, the third model though cannot differentiate between the push button position, and valve position, is a reasonable abstraction of the system as it models the failure output on receiving *?nack* message as specified in the requirement.

In both these examples, it can be seen that the equal weighting for cheating transition is not adequate since some simple but more erroneous model cold be adequate enough for a scenario compared to a complex but less erroneous model. The cost must take into account this subtle relation of granularity with respect to the requirement. In other words, transitions related to requirements must be penalized more than the transitions which have no effect. To this effect, the error model is modified with a differential weighting to account for these differences and cost is given by a standard objective function such as limit average.

5.3.2.5 Differential Weighted I/O Error Model:

An input/output error model [Cerny,2014] based on the transition relevance to requirements is a function $E: \tau^{I} \times \tau^{I} \rightarrow \mathbb{R}_{0}^{+}$ for input and $E: \tau^{O} \times \tau^{O} \rightarrow \mathbb{R}_{0}^{+}$ for output. This needs to be a directed metric satisfying reflexivity and triangular inequality which states for all $\tau^{1,2,3} \in \tau^{O}$or τ^{I}, $E(\tau^{1}, \tau^{1}) = 0$ and $E(\tau^{1}, \tau^{3}) \leq E(\tau^{1}, \tau^{2}) + E(\tau^{2}, \tau^{3})$ respectively.

Let us denote the actions of interest on system model by $\tau^{\varphi} \subseteq \tau_{sys}$ where $\tau_{sys} = \{\tau^{I}_{sys} \cup \tau^{o}_{sys}\}$ and whenever the defender cheats on these actions it incurs higher penalty than when it is not. The error weighting function is given by $e: \delta \times N \rightarrow \mathbb{R}^{+}_{0}$ where $\delta = \delta_1 \cup \delta_2$ and N refers to number of transition. In the two player game with turns m=1,2 the distance is calculated at the end of every defender move i.e.$\forall 2n$ where $n \in N$ is the number of transition. The two different weights are denoted by W_1 and W_2 respectively which could either be a simple positive number or a function of transition $W_{1,2}(n)$.

$$
\begin{aligned}
\forall\, \delta_2 \in \delta_{sim},\ & \{\sigma^{I}_{sim} \neq \sigma^{I}_{sys} \wedge \sigma^{I}_{sys} \in \tau^{\varphi}\} \Rightarrow e = w_1(n) \\
& \{\sigma^{I}_{sim} \neq \sigma^{I}_{sys} \wedge \sigma^{I}_{sys} \notin \tau^{\varphi}\} \Rightarrow e = w_2(n) \\
& \text{else} \qquad\qquad\qquad\qquad\quad e = 0 \\
\forall\, \delta_2 \in \delta_{sys},\ & \{\sigma^{o}_{sys} \neq \sigma^{o}_{sim} \wedge \sigma^{o}_{sys} \in \tau^{\varphi}\} \Rightarrow e = w_1(n) \\
& \{\sigma^{o}_{sys} \neq \sigma^{o}_{sim} \wedge \sigma^{o}_{sys} \notin \tau^{\varphi}\} \Rightarrow e = w_2(n) \\
& \text{else} \qquad\qquad\qquad\qquad\quad e = 0
\end{aligned}
\tag{28}
$$

In particular, weights are assigned based on the moves on the system input or output. In order to explain the subtleties of this weighting consider the example in figure 5.7 and 5.8 where intuitively one can see that first model is better than the third since the desired output *light ON!(OFF!)* is not present in it. However, to quantitatively decide this, it is important to impart this knowledge into the error model via these weights. In the game when the attacker chooses output of model *command!* the defender cheats the original transition of system i.e. *light ON!* with this model transition. However, since it cheats on system transition it is checked with V&V objectives and assigned higher weight. The case for cheats on system inputs is straightforward.

Let us apply this approach with linear weighting such as in figure 5.1 to the two cases described before. For the sake of illustration, consider for every cheating move n_p, let the weight varies in steps of -0.1 for primary weight i.e. $w_1 = (1 - 0.1n_p)$ and +0.1 for secondary weight with each transition i.e. $w_2 = 0.1n_p$ such that $|w_1 + w_2|<1$. Consider case 2, where the objective is to verify FAIL! output for push OFF? and *nack?* input. The game commences with attacker input move on *push OFF?* for which the defender responds by cheating on *push?* input on simulation model M_3 incurring a cost 0.1. The attacker then chooses output *command!* on M_3 where the defender has to cheat with *close valve!* on system model with a penalty 0.2. However, for subsequent attacker moves the defender is able to match them and the error remains same. This is not true for model M_2 which does not model the *nack?* output and thus incurs high penalty 0.9 at third transition and further 0.8 at fourth transition for cheating on *FAIL!*. The same phenomenon can be seen in first model too and for this requirement the first models are not useful. For the case 1, however, both models can be used if the requirement is only until four transitions and beyond the second model is more representative though it does not contribute to the requirements but only being more absolutely representative.

The implementation of absolute fidelity quantification alone is presented in section 6.2 similar to other formalisms of sections 5.1-2 as the implementation of relative quantification is straightforward and our focus is more on generating a quantitative reachability to do further analytics. However, it must be noted that, the value of the game is computed in PSPACE time and dependent on largest weight used. In this case, due to the presence of two weights, the complexity

becomes $\mathcal{O}(|w_1||w_2||X|^3|E|)$ where the two weights are given by $w_1$and w_2, E are the number of edges and X be the number of game states.

Table 5.2: Equal weighted error of models

(a) Light ON/OFF case

Step, i	$\varepsilon(M_1)$	$\varepsilon(M_2)$	$\varepsilon(M_3)$
1	0	0	0.9
2	0	0	0.8
3	0.1/3	0.1/3	1.6/3
4	0.4/4	0.4/4	1.6/3+0.1/4
5	0.9/5	0.4/4	1.6/3+0.4/5
6	1.6/6	0.4/4	1.6/3+0.9/6

(b) Failure output case for OFF position

Step, i	$\varepsilon(M_1)$	$\varepsilon(M_2)$	$\varepsilon(M_3)$
1	0	0	0.1
2	0	0	0.4/2
3	0.9/3	0.9/3	0.4/2
4	1.6/4	1.6/4	0.4/2

6. IMPLEMENTATION

The game semantics described in previous sections has been implemented in ProDEVS, a DEVS simulation platform [Vu,2015]. ProDEVS is a Discrete EVent Simulation (DEVS) platform and amongst other features such as FMI co-simulation, it can also be used to do model automata (classical and interface) and perform formal verification with TINA toolbox [Berthomieu,2004]. TINA toolbox is used to the edit and analyse (Timed) Petrinets, an extension of classical Petrinet formalism [Peterson,1981] with firing time for the events and an extension of it with data handling called Time Transition Systems. Such a formalism is widely used to represent the timed execution of discrete event systems interleaved with (possibly zero) delays. DEVS is a more general case of the FSA formalism with embedded time and differentiation between input and output labels i.e. akin to interface automata but with time. Since we intend to extend the current quantitative approach to timed interface automata, and then further to DEVS formalism, we construct classical automata and untimed interface automata models in ProDEVS. On the other hand, the game semantics are modeled in (un)timed Petrinet formalism since Petrinets, with their token based formalism, are amenable to modeling such turn based games between two FSA. These games are automatically modeled in Petrinet which could be then visualized using the graphical editor of the TINA toolbox. In addition, using the TINA reachability generator [Berthomieu,2014] along with the data encoding in guards and actions of the underlying Petrinet transitions, the quantitative reachability graph could be generated. This graph is then parsed to perform some analytics for better understanding of the model fidelity.

Let us formally introduce Petrinet, a formalism widely used to represent the timed execution of discrete event systems interleaved with (possibly zero) delays. The timed petri-net is an extension of classical petri-net formalism with firing time for the events. Though Petrinets per se is a richer formalism to model transition systems due to its ability to model parallel processes, we will restrict our Petrinet models to FSA where the states are finite and have no parallelism. Formally, a Petrinet is a tuple defined as follows,

$$M =< P, \tau, A, w, p_0 > \tag{29}$$

- P is a finite set of symbols called places

- τ is a finite set of symbols called (timed) transitions with $P \cap \tau = \emptyset$

- $A \subseteq (\tau \times P) \cup (P \times \tau)$ is the set of arcs defining the flow relation

- w: $A \rightarrow N$ is the function defining the respective weights of the arcs, N=1 in our case

- I_s: $\tau \rightarrow I^+$ is static interval function with I^+, the non-empty set of positive real intervals including 0.

- $p_0 : P \rightarrow N$ is the initial marking

Informally a transition, τ is enabled if there is a token at the corresponding place, $p \in P$ and moves to the next state defined by the flow relation. This token and place formalism of Petri-net is amenable to model the two player turn-based game which is alternating in terms of player turns. In the current study no concurrency is assumed and the resulting games have only total states. A state, s of a Petri net is a couple $< m, I >$ where m is the marking and I is the interval function, $I: \tau \rightarrow I^+$ which associates to each enabled transition at marking m a temporal interval. In addition, only intervals under the form $[\theta, \theta]$, i.e. deterministic event timings are considered although firing at timings drawn randomly from uniform distribution is also possible.

It can be easily seen that a Petrinet with neither weights nor parallelism is a classical automaton and with time, it becomes a timed automaton. For the sake of brevity, let us denote places as states and markings denote the current state. Informally a transition, τ is enabled if there is a token at the corresponding place, $p \in P$ and moves to the next state defined by the flow relation. This token and place formalism of Petrinet is amenable to model the two player turn-based game which is alternating in terms of player turns. In the current study no concurrency is assumed and the resulting games have only total states.

The classical (un)timed games or untimed interface games are constructed in a single Petrinet file and could be run directly from the ProDEVS. Since Petrinet simulator per se does not handle data, these are encoded as guards and actions on the transitions through associated 'c' files to generate 'dll' files. The generated reachability graph is in text form and the data needs to be parsed for better understanding and visualization. The parser, written in JAVA and integrated in ProDEVS has many functions such as plotting the evolution of cheats along the play, distribution of cheats etc. The sample pseudo algorithm is presented in section 5 of annex. In particular, it constructs a reachability tree which can then be visualized. The replay feature allows to choose a particular cheat from the cheat distribution plot to see the associated path to better understand when and where the simulation model behaviour differs with respect to the system. The methodology is briefly given in the figure 6.1 [Ponnusamy,2016].

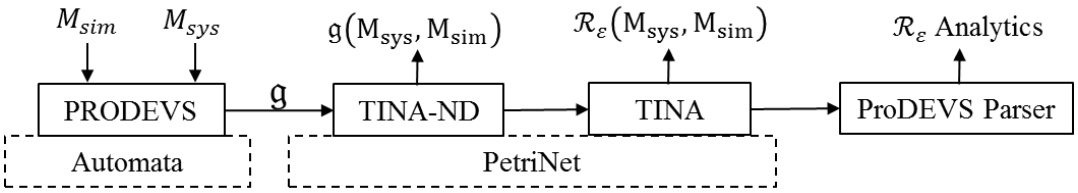
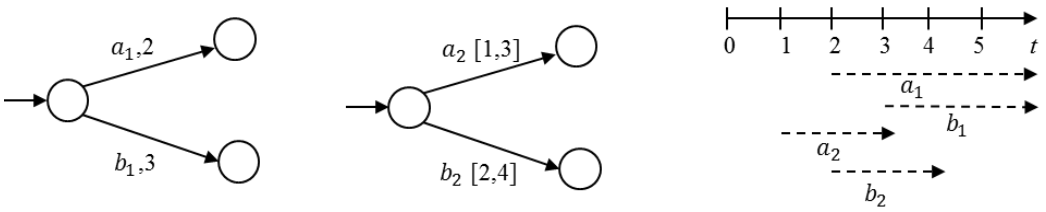


Figure 6.1: Implementation

It can be seen that the modeling and parsing are done in ProDEVS with rest being in TINA. Alternatively, the modeling and game can be done in TINA-ND graphical editor tool as well and the reachability is generated by TINA later. It may be seen that, given a system design model and a simulation model in same formalism, for example in timed automata, the game is constructed automatically and the resulting output is exhaustive error quantification over all possible transitions. The simulation user or the developer may then decide to improve the simulation model or relax the V&V requirements. This approach, apart from quantifying the global fidelity independent of V&V objectives, is also useful in iteratively refining the design with respect to V&V scenarios especially in the early system development when the design is not frozen.

6.1 UNTIMED & TIMED GAMES IMPLEMENTATION

The untimed games implementation is straightforward with turn semantics of open interval timed transitions i.e. [0,∞] to model the untimed behaviour. In addition, for the sake of simplicity only at exact i.e. deterministic event timings are considered although firing at timings drawn randomly from uniform distribution too is possible. In our timed games implementation, branching is taken into account in two different fashions. For the sake of illustration consider two automata, first one firing at exact time and second at an interval as in figure below. The first type of branching is straight forward where the earliest event is fired, for example in figure 6.2 it is always a_2 and in case of interval both during interval [2,3]. However, often according to the phenomenon both branches need to be explored. Intuitively, for example in figure 6.2.a, it means a state can make a transition either at 2 time units or 3 time units depending on the event and in such cases both branches are explored.



(a) M^1_{sim} (b) M^2_{sim} (c) Temporal Evolution

Figure 6.2: Branching in Timed Automata

This is implemented in Petrinet by the following mechanism of introducing an intermediary instant transitions ($\tau_{a_1 b_1}, \tau_{a_1}, \tau_{b_1}$) and states. The dark circle denotes the token and vertical bars denote the transition and when a transition is enabled the token is passed to the successor state. Further details on Petrinet formalism and its simulation can be seen in [Berthomieu,2014] whereas the game model can be seen in the section 3 of the annex. The semantics of this branching for M^1_{sim} in figure 6.2.(a) is illustrated in figure 6.3.

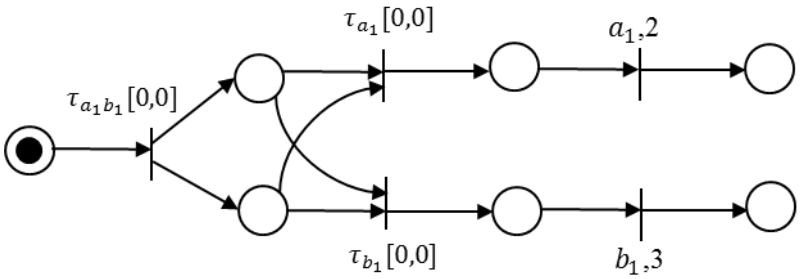


Figure 6.3: Branching Implementation

Denoting the number of original transition branches at place, p by n_b such a modification increase the complexity to $2n_b + 1$. For example, in figure 6.2.a there were two transition branches, n_b=2 which then increases to 5 transitions in figure 6.3. Similarly, the branching of figure 6.2.(b) can be constructed. Further details on play truncation, error estimation and reachability construction are illustrated along the application case in section 2 of chapter V and section 3 to 5 in annex. These timed and untimed games implementation is illustrated in the following figure.

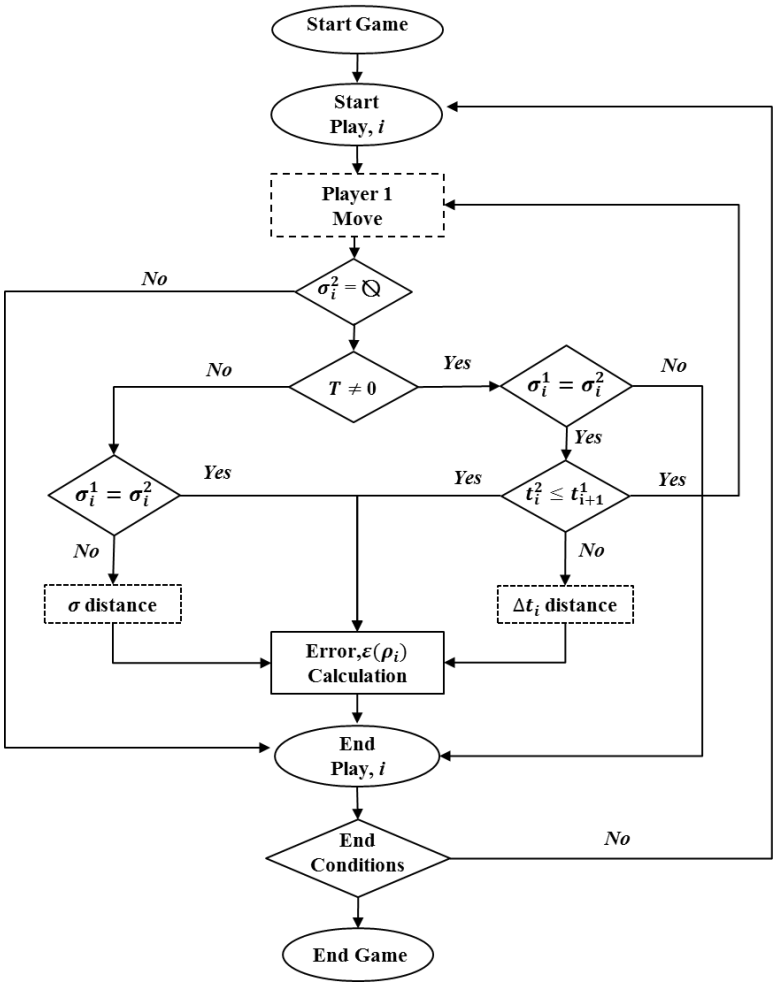


Figure 6.4: Automata Games Implementation

6.2 UNTIMED INTERFACE GAMES IMPLEMENTATION

In this game, there is a distinction between input and output transitions and similar to untimed automata game Player 1 starts the play and it is complete when player 2 finishes his turn. The only difference in semantics in addition to non-interface games there are identifiers for input and output types which are connected to input and output transitions of models respectively. The play is over if player 2 cannot have either an output transition on system or input transition on simulation model. The implementation is presented in detail in section 4.4 of annex along with the reachability constrution and analytics similar to previous sections in section 5 of annex. For the sake of brevity a flowchart similar to figure 6.4 is not presented.

A brief discussion on fidelity quantification in the context of infinite state systems i.e. continuous systems based on the principles of geometric control theory [Pappas,2003] can be found in section 6 of annex.

7. CONCLUSION

A formal quantitative approach to simulation fidelity based on simulation relations and two player game is presented is presented in detail for discrete systems and with some preliminary theoretical results for continuous systems in section 6 of annex. Broadly the contribution is threefold, first, extending timed games into a fidelity problem, implementing this game in Petrinet formalism for discrete systems (timed and untimed), generation of quantitative reachability and analysis with some fidelity metrics. A key possibility with such quantitative reachability graphs is to utilise efficient graph search algorithms to analyse for the shortest or optimal traces which gives further insight into the simulation model behaviour with respect to the system behaviour.

However, this explicit enumeration of traces along with their (timing) distances may suffer from the curse of dimensionality and of limited use in large scale systems. This may be mitigated by using efficient data structures such as using Binary Decision Diagrams (BDD) and studies need to be made in abstraction, and abstraction refinement techniques, especially for continuous systems. The abstraction problem could be posed as a planning problem of reaching a set of states by taking advantage of the BDD based symbolic methods. For continuous systems, discussed in section 6 of annex, the problem of finding abstraction maps which preserves bisimulation property as well as compliant with fidelity tolerance requirements was identified as a controlled invariance problem to ensure bounded distance. Further research is needed in better understanding this relation between abstractions and end use objectives. In addition, efficient parsing techniques for data analytics such as visualization needs to be developed since this requires reconstructing an explicit reachability graph and exploiting it further which could be prohibitively expensive especially for large scale systems. These challenges and future work including some perspectives on on-going work on model synthesis are further discussed in the section 2.2 of chapter VI.

Despite the challenges ahead both academic and industrial, this quantitative perspective will enable different stakeholders in the system V&V process to develop and reuse models with a known and assured level of fidelity. For example, the model developer could gain key insights into the model behaviour and chooses the best abstraction of the system vis à vis the scenario. On the other hand, the system test team would have a measure of fitness on the models being used for the V&V which would mitigate unfeasible or unclear model fidelity requirements. In addition, this would

benefit the system designer in making improvements or modifications to the system model. These benefits would allow not only to select a consistent model with sufficient level of fidelity according to the test case with different criteria such as performance, robustness etc. but also to help in quantifying the fidelity of the overall V&V process. Such a quantitative framework to fidelity will enable significant benefits in avoiding redundant modeling and validation effort thereby saving cost and time in product development especially in replacing real tests with simulation i.e. virtual testing.

CHAPTER V

APPLICATION CASE STUDIES

In this chapter, the case studies for the semi-formal and formal approaches are presented. The case study for the ontology based semi-formal approach is a real industrial case study from Airbus whereas formal approach is demonstrated with academic case studies.

1. DOMAIN MODEL APPROACH CASE STUDY - AIRCRAFT NACELLE ANTI-ICE SYSTEM

The principles of ontology based domain model approach in building a MR which explicitly takes fidelity requirements has been presented in chapter III along with the domain model development in Protégé tool. A real life industrial case study needs to be taken to demonstrate the feasibility of such an approach in technology readiness for the industry. We have considered different case studies: engine failure case modeling, control system V&V and aircraft engine Nacelle Anti-Ice System (NAIS). The NAIS was chosen due to its relative simplicity compared with other highly complex systems, data availability and other industrial constraints.

A generic description of the aircraft Nacelle Anti-Ice System (NAIS) is presented followed by instantiation of the domain model built from the ontology defined in section 4.1-3 of chapter III.

1.1 NAIS EXPERIMENTAL FRAME

The NAIS is an aircraft system which used to prevent ice accretion at the engine nacelle inlet by blowing hot gases from the engine exhaust. The system is activated whenever aircraft flies at icing conditions and this system is comprised of controllers, valves, solenoids, ducts etc. The NAIS system is connected to other aircraft systems such as Flight Management System (FMS), engine monitoring system, Full Authority Digital Electronic Control (FADEC) etc. The system is (de)activated by the pilot using the push buttons, P/B in the cockpit panel which sends the signal to the NAIS monitor and the underlying software validates the command and then sends it to the controller, C. The controller according to the feedback from pressure transducer, PT energizes or de-energizes the solenoid, $S_{1,2,3}$ which the then opens or closes the valve $V_{1,2}$. The flow control through valve changes the downstream pressure which is monitored by the pressure transducer.

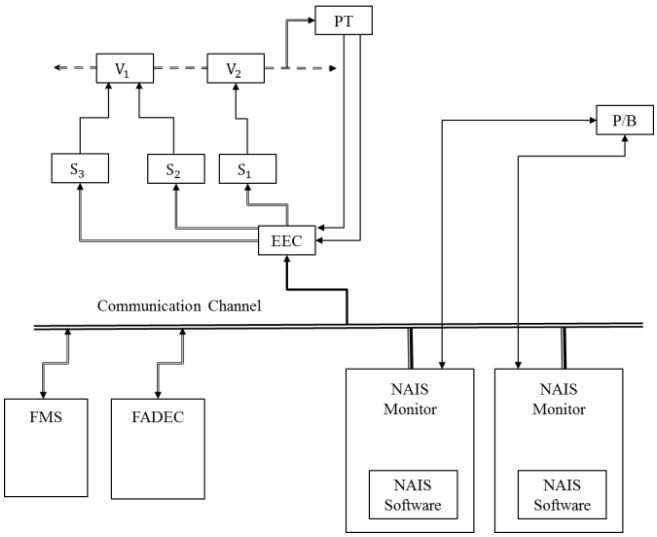


Figure 1.1: NAIS System

Let us consider a V&V activity where this system or a component of it needs to be validated against some test scenarios laid out according to the V&V plan in different test benches. In order to perform these tests on a component(s), controller of NAIS in our case, the problem is selecting elements of NAIS (e.g.: valves) and the associated systems (e.g.: Flight Management System), environment (e.g.: engine) with respect to this component(s) and its scenario. The figure 1.2 [Ponnusamy,2016] illustrates, albeit in an abstract sense, this experimentation in EF formalism where the environment representing the context under which the controller will be tested. The general system interaction is shown by solid line and the scenario specific observability of phenomenon (e.g.: pressure data from sensor) is denoted in dotted line. This EF description helps in a lucid visualization of what is being tested and what is needed for the test in addition to how it is tested (controllability) and what is expected of the test (observability).

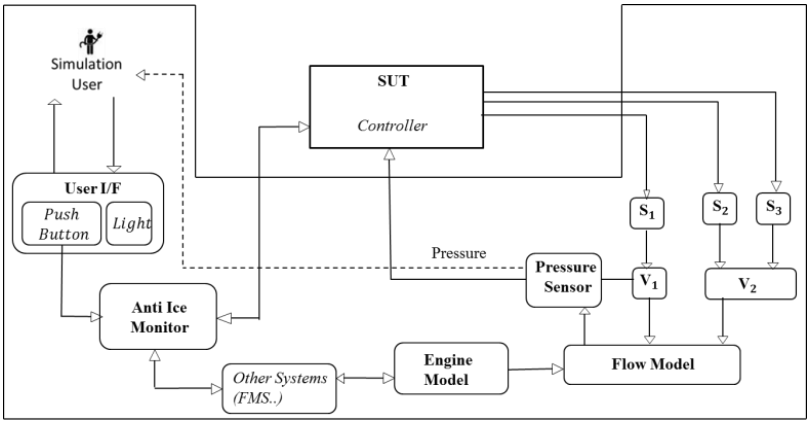


Figure 1.2: Experimental Frame of NAIS Controller

1.2 MR CONSTRUCTION

Recalling figure 5.1 of chapter III, the first step in MR construction is the domain model instantation from SD and TR body of knowledge.

1.2.1 Formalization: Domain Model Instantiation

Following the process described in section 5.1 of chapter III, the TR and SD are converted into domain model instances. Let us denote, the domain model concept and its corresponding instance by notation, $\mathfrak{E}:\mathfrak{I}$ and the relationship between concepts by $\mathfrak{E} \xrightarrow{\mathfrak{r}} \mathfrak{E}$. For example, from SD architectural descriptions of controller connected to solenoids (*electronic* type equipment), $S_{1,2,3}$ through different channels, this textual info is translated as *Equipment:controller* $\xrightarrow{isStructConnectedTo}$ *Equipment:*S_1). This process is repeated and the list of instances is shown in figure 1.3. For the sake of clarity, instances of SD and TR are noted by prefixes 'SDD_' and 'LTR_' respectively.

The instances in bold are defined with explicit relationships with other individuals or classes or both, for example *Designer:Designer-EYAK2* $\xrightarrow{designs}$ *System:SDD_NAI*, whereas the others are simply defined without any associations. This is particularly useful in dealing with complex TR and SD where only minimum instances need to be defined explicitly with the other instances are inferred using the reasoned and associated with properties or classes if there exists an implicit relationship. Such inferred instances or classes or relationships are shown as highlighted instances in figure 1.4. For example, *Equipment:LTR_EEC* $\xrightarrow{isStructConnectedTo}$ *Equipment:LTR_Solenoid_A* refers to the structural connection between 'EEC' and 'solenoid_A' in the LTR. In addition, another instance 'PRSOV_A' is related to this 'solenoid_A' through *Equipment:LTR_PRSOV_A* $\xrightarrow{isStructConnectedFrom}$ *Equipment:LTR_Solenoid_A*. This entails that LTR_EEC is connected to PRSOV_A as well since the underlying property 'isStructConnectedTo' (which is inverse of 'isStructConnectedFrom' and vice versa) is defined as transitive property [Protégé]. Similarly, other properties can be defined in the domain model according to the knowledge being formalised. Some such concepts and relationships can be found in the annex along with some metrics on the domain model in general and the application case in particular.

It may be reminded that neither SD nor TR expressed in natural language can be translated in its entirety and the goal is to translate them only to the extent possible. In case of TR which usually has different test some of may subsume each other (derivability and applicability definitions in sections 3.2 of chapter II). Thus, for the sake of practicality, the most complicated test case is taken and translated into corresponding domain model instances and in our study such cases only where chosen resulting in a total of about 92 instances.

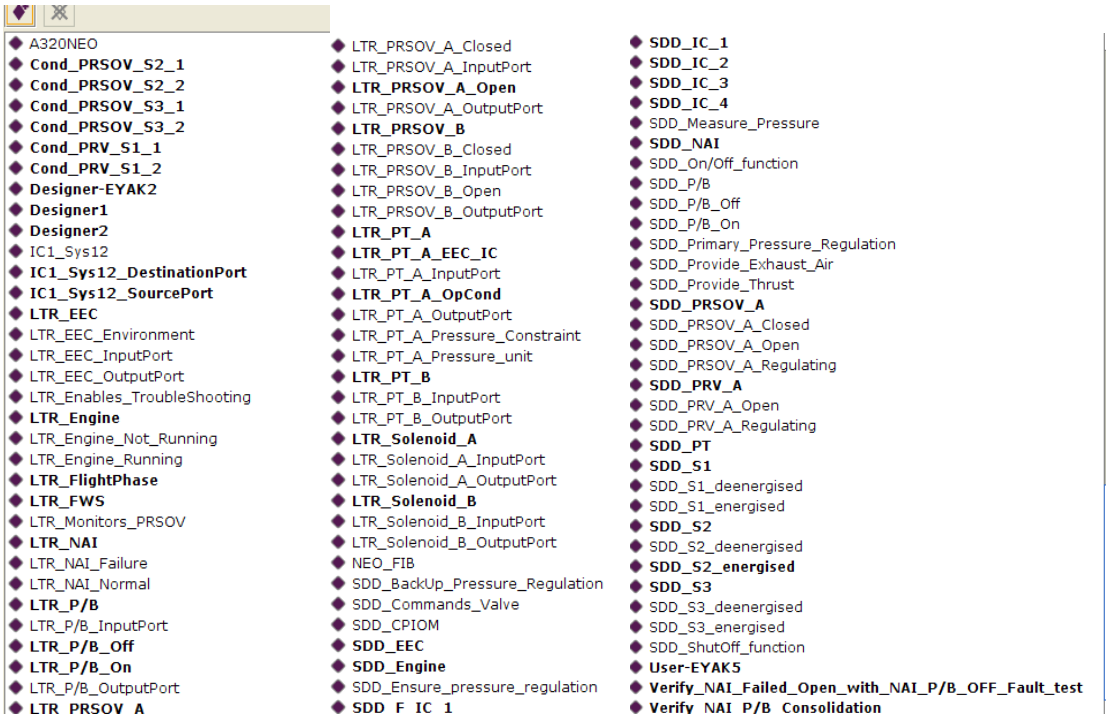


Figure 1.3: NAIS TR & SD Instances

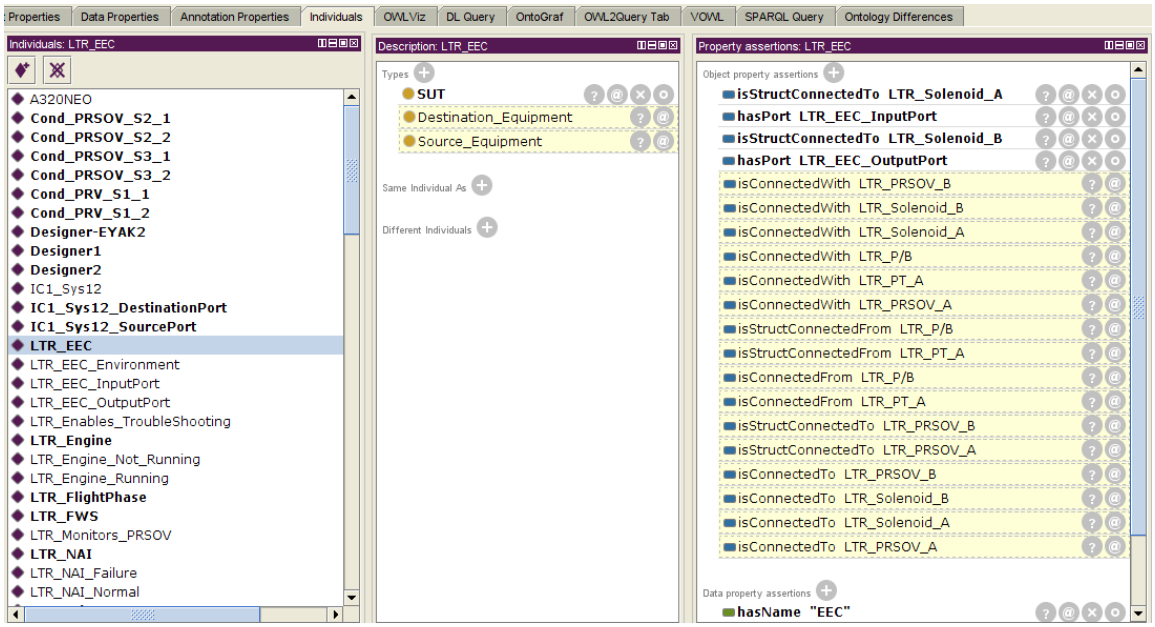


Figure 1.4: NAIS TR & SD Inferred Instances

1.2.2 Verification & MR Extraction

The phase after formalization of the SD and TR knowledge as domain model instances is verifying whether instantiations are consistent according to underlying domain model rules. The inbuilt reasoner of Protégé such as Fact++ [Fact++], Pellet etc. is used to evaluate this consistency which identifies reasoned instances in addition to defined instances. For example a relation, *Equipment:FMS* $\xrightarrow{performs}$ *Function:Manage Flight* is deemed inconsistent as only a concept, $\mathfrak{E}$ named System can perform a function i.e. *System* $\xrightarrow{performs}$ *Function*.

The figure 1.5 illustrates some of the instances with respect to their classes and the relation with other class instances. For example, the class *Multi_System_Function* has different instances both from TR and SD i.e. *LTR_** and *SDD_**. Such information can be efficiently extracted and manipulated using SPARQL queries. Let us show this by taking a sample instance *SDD_NAI* of class *System*. From various instances and their relationships defined and reasoned, it can be easily seen that this system is designed by the system designer *EYAK5* and performs a function, *SDD_Ensure_Pressure_Regulation* of class *Multi_System_Function.* In addition, this function needs a function *SDD_Provide_Thrust* performed by other system *SDD_Engine* which in turn designed by another designer. These dependencies are show by dotted rectangle in the figure and this not only helps to check the explicit consistencies but also extract implicit elements through reasoning and evaluate the consistencies.

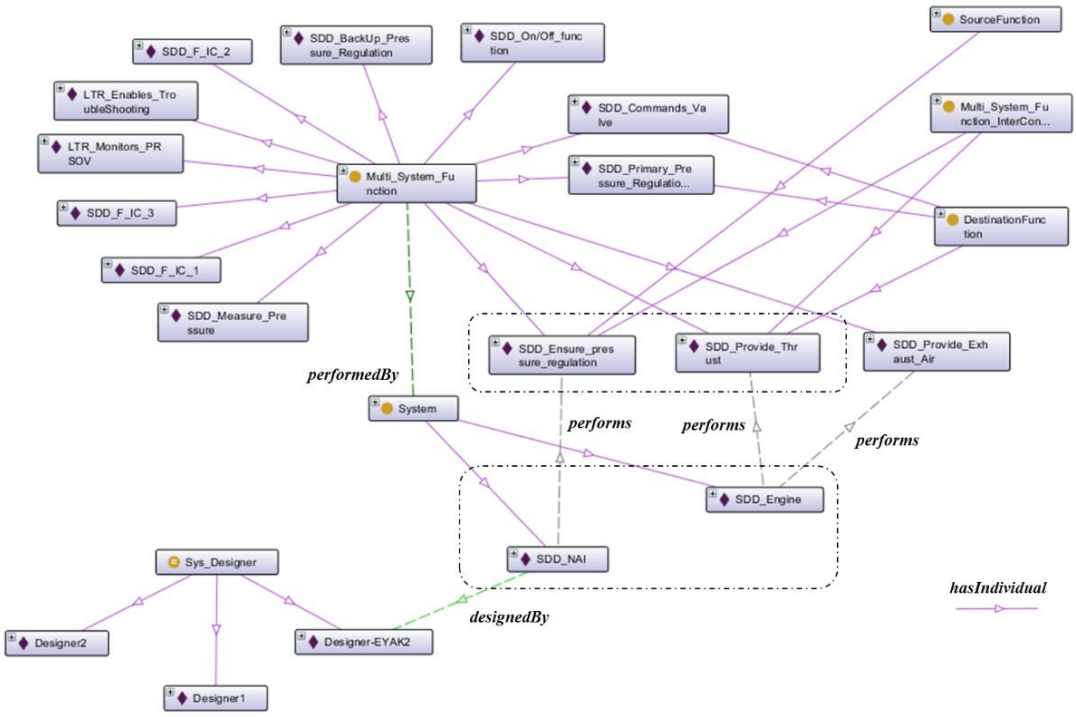


Figure 1.5: NAIS Instances Design Space Exploration

In addition, the verification and/or design space exploration to extract MR can be done using SPARQL queries where instances of particular concept denoted by $\mathfrak{T}(\mathfrak{E})$ or having a relationship

$\mathfrak{T}(\mathfrak{E}) \xrightarrow{\mathbf{r}} \mathfrak{T}(\mathfrak{E})$ can be queried, matched etc. The application of queries on verification and design space exploration is presented below.

1.2.2.1 Verification

A simple case of verification is checking whether the SUT required in TR exists in SD according to some criteria such as type, maturity level, etc. In our case, for the sake of illustration, the SUT defined in TR i.e. controller denoted by 'LTR_EEC' and having a name as EEC is matched with an instance in SD having the same name. If the SD has an instance, the associated system it belongs to and the designer of that system is extracted. From the results it can be seen that SD has a controller denoted by instance 'SDD_EEC' with the same name as TR instance and the system it belongs is 'SDD_NAI' designed by 'Designer_EYAK2'. Thus the query can be customized according to the user requirement.

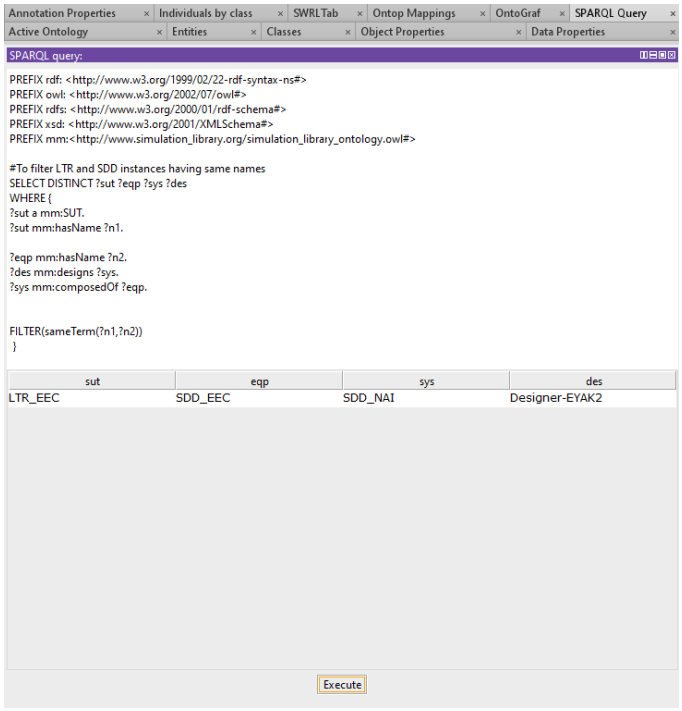


Figure 1.6: SUT Consistency Evaluation

An important but often overlooked aspect is comparing the requirements and specification instances before exploring the design space to extract necessary modeling elements to be included in MR. A key example is illustrated in figure 1.7, consider the real SD and TR instances depicting the structural connections to the SUT i.e. controller, C. In practice due to the unavailability of TR it is a common practice to utilize an available equivalent TR of a similar system. In the current study such a TR was used along with the SD of the actual system. It can be seen that, on comparison the architecture of two systems were different, for example only two solenoids were present in TR whereas SD has three and so are their respective connections. This by extension have implication on building MR since a test case defined for an element present in TR but not in SD is spurious and conversely the test case for elements in SD but not in TR cannot be directly extracted which

needs further deliberation with the stakeholders. Such information can be seen by querying the instances and comparing one to one before the MR extraction through design space exploration phase.

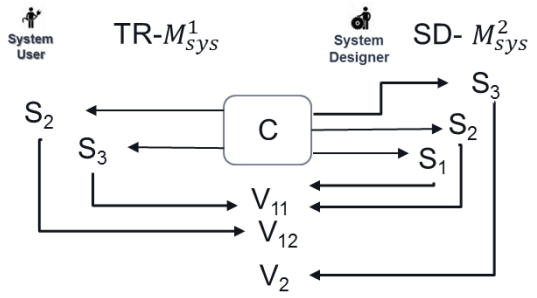


Figure 1.7: Architectural Comparison

1.2.2.2 Design Space Exploration

The second verification step is followed by superimposing the SD and TR instances which could be done by linking the required SUT from TR, $\mathfrak{T}(\text{SUT}_{\text{Req}})$ and the available SUT from SD, $\mathfrak{T}(\text{SUT}_{\text{Avb}})$. The associated elements are extracted either manually or through pre-defined queries. An example of design space exploration is given in this section. Let us consider a test scenario which states 'the controller, C must open valve, V_2 in time t_1s and close valve in t_2s'. The controller and valve in TR are denoted by 'LTR_EEC' and 'LTR_PRSOV_A' and then intuitively, it may be seen that in order to model the response of valve to the controller input, the valve and its associated instances must be extracted from SD. Similar to SUT consistency evaluation in previous example, in this case the presence of valve in SD is checked and if it exists all the equipment which are needed for this valve to operate are extracted. From exploring the design space i.e. SD, it can be seen that the equivalent valve 'SDD_PRSOV_A' needs other equipment, solenoids in this case denoted by 'SDD_S2' and 'SDD_S3'. In turn the valve may have different modes of operation and each of the mode are associated to the solenoid mode (energies or de-energized) based on the operating mode definition which is explained further in the next section. The queries are given in section 2.2 of annex and only the output is shown in figure 1.8.

c_req	c_avb	c_avb_1	m_avb_1
LTR_PRSOV_A	SDD_PRSOV_A	SDD_S2	SDD_S2_energised
LTR_PRSOV_A	SDD_PRSOV_A	SDD_S2	SDD_S2_deenergised
LTR_PRSOV_A	SDD_PRSOV_A	SDD_S3	SDD_S3_deenergised
LTR_PRSOV_A	SDD_PRSOV_A	SDD_S3	SDD_S3_energised

Figure 1.8: NAIS Instances Design Space Exploration

In addition, there are other queries written to extract for example, systems and functions needed to simulate, OM to simulate the NAIS behaviour to controller input under normal and failure conditions etc. An application of the OM concept to the failure mode simulation of NAIS valve is presented in the following section similar to example in section 4.1.2 of chapter III. It may be noted that the other parts of this study including different query mechanisms used in building the MR has not been discussed in detail for the sake of brevity but presented in annex.

1.2.3 MR for NAIS Failure Scenario

Let us consider a test scenario where TR requires the simulation of valve V_2 failure at closed mode. The test request typically says at which conditions the failure is triggered, where and what are the expected outcomes. On the other hand, SD of NAIS describes all the possible behaviour of system, in this case, dependency of valve V_2 modes with the solenoid $S_{2,3}$ modes (e.g.: valve is open when solenoid is energized & closed when solenoid de-energized). It then becomes imperative to identify the components and its associated modes causally affected by this failure condition. Inferring the instantiated OM concepts and querying over this knowledge, desired information such as dependent component or the components that can be abstracted can be obtained with ease. It alleviates the burden of the tedious and often error prone task of keeping track of disparately located but hidden information which is related to each other. Following the notation given in section 4.1.2 of chapter III, the SD then becomes [Ponnusamy,2016]

$C^1 = \{V_2\}$ M_1^1 = open M_2^1 = close M_3^1 = regulating

$C^2 = \{S_2\}$ M_1^2 = de-energised, M_2^2 = energised

$C^3 = \{S_3\}$ M_1^3 = de-energised, M_2^3 = energised

The OM is built from the mode data and is illustrated below, for the sake of clarity each OM is shown separately.

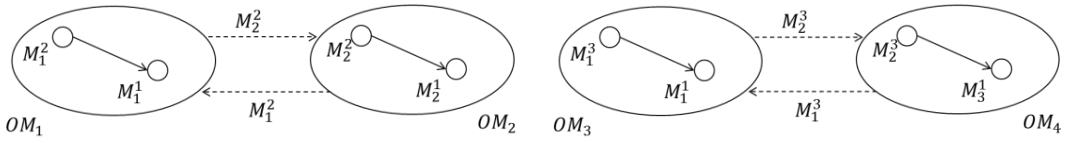


Figure 1.9: Operating Modes of Valve and Solenoid

Consider a test on the controller to validate its failure monitoring and reconfiguration of valves. It can be seen that, in order to simulate the valve failure when closed, it is imperative to simulate the solenoid S_3 in de-energized mode to see it does not have any effect. However, this information is not explicitly given in TR as it describes expectations on system at higher levels of abstraction whereas SD describes all possible behaviours of system. Thus it becomes important to identify only the necessary functions and associated systems to be modeled to avoid over or under detailing of models.

In addition, such an approach will help visualize and identify possible emergent behaviour which may not have been modeled otherwise. For example, from the valve which is failed at the closed position, the regulating mode can be reached in two steps by having S_3 de-energized and S_2 energized. Similar extensions are possible and such information is usually not given explicitly either in SD or in TR, and this formalism helps the model specialist in writing a MR with autonomy. This particular example, though done manually, is found to increase the efficiency during test since provisions for failure triggering is explicitly identified and provided along with necessary functionalities to model the failure propagation.

1.3 DISCUSSION

A classical way to measure the efficiency of this approach is to compare with the manual MR construction. Two key metrics of interest are, how far this method yields (at least) the same results as the manual approach and how quick it does it. For this particular case study, for a subset of MR construction, it is found to be at least twice as efficient as the manual-experience based heuristics approach, which is arduous and error prone at times. In addition, the coverage of results is about 80% against manual approach. However, as noted in chapter III, this approach is only complementary to the classical manual approach whose necessity will always exist in a collaborative system engineering process and a truly automated MR construction is a far cry from the reality. Nevertheless, as seen from this case study, this semi-automated process helps significantly in standardization, extraction, archival and exploitation of the system design and V&V activity knowledge.

2. FORMAL APPROACH CASE STUDIES

In this section, the implementation of quantitative approach discussed for different class of discrete systems is presented. The application case is a buffer system and different abstractions of it are used to demonstrate the approach. In case of untimed interface games, a simple NAIS model as shown in section 5.3 of chapter IV is used as case study.

2.1 AUTOMATA: BUFFER SYSTEM CASE STUDY

The buffer is a simple FIFO which receives jobs from the job generator and sends them to the processor whenever the processor is free. It works as following, whenever a job is received the queue, *q* is incremented and decremented when the job is sent to the processor. The received and sent jobs are denoted by label *e0* and *s0* respectively, processor status by *e1*. Let us imagine the processor to be the SUT and the requirement is to model the buffer with sufficient fidelity such that some scenarios, $\varphi_{i=1}^{N}$ on the processor can be tested. This experimentation is illustrated as an EF in figure 2.1 [Ponnusamy,2016].

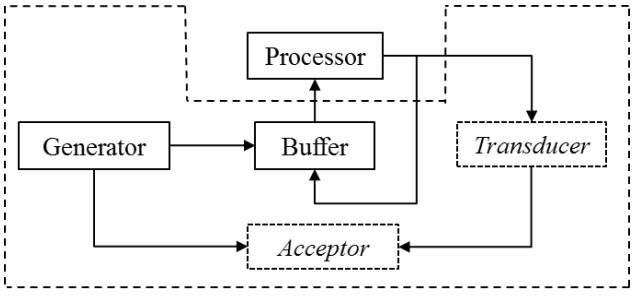


Figure 2.1: Processor Experimental Frame

It can be seen that in addition to the generator and buffer, the experimentation may involve a *Transducer* to interpret the processed and generated jobs and an *Acceptor* which compares the jobs generated vs processed to ascertain the validity of the processor. The system specification of buffer,

M_{sys}, is supplied by the designer and the scenarios by the test team. The model developer who intends to build an abstraction i.e. a model of this buffer, M_{sim}, needs to quantify his model with respect to the system both globally and with respect to $\varphi_{i=1}^{N}$. The games are discussed for timed and untimed model in the following sections.

2.2 UNTIMED BUFFER MODEL

Consider an un-timed automaton modeling buffer behaviour. The guards and actions are denoted by {} and [] respectively. The system model, M_{sys}, and four candidate simulation models, $M_{sim}^{1..4}$, are shown in figure 2.2 and 2.3 respectively [Ponnusamy,2016],

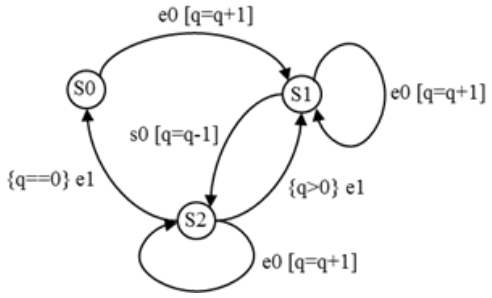
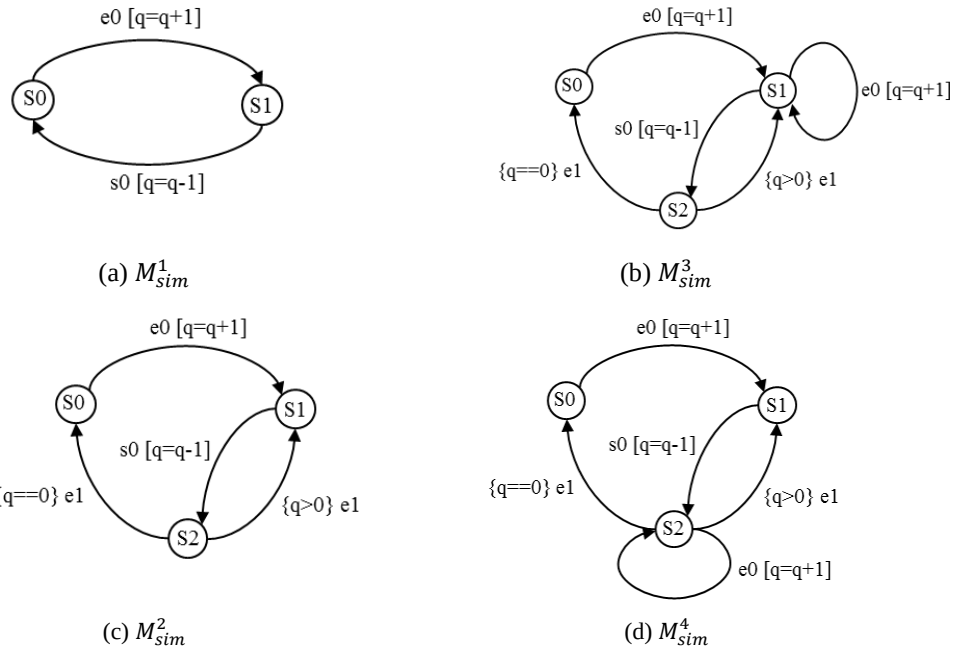


Figure 2.2: Buffer System Model, M_{sys}



(a) M_{sim}^{1} (b) M_{sim}^{3}

(c) M_{sim}^{2} (d) M_{sim}^{4}

Figure 2.3: Buffer Simulation Models

For example, the game between the two models, system model and simulation model 1 (c) is informally described as follows,

$n_p = 1$: From initial state, S_0, player 1 chooses transition e0 and moves to S_1. Player 2 does the same.
$n_p = 2$: From S_1 the player 1 chooses e0. The player 2 cannot match and thus cheats with s0.

This continues forever and one can see the weighted error is simply $(n_p - 1)/n_p$. Similar games can be played between other models. Figure 2.4 illustrates the distribution of trajectories, also called traces, based on the (absolute) weighted fidelity i.e. $(1 - \mathcal{E})$ which is usually shown in percentage. It can be seen that higher the number of trajectories close to 100% or required fidelity, the higher the simulation model fidelity. For the sake of illustration only trajectories up to the third play from a total of 10^3 plays are shown.

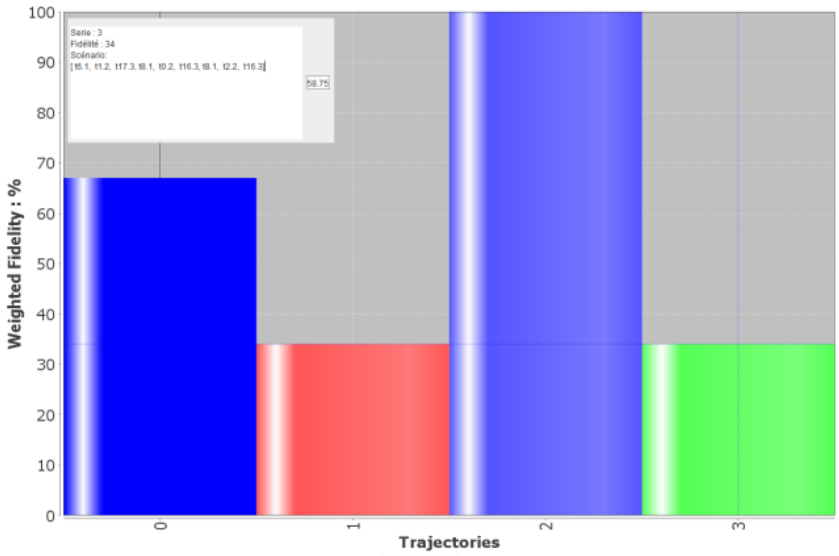


Figure 2.4: Trajectories Fidelity Distribution

From this graph, one can see that out of four trajectories generated at the end of the third play by the system, only one is matched by the simulation model exactly and the second trajectory (in rose and green) cheats twice out of three transition i.e. 33% representative, whereas the fidelity of the other (in blue) is 67%. A particular trajectory can be picked up for visualization by clicking on the graph as shown in the upper left box inside the figure 2.4 and in addition, the mean fidelity, in this case ~67% is also shown. The reachability tree is presented in figure 2.5 for the sake of illustration. This reachability can also be analyzed as a measure of total number of cheats per turn with respect to the total number of trajectories at that turn. For example, in the above example out of four trajectories two are cheating at the third play and in general, lower this ratio, the worse will be the fidelity. This is illustrated for all the four models in Figure 2.6. In addition, the number of trajectories cheated (in black) and the total number of trajectories (in red) at each play is also given.

In the case of relative cheating, instead of absolute weighting in error calculation, relative weighting is employed. Let us consider a scenario, φ_1 stating that the processor must process all the jobs generated, or in other words, no job is lost by the buffer. For this particular scenario, weighting is more on *e0* and less on other labels similar to example in section 5.1. Similar such weightings can be done for other scenarios and analysis is done as in figures 2.5, 2.6 [Ponnusamy,2016]. In addition, sensitivity of weights to the error for a given scenario can be

studied as well to ascertain a viable trade-off between model abstraction i.e. complexity and fidelity.

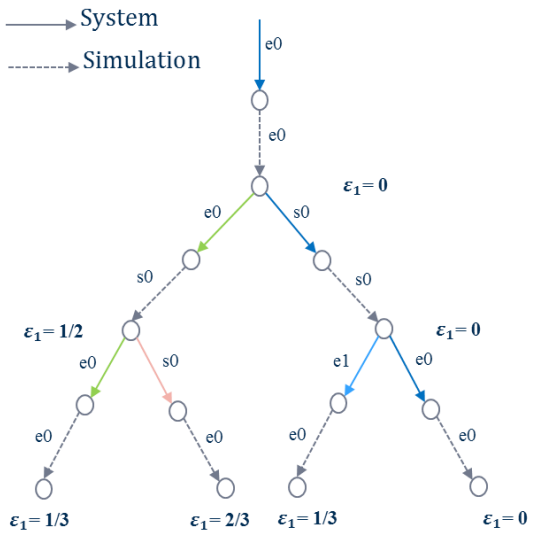
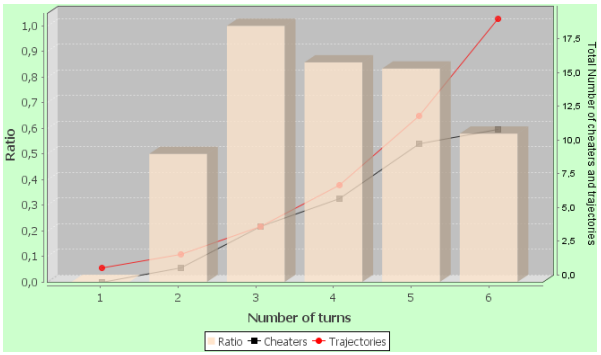
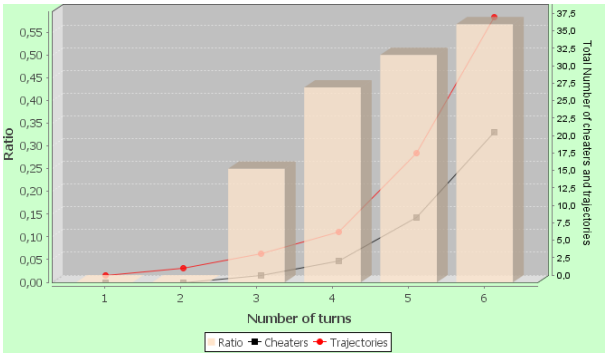


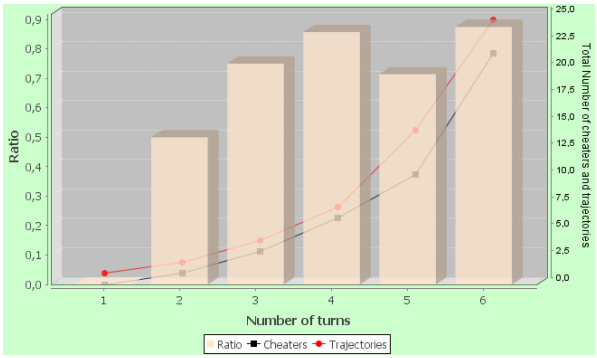
Figure 2.5: Quantitative Reachability



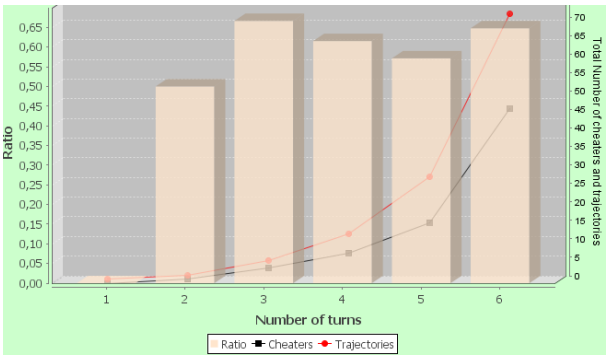
(a) M_{sys} vs M^1_{sim}



(c) M_{sys} vs M^3_{sim}



(b) M_{sys} vs M^2_{sim}



(d) M_{sys} vs M^4_{sim}

Figure 2.6: Trajectories Distribution

2.3 TIMED BUFFER MODEL

In this section, timing aspects are taken through timed games for the given buffer system specification, M_{sys} and two simulation models of the system, $M_{sim}^{1,2}$ as shown in figure 2.7 and 2.8 respectively [Ponnusamy,2016]. The transition labels are typically given in the form of tuple <{}, σ, t, []> where {},[] refers to guards and actions respectively. In this case, the guards and actions are on the queue variable, q.

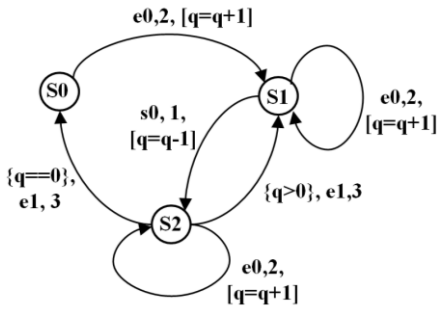
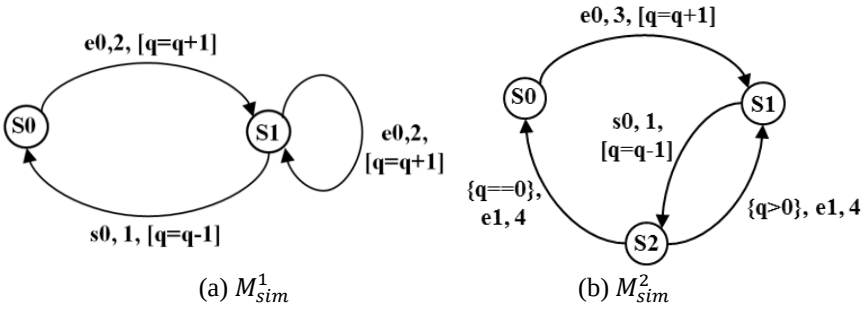


Figure 2.7: Buffer Timed System Model, M_{sys}



(a) M_{sim}^{1} (b) M_{sim}^{2}

Figure 2.8: Buffer Timed Simulation Model

This game can be either played state bounded or equivalently play bounded. In the former, the maximum number of state classes generated during reachability construction is fixed whereas in the latter the play is terminated only if all the winning trajectories (if it exists) where $\sigma_i^1 = \sigma_i^2$ of player 2 are played. In addition, a play can be terminated prematurely if the number of lost trajectories exceeds a certain user defined bound. Different such techniques could be employed to manage the game and interpret the results to determine the fidelity according to the user requirement. In the following section some fidelity metrics are discussed for the buffer model.

2.3.1 Analysis Results

The timed fidelity game is played between M_{sys} and M_{sim}^2 and a quantitative reachability graph is generated for a maximum 10^3 state classes. Since the size of the resulting $\mathcal{R}_\varepsilon$ is limited, the first question is how many traces are generated and how long they are i.e. length. In total 4661 traces were generated with 3640 traces has maximum trace length of 26 transitions. It may be reminded that in this case, the system model makes infinite number of turns regardless

of the simulation model and incompleteness of each trace is predominantly due to the truncation of reachability states generated. The distribution of all such transitions can be visualised in the following figure 2.9 [Ponnusamy,2016]. It can be seen that most traces have one or two transitions empty due to reachability graph truncation and this information can be used to limit or extend the limit of exploration.

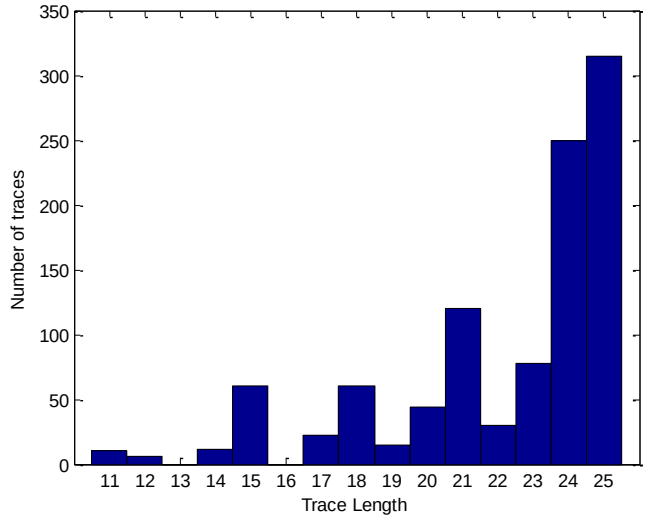


Figure 2.9: Trace length vs Number of traces

For each trace, the number of plays may be different i.e. a play might be lost but still the trace contains only player 1's transitions. It can be seen from figure 2.10 that simulation model can match the transition labels for a maximum of 5 plays for 45 traces. For each of these traces, associated timing error can be extracted similar to figure 2.5 [Ponnusamy,2016].

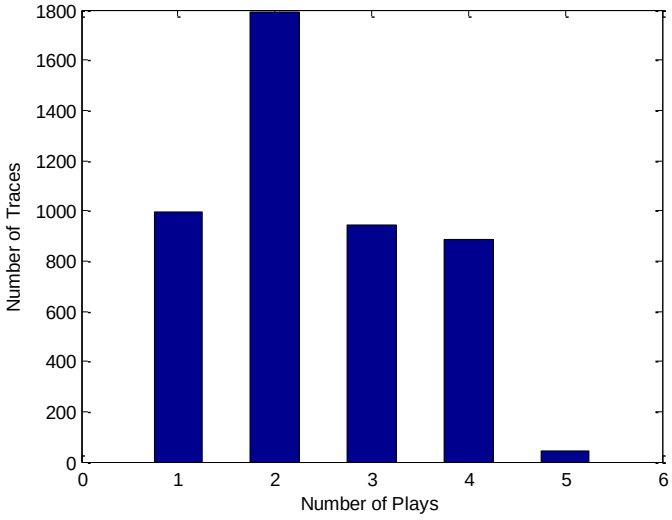


Figure 2.10: Total number of plays distribution

For example, the trace with transition sequence $e0 \xrightarrow{2} s0 \xrightarrow{1} e1 \xrightarrow{3} e0 \xrightarrow{2} s0 \xrightarrow{1} e1$ of system model can be matched by the corresponding sequence $e0 \xrightarrow{3} s0 \xrightarrow{1} e1 \xrightarrow{4} e0 \xrightarrow{3} s0 \xrightarrow{1} e1$ of the simulation model and the net timing error is 3 time units at the end of fifth play. However, for some other traces it can match only partially, for example one can intuitively see that a job can arrive at any state for the system model whereas the simulation model can take job only at state S0. In such traces, (e.g. $e0 \xrightarrow{2} e0 \xrightarrow{2} s0 \xrightarrow{1} ...$ by the system) the game is partially lost and such information too can be obtained.

Another key information of interest is the lead information i.e. how far the system is in advance before the simulation model and this represents the overall lag of the simulation model with respect to system. A near perfect simulation model has less lag and increase in lag is either due to the play being lost in that trace especially for systems with loops such as buffer or simulation model timings are higher. The following figure shows this difference and it can be seen that almost all lag is due to the play being lost in corresponding traces. At maximum only one transition i.e. e0 is matched for 995 traces.

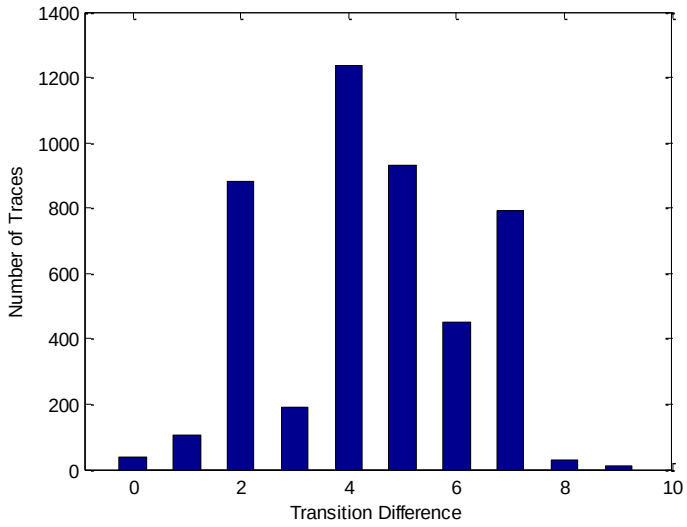


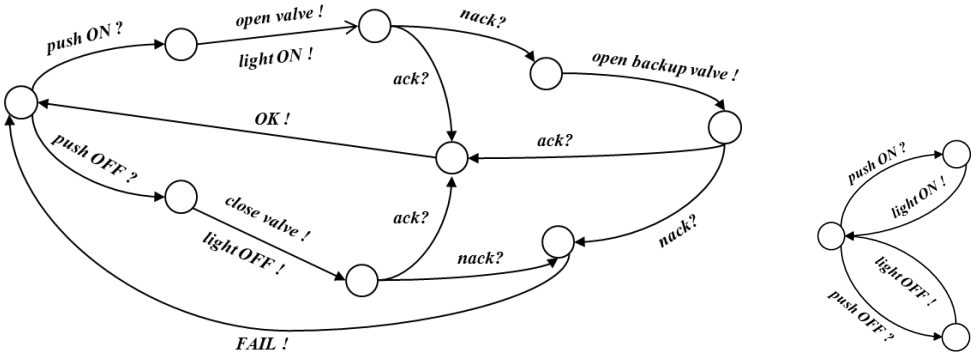
Figure 2.11: Transition difference distribution

A key aspect which is not discussed is the role of V&V objectives in this fidelity quantification. In the current study, all the differences in transition timings are equally weighed. However, in reality a model is developed with some V&V objectives behind and in such cases some transitions are of more interest than the others. Let us consider a requirement: φ_1 on SUT stating that all the sent jobs must be processed by the processor i.e. no job is lost. In other words, an ideal buffer must store and send the jobs to processor as a function of processor status. In case of first simulation model this is not true as the processor status is not modeled. This is characterised by the losing game in the third play of the game whenever the system makes a move with *e1* label. However, in case of second simulation model the game is not lost but the event timings are different. On the other hand, consider requirement, φ_2 on SUT stating processor expects at least one job at delivered by the buffer at 3s and in this case first simulation model matches exactly the transition timings $e0 \xrightarrow{2} s0 \xrightarrow{1} e0$ compared to the M_{sim}^1. Thus, depending on the requirement, some transition timings are weighed more with weighting w_1, than the others with weighting w_2 as shown in sections 5.1

of chapter IV. Such a relative weighting approach though not shown here is illustrated in the annex with a simple example.

2.4 INTERFACE AUTOMATA

In this section, an application case to demonstrate fidelity quantification for untimed interface models is discussed. Let us recall figures 5.7 and 5.8 of chapter IV of the controller system and the first simulation model.



(a) M_{sys} (b) M_{sim}^{1}

Figure 2.12: NAIS Controller Model

The game is played between these two systems based on the theory given in section 5.3.2 and its implementation in 6.2 of chapter IV.

2.4.1 Analysis Results

The game is played such that it generates atleast 10000 state classes though TINA is capable of generating more. The resulting reachability graph is parsed and the fidelity evolution is shown for first seventeen plays. In other words, the depth of reachability graph is 52 as a play has three transitions. In this seventeen plays, a total of 14048 trajectories were generated whose fidelity distribution of is illustrated in figure 2.13. The maximum fidelity acheivable is 53% for 2592 trajectories and minimum 36% for 512 trajectories. It can be seen that in most of the cases, this simulation model can produce atleast one out of two transitions of the system model i.e. 50% fidelity. It may be noted that this error might be different depending on where the play is terminated and it may be chosen according to the SOU.

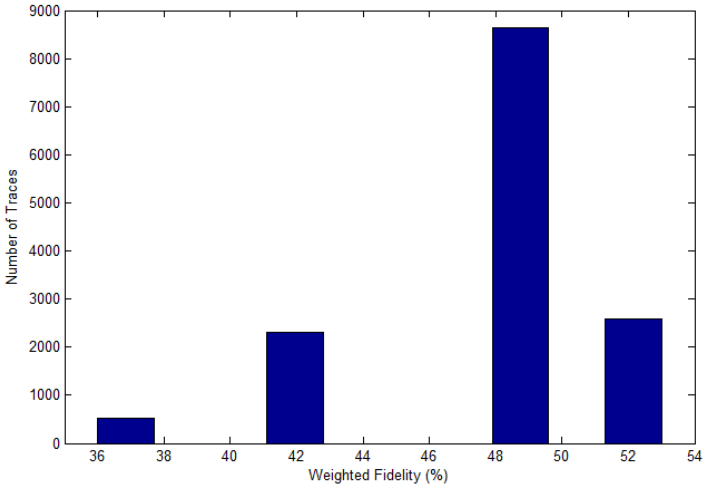


Figure 2.13: Fidelity distribution

In the next figure, the fidelity evolution similar to figure 2.6 in section 2.2 for untimed interface automata games is illustrated. The upper bounds and lower bounds of achievable fidelity is marked by red and green curves at each play.

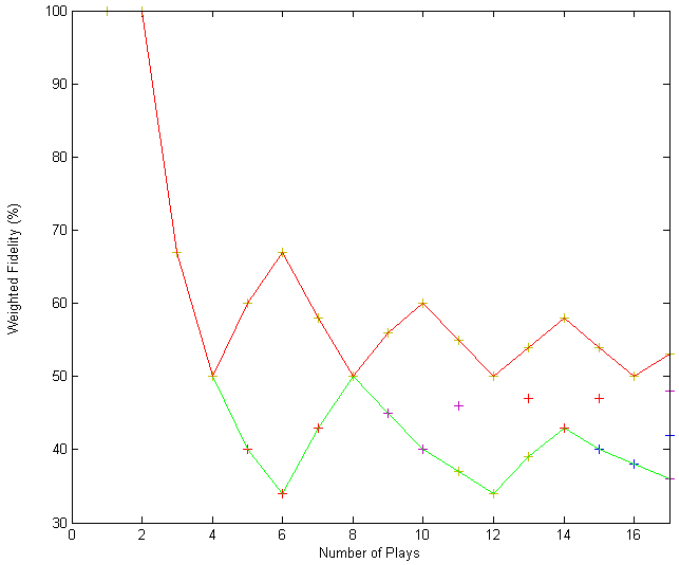


Figure 2.14: Fidelity evolution

It can be seen that initially the fidelity is 100% since for the first play the player 2 can match *pushON?* or *pushOFF?* input move of the player 1 on sytsem move by the same move on simuation model. Similarly for the second play *lightON!* or *lightOFF!* output move of the player 1 on simulation model can be matched exactly by player 2 on system model. However, if player 1 chooses to play for example *ackON?* on system model then the player 2 has to cheat with its input action and thus the fidelity becomes 2/3 i.e. 66%. Thus the fidelity starts to degrades progressively though it rises and falls intermittently since the system model and simulation model comes back to inital state after some transitions.

Similar such studies were done for other models shown in figure 5.8 of section 7 in chapter IV to select the best model both globally and locally i.e. with respect to SOU. The relative weighting approach, discussed in section 5.3.2.5 of chapter IV, is not shown here as in principal the analytics is same such as the one shown here with the only difference being the effect of weighting can be varied and analysed with respect to the fidelity distance calculation.

2.5 DISCUSSION

It may be noted that despite the novelty of the approach in quantifying fidelity explicitly and formally, as noted in chapter IV, it is not scalable for higher dimensional systems or higher number of plays. Despite the fact that reachability generation is fairly easier with TINA which uses state of art techniques to generate the graph, exploiting it is difficult. The exploitation requires an explicit reconstruction of this reachability graph with associated error information at the end of every play. The current method involves classical reconstruction based on parsing the output file which essentially starts at the end of the tree and progressively builds the tree backwards until the inital state is reached. The algorithm, given in annex, though is efficient since it avoids redundant exploration whenever a branch is encountered, it still needs to be improved using efficient data structures and other programming techniques. In addition, the current analytics involve only information such as evolution of fidelity distance, distribution of associated trajectories etc. This could be augumented with additional information such as how are those trajectories related and which player 1's move causes higher or lower error and vice versa for player 2 etc. Such information could be then used to refine the simulation or system design further. Nevertheless, this approach once implemented, even for a limited number of plays gives a formal distribution of fidelity withrespect to or independent of scenarios which could be reused for similar or derivative systems i.e. systems which are a variant of the original systems validations.

3. CONCLUSION

The application of ontology based semi-formal approach and quantitative reachability based formal approach has been presented. The semi-formal approach is applied on a real industrial scale problem whereas the formal approach is applied on various classes of illustrative academic and simplified industrial examples. The formal approach needs to be extended to higher dimensional industrial models which are currently being carried out. The semi-formal approach, with its encouraging preliminary results, is currently being compared against classical system engineering approaches [Roques,2016], [SysML,2006] for different case studies along with improvements being done on methodology refinement and tool development for the stakeholders. The future work and overall conclusion is discussed in the chapter VI.

CHAPTER VI

OUTLOOK & CONCLUSION

In this chapter, current and future work for two approaches discussed in chapters III and IV are presented.

1. OUTLOOK ON SEMI-FORMAL APPROACH

The domain model approach presented in chapter III needs to be further developed, automated and integrated with the engineering process with user friendly interfaces. The SBFIO ontology used to define this domain model could be improved with additional concepts based on naïve and basic physics and other domain specific concepts. This is particularly useful to document hypothesis behind specifying a system by system designers which will be later used by the model developers. The preliminary results of the application case described in chapter V demonstrate the flexibility of this approach in archival and exploitation of domain knowledge. The future work also includes development of a user friendly graphical interface for domain model instantiation, queries and formalization of a centralized ontology management process which are imperative for its utilization across the enterprise.

One of the fundamental challenges in this domain model approach is its validity. Though the very purpose of ontology is to establish a set of vocabulary which are validated, in our approach a mix of established as well as new vocabulary were introduced since our primary concern is to capture and exploit the system design, testing and modeling knowledge. This will naturally be followed by the validation with concerned stakeholders and ontology refinement. In addition, this being a flexible approach, extending it to many other domains, the complexity of such ontology might become prohibitive and render the approach difficult to use in practice. Hence, our approach concerns only the M&S domain with a specific focus on MR extraction from the SD and TR knowledge. Even in such a focused application specific domain model approach, not all the concepts of TR and SD could be formalized and in order to have a reasonably adequate model with tractable complexity, only some important and frequently used concepts and relationships were implemented to illustrate the flexibility and adequacy of the approach for our problem.

It may be noted that the queries used to extract artifacts from TR and SD body of knowledge is based on specialized language such as RDFS [Sintek,2002] which may not be amenable for the end user. The queries need to be customized often according to the model specialist need and this may be a challenging task especially in an industrial setting where the domain model in all probability will be managed by a separate entity. Most often than not, queries fail to deliver answer due to incomplete or wrong domain model instantiations and the users may not be aware of the origin of this problem. In order to mitigate, query relaxation techniques [Foukou,2016], [Smits,2013] which return associated alternative answers will be valuable to the end user to further explore and identify

inconsistencies, incompleteness which will be further used to refine the MR in relation with SD and TR.

Another key challenge besides the validation of this domain model approach is the complicated task of parsing documents written in natural language into the domain model concepts described in chapter III. This process is currently manual and involves stakeholders to manually instantiate the domain model which is cumbersome and at times error prone though they could be identified to a certain extent [Ponnusamy,2016]. There are some initial studies based on Natural Language Processing (NLP) techniques such as text mining [Ileiva,2005] to automate this process, it is far from being done especially in an industrial context. This problem needs to be studied with cognitive techniques such as data analytics and deep mining based on iterative learning techniques for better usage of this domain model. These cognitive techniques, by extension are also being studied to automatically construct ontologies from existing body of knowledge such as text [Dahab,2008],[GATE]. Such an approach, in addition to being complementary, will also help in validating our approach which was a manual construction based on a study of processes, documents and existing practices.

A possible drawback of our approach is the implementation of class definition in Protégé and model selection implemented in other language for example in SysML. This is done in order to leverage the flexibility, scalability, query and reasoning powers of ontology with the control flow execution, graphical interface capabilities of SysML. However, this approach has limitations in terms of effort and at times redundant since instantiations are done twice, first in ontology tool such as Protégé and second for the model selection in SysML (or any other tool). This necessitates an integration of SysML and OWL as remarked by [Greves,2009] and [Wagner,2012]. Such a mutual transformation between SysML and ontology will help practising engineers to capitalize on their graphical syntax and reasoning capabilities respectively and thereby ensuring seamless design and product V&V activities. For the problem of design fidelity approach, despite its graphical multi-layered approach, practical limitations such as lack of operational capability and activity information in SD or TR, persistence of ambiguity due to natural language, lack of ability to create a class of functions or their reuse etc. needs to be addressed. In this context too, ontology based approach serves complimentary to such classical industry standard graphical system engineering methods. In addition, feasibility studies are being made to assess and provide feedback on using other MBSE approaches such as Cappella tool based on Arcadia framework [Roques,2016] for aircraft system architecture definition and simulation with respect to our approach.

An important area to be addressed in the overall V&V process is the synthesis of requirements. Requirements are usually written in natural language text and unless they are managed by tools such as DOORS [DOORS], it becomes a tedious task to consistently update, trace or modify the requirement database. An active area of research is to move from informal natural language description to a more semi-formal MBSE approach and in some cases formal description in some temporal logic such as Linear Temporal Logic (LTL) [Pnueli,1977] or Signal temporal Logic (STL) [Donze,2014] etc. especially for formal approach. A more formal definition of requirements would enable better rapid prototyping of systems. Such a method will help in coherent model development and deployment from top level requirements capture to low level behavioural modeling by mapping the related concepts at each intermediary step. In addition, studies are being carried out to extend this approach for other perspectives of experimental frames discussed in chapter II.

1.1 MAPPING TO BEHAVIOURAL FRAMEWORK

There exists a gap between the rigorous behavioural abstraction frameworks based on simulation relations presented in chapter IV and less formal system engineering approaches such as domain model approach presented in chapter III [Retho,2013]. It has been widely emphasized that the complexity of current engineering systems requires integration of different layers of abstraction and consistency between them. In classical system engineering tools such as SysML, there exists mechanism such as Syndia [Syndia] allowing interconnection between behavioural simulation tools in addition to CAD tools etc. to SysML. However, our inclusion problem necessitates formalization and exploitation of knowledge to write MR such that these capabilities are useful in specifying a model, MS consistent with MR i.e. a verification problem. In our study, the formal approach assumes all the model's behavioural tolerances are explicitly specified at least at compositional level i.e. TR is complete. However, this is usually not the case as the MR construction in terms of behaviour for simulation product components not specified explicitly in TR is still an open problem. For example, questions such as, what are the allowable tolerances on behaviour of a model or what are allowable abstractions to model rise time of a model output and what are its impacts on fidelity etc. are questions traditionally done based on expertise and measured fidelity approach. In addition, questions on how are these behavioural requirements impact other perspectives such as system, function etc. and vice versa.

One of the main reasons is the absence of a connection between these two levels of abstraction especially in the inclusion context. The concept of Operating Modes (OM) could serve as a connection between high level functional description through the domain model approach and low level behavioural description through quantitative transition system. Since OM are high level behavioural description, this would lead to better identification and modeling of transitions to capture the low level behaviour, especially during incremental model synthesis. These operating modes can be mapped to automata which model the system behaviour and this may then be applicable to hybrid automata defined by invariants, guards and resets as well [Tomlin,2003]. Such behaviour can be formally verified by reachability analysis and significant progress has been made in the control community in developing various geometric abstractions such as zonotopes [Girard,2007], polyhedrons etc. to perform reachability analysis over dynamic systems [Stursberg,2003]. In addition to verification, syntheses of abstractions are also studied with the help of approximate bisimulation techniques in [Girard,2007] & [Pappas,2003]. Alternatively, such a model can be executed using a discrete event system (DEVS) simulator such as ProDEVS, or state of art DEVS simulators such as CD++ [Wainer,2002] or classical simulators such as like SIMULINK or Modelica [MODELICA]. This domain model for model execution complements the domain model for model building. Such an integrated domain model approach helps in standardizing M&S activities and thereby improves the overall fidelity.

In the semi-formal approach, fidelity is qualitatively determined and as discussed in section 3 of chapter III, hierarchical ordering of abstractions and the notion of lattice distance will lead naturally to quantification. Similarly, the semi-formal requirements need to be mapped to formal system requirements as discussed in chapter IV in order to have a unified approach to the problem of fidelity. However, these studies are still in its infancy and studies need to done to bridge this behavioural approach with the semi-formal domain model approach to build a unified framework addressing the simulation needs capture at from high level to low level model behavioural requirements definition.

2. OUTLOOK ON FORMAL APPROACH

The formal fidelity quantification approach discussed in chapter IV is limited in its scope of class of systems, abstraction mechanisms. The current abstraction mechanism is only of omission (states and/or transitions) for discrete systems and future works include the extension of this approach to other widely used abstractions such as state aggregation. The choice of error model (discounted average, mean average etc.) in our approach and its associated weighting function, though beneficial in terms of its flexibility to represent fidelity distance in different perspectives, is still a challenging task to identify, converge and implement. The current heuristics based method needs to be improved and in this regard, an ontology of different error models need to be developed to standardize such implementation especially in the context of class of fidelity requirements such as performance, robustness etc.

The open vs closed systems fidelity perspective in terms of classical and interface automata needs to be further developed especially from a process perspective. The component perspective of what does it do i.e. automata vs interface perspective of how can it be used i.e. interface automata, though complimentary to each other, need to be integrated in unified process where the model developer chooses the formalism according to the need. For example, in cases of input universal i.e. closed assumptions automata games could be used for fidelity quantification and in cases of input existential i.e. open assumptions interface games could be used. In both such approaches, our formal quantification method assumes equivalence on actions i.e. labels as described in section 5 of chapter IV. In reality, this is usually not the case, however, an ontology will help in standardizing the convention for using labels which are common across different model specifications.

The interface distance notion of experimental frame needs to be augmented with internal actions and such an approach will help in modeling the internal behaviours of components and this will help further in unifying the component and interface perspective especially in the fidelity context. In the current behavioural approach presented, in addition to the problems being studied, it must be noted that the quantitative notion needs to be related in the EF context of transducer and acceptor. Since transducer and acceptor depend on observations of abstracted model, the approximate language equivalence i.e. fidelity distance must be compatible such as EF cannot expect incompatible precision on outputs of an abstracted model. Also, the abstracted system can have different control input than original system and this need to be studied through the EF component abstraction such as abstractions of generator. The controllability and observability analogy of dynamic systems also have to explored in the context of experimental frame to extend the approach to higher dimensional systems such as section 6 of the annex.

As discussed in section 7 of chapter V, a possible axis of future work is to extend the formal approach to DEVS V&V [Labiche,2005], [Wainer,2009], [Saadawi,2009]. In [Albert,2016], a Petrinet implementation of the parallel and classical DEVS has been discussed and this is currently being studied in the context of game theoretic quantitative formal verification for DEVS models. In addition, Giambasi et al [Giambasi,2003], discuss the mapping between DEVS which is deterministic in nature and timed automata which allows for non-determinism. This mapping based on simulation relations has been used for the formal verification of DEVS in [Giambasi,2003]. However, a mapping between DEVS and timed interface automata which explicitly takes into account the timing as well as input/output events is far from being complete. This is also due to relative infancy of timed interface formalism and its formal verification aspects. It is worth

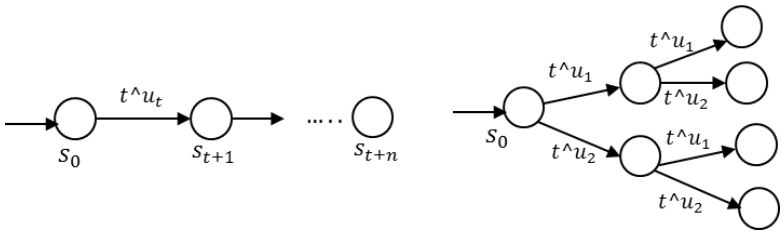
exploring such an approach since the formal verification of DEVS model allows, for example, to prove that the implementation of high level specification modeled in timed (interface) automata as a DEVS model is correct or incorrect.

The fidelity quantification approach concerns predominantly discrete state and/or time systems and to a limited extent on continuous systems (section 6 of annex). The discrete dynamics could be augmented with stochastics into probabilistic automata and our distance based approach could be complimentary to existing probabilistic model checkers such as PRISM [Kwiatkowska,2001]. Despite the focus on discrete systems, in reality, the models of many domains especially control are usually continuous whereas informatics domain is discrete with hybrid models having discrete with embedded continuous dynamics. In the formal verification for continuous models, state enumeration is done symbolically by geometric over-approximation using zonotopes etc. in quantitative verification of properties such as safety which is usually independent of SOU. On the other hand, model checking with its explicit enumeration suffers from state-space explosion and not useful for continuous systems. A solution is to convert continuous systems into timed automata formalism and do model checking [Maler,2008]. This is useful to find a degree of similarity between two systems through a timed-game theoretic approach. However, this is challenging since it involves state space partition which must be done right to avoid any spurious behaviours as remarked by the authors. By contrast, this degree of similarity for continuous systems given by approximate bisimulation is a global measure i.e. all trajectories of system are bounded by this measure with respect to other system and this can be demonstrated via symbolic reachability post priori i.e. assuming same timing of events. In the field of simulation, finding bounds on each trajectory i.e. a quantitative reachability suffers from the curse of dimensionality as it is explicit enumeration of states. Thus a compromise is to have a rough timed automaton of a continuous system and then play the game with its counterpart to determine this error. In this case, the global fidelity is introduced by first approximation of changing formalism and then the usual fit of similarity between the systems. But this may not change our approach since the fidelity measure is relative i.e. independent of this approximation by timed automata. Let us consider a trivial example by converting a discrete timed system into timed automata. Let us take a state space system, state y is input driven while state x is not, then the continuous and discrete versions are given by

$$\dot{y} = 2x + y + 6u \qquad \dot{x} = 5x \tag{1}$$

$$y_{t+1} = 2x_t + y_t + 6u_t \qquad x_{t+1} = 5x_t \tag{2}$$

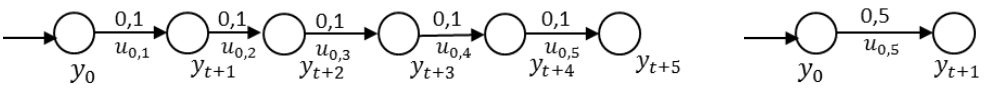
Let us imagine two sampling time, $\Delta t_1 = 0.5$, $\Delta t_2 = 0.1$ time units. In terms of timed automata i.e. a Petrinet with no partial states, a discrete system of form $\dot{s} = As + Bu$ where A, B are fixed matrices, s is state and u is input will take the form as follows,



a. Timed automata b. Reachability

Figure 2.1: Timed Automata & Reachability

The guards and actions perform the new state computation. Then the example in Eq.(2) becomes,



a. System Model b. Simulation Model

Figure 2.2: Effect of Sampling

Let us consider the system and simulation model, intuitively, one can see that for every move of system at 0.1 s the simulation model fails to match and the time difference is 0.4 and system is ahead by 4 transitions. However, this is counterintuitive if we view them at t=0.5 where the outputs will not be different (if gain is static). Thus the difference is essentially time view or the label view and our game is timed view i.e. to quantify timing difference between transitions of two systems. Such approaches need to be further studied especially in the context of fidelity quantification.

In this context, using fickle transitions in timed Petrinet [Berthomieu,2014] potentially opens new ways of formal verification of continuous systems through Quantised State Simulation (QSS) [Cellier,2008] approach which then can be extended for hybrid systems verification. We are currently studying quantitative fidelity approach between two continuous systems which essentially uses a QSS1 [Cellier,2008] approach and timed game semantics similar to section 5.2 of chapter IV to quantify the similarity between these two systems. This approach includes timing information of continuous systems and is expected to be complementary to quantitative approaches such as [Girard,2007] which uses symbolic reachability without timing information. However, such explicit enumeration inevitably will lead to state space explosion and efficient symbolic methods in such quantification need to be studied in future.

2.1 FIDELITY SPECIFICATION

A key practical challenge is the availability of the system specification, especially in formal language such as timed automata. Even in case of such availability, there could be interoperability issues between the modeling formalisms used by the model developer and the system designer. In addition to this challenge of availability of a formal model, an important but often ignored aspect of simulation is its fidelity distance specification i.e. maximum permissible distance between the system and simulation behaviour, ε_{φ}. The classical bottom up approach usually defines this requirement in an informal and vague manner such as ‘as representative as possible’ or ‘closer to real system’ without actually specifying the real need in quantitative terms. Such qualitative requirement for models often leads to inconsistency in system validation. On the other hand, this problem stems from the gap in knowledge between the stakeholders. The test team typically knows the scenarios to test systems but not the models whereas, the design team knows about the system but not the scenarios. Thus it is imperative for the model specialist to elicit this disparately located but often related knowledge to frame the model fidelity requirements. In this context, the usage of ontologies would help to standardize and exploit this knowledge to write the MR. However, this approach is useful only at high level to define requirements on functions, modes, structural composition etc. and less relevant to specify behaviour especially for components not explicitly given in TR.

In order to formally specify model behaviour, it is important to use temporal logic [Pnueli,1977] formalisms such as Linear Temporal Logic (LTL) or Computation Tree Logic (CTL). Recalling definitions in sections 4.1 of chapter IV, the model and its specification are in essence a description of temporal evolution through computation and these computations can be interpreted as words over some alphabet. Thus the model and its specification relationship is essentially a relation between the languages in an automata perspective. The fidelity requirements then could be expressed on this automata formalism as some tolerance over the language of the automata defined for the system being simulated. These requirements are compared with the implemented tolerance which is measured as value of two player game as shown in our formal approach for different class of discrete systems.

A temporal logic formula is built up from finite set of atomic propositions with logic and temporal operators. The tolerance model is given over these basic formulas and the tolerance can either be on label or on time or both. The tolerance is defined over the language $\mathcal{L}$ in terms of alphabets, its limit or both. Recalling the automaton definition, $T^{\varepsilon} = < \Sigma, X^{\varepsilon}, X_0, \delta, R^{\varepsilon} >$ where the language $\mathcal{L}(T^{\varepsilon})$ corresponds to the set of words accepted by T^{ε}. It may be noted that the alphabet remains same whereas the state set and accepting state is different. For example, if the requirement is ◊(2b?⟹c!), which informally means an input of two successive b's eventually (denoted by ◊ operator) results in an output of c. The system shown in left side of the figure below satisfies the property however when this requirement is relaxed with a tolerance on the alphabet as ◊(b?⟹c!), this is satisfied by both the model shown in the right and system.



Figure 2.3: Fidelity Tolerance Example

In addition, the fidelity requirements are usually expressed with some constraint set such as *atleast, atmost* or *exactly*. For example, consider *atleast* requirement over some label and assume a system with output denoted by b! and the TR states atleast three b's are observed every five transitions. When replacing this system by a model, the fidelity requirement on this model can be defined by this tolerance model. The three broad combinations possible as follows: atleast two b's observed every five transitions, atleast three b's observed every six transition and atleast two b's observed every six transition.

The tolerance model can equally be defined for Signal Temporal Logic (STL) which is an extension of LTL with real time and real valued constraints [Donze,2014]. This could be useful in requirement synthesis studies such as [Donze,2013] where a counter-example guided inductive synthesis approach to generate requirements in a specification language based on STL is discussed.

Such a model definition can be integrated as a library with standard requirement management tools such as DOORS [DOORS,2014], REQTIFY [REQTIFY,2015] for defining model requirements for model developers. Such a method allows flexible and standardized definition of tolerance requirements of simulation. However, such a definition could be a cumbersome process since the test team almost always requires models with near zero tolerance whereas, the model developers are unable to develop models satisfying such requirements owing to inadequacies and immaturity in upstream knowledge about systems being developed by system designers. This

definition of tolerances is thus an iterative process and using MBSE approaches will help in keeping track of this evolution and document it for a standardized knowledge exchange between the stakeholders.

2.2 SIMULATION MODEL SYNTHESIS

The game theoretic notion of EF could be well extended from the verification perspective to the synthesis perspective i.e. synthesizing a model given a specification. This synthesis could either be component synthesis modeled as an automata or an interface synthesis modeled as interface automata and in this section only interface synthesis under study is discussed. This correct by construction approach has long been attention of research especially in the computer science [Alur,2015] and control domains [Baleani,2005],[Mazo,2010]. However, a simulation model synthesis for system V&V has never been explored adequately to the best of our knowledge. Simulation model synthesis is similar to classical open systems synthesis problem which is essentially solving a satisfiability problem for every possible input. The only difference is test requirements are given with respect to SUT and the model is synthesized with respect to the system model. In other words, the synthesized simulation model could replace the original system model *iff* it simulates the system model dynamics within the bounds given in the test requirements. It is illustrated as follows,

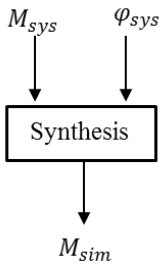


Figure 2.4: Model Synthesis

This section briefly presents a preliminary recursive procedure under study for synthesizing a simulation model through incremental addition of states and checking alternative simulation relation between the system and simulation model. A brief overview is given here to complement the verification perspective discussed in chapter IV. The procedure being studied is discussed only for untimed interface automata which is informally presented as follows,

1. The recursive procedure starts with a simplest partition of system model, M_{sys} taken as simulation model, M_{sim}. This model could either be given as an initial guess or simply the initial state and its outgoing transition pair is taken.
2. Refinement is checked through alternating simulation relations between the system and simulation model. If the refinement error is below the bound the procedure terminates.
3. If the refinement error is above the bound, then the simulation model is improved with subsequent states and transitions of the system model. This is done by comparing the outgoing transition with specification, if the input/output action is present in specification then the transition is conserved else abstracted. If no successor transition is present a dummy transition is added with action and label.
4. Goto Step 2 and repeat the procedure until termination.

It may be noted that the precondition for tolerance evaluation is the reachability objective i.e. only when the simulation model has the corresponding action required in the specification the refinement is checked. The procedure is illustrated with the example in figure 2.5, consider a V&V objective of failure simulation when controller is pushed off. In other words, in case of failure of valves during closing, the controller must output *FAIL!* upon receiving *nack?* input. Intuitively the other section of system model can be abstracted since they do not contribute to the implementation of the specification.

The pseudo algorithm is given below, let the outgoing transitions actions of a state x_i be given by $A^{out}: x_i \rightarrow \sigma_i$. The state at end of every transition is denoted by x_i^N, where *i* refers number of successor states at the end of a transition *n* whose limit is given by N. It can be seen that the successor states can be nondeterministic due to branching and here we assume N to be finite i.e. finite discrete dynamics.

Step 1: At each transition, for the state, x_i^n, get all outgoing transitions σ_i^n of that state

Step 2: Get the next transition state x_i^{n+1}

Step 3: Get all outgoing transitions σ_i^{n+1} of that state

Step 4: Check that $\forall\, \sigma \in \sigma_i^{n+1}, \sigma \in \tau^{\varphi}$, add that successor state to model along with its transition else keep the successor empty

Step 5: Repeat this procedure until all actions of τ^{φ} are included in model or n=N.

Step 6: Measure the interface simulation distance between two models and verify $\mathcal{V}(G_{M_{sys},M_{sim}}) \leq \varepsilon^{\varphi}$, if satisfied, terminate the procedure

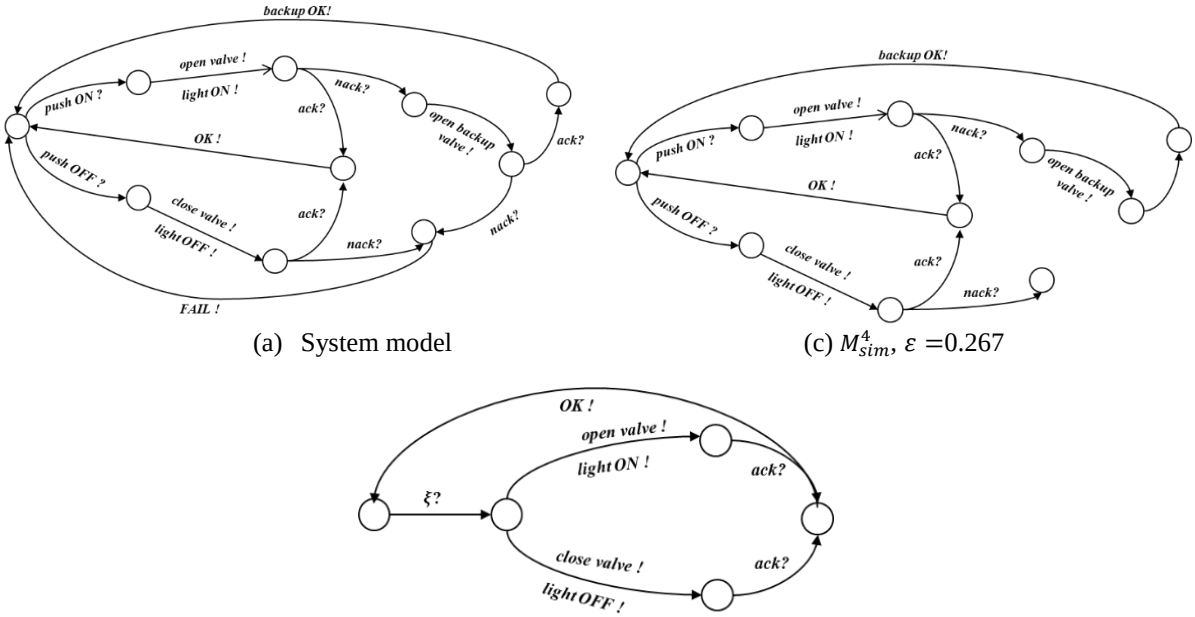
Step 7: If not satisfied, improve x_i^n by adding successor states of system and start procedure from it

Step 8: Goto Step 2

The iteration continues until the model respecting the bound is found and if the procedure does not terminate the error tolerance could be relaxed or system model refined. The termination of this naïve iterative procedure depends on the complexity of the specification i.e. more the actions present in specification then higher is the complexity of the resulting model. It can be seen that this procedure is immediate look ahead i.e. only the immediate successor state's outgoing transitions are visible and abstraction is decided based on that. This refinement can be improved by scaling factor such as look ahead of several transitions but at the price of complexity. A potential pitfall of this procedure is its look ahead nature. Let us slightly modify the example by changing the output action after opening the backup valve to *backup OK!* meaning there is a difference in output of controller in normal and back up mode of operation.

Consider a simple (counterexample) requirement that an *ack?* input is immediately followed by *OK!* output. It may be seen that the system does not satisfy this requirement and an adequately representative model should allow to draw the same conclusion. Intuitively, the ideal model would be the one without *nack?* for closed valve and open backup valve branches but not the open valve branch as it leads to *ack?* of backup valve which gives different output *backup!*. However, a simple simulation model shown in figure 2.5.b with error of 0.405 says *ack?* is always followed with *OK!* which is erroneous. This model is iteratively improved with states and it may be seen that the final model is almost the same as system model with error 0.267 which is an over approximation of the

ideal simulation model. On the other hand, this procedure yields a correct abstraction by taking into account the V&V objectives. The error of 0.267 comes when the attacker selects *nack?* branch of closing valve which the defender cannot match. The attacker's strategy of maximizing error can be seen and the resulting error bound intuitively accounts for every transition of system model. Thus this method is tied inextricably with the reachability objectives or in other words the complexity of the method is proportional to the complexity of the desired reachable states. It is then important to frame adequate but reasonably complex fidelity requirements. For example, a requirement ε^{φ}=0.1 is too stringent whereas ε^{φ}=0.5 is too lenient as it may result in first simulation model being valid.



(a) System model

(c) M^4_{sim}, ε =0.267

(b) M^1_{sim}, ε =0.405

Figure 2.5: Controller Simulation Model Synthesis

In order to better understand the procedure, consider a tree structure of reachability graph. The graph on left refers to system model reachability and the right refers to synthesized model reachability. Let us recall the state are denoted by x_i^n and transition for x_i^n as σ_{im}^n with m being number of outgoing transition. For the sake of simplicity, the input or output actions are not denoted. Let us assume the requirement is to get the output σ_{21}^4 at the end of fourth transition i.e. reach x_2^4 from the initial state denoted with an unfilled circle. The procedure starts with initial state and since the outgoing transition of its successor is not specifically mentioned in requirement the $x_{1,2}^1$ states and the associated transitions are abstracted. However, the outgoing transition of successor of $x_{2,3}^2$ state namely x_2^3 is associated with transition σ_{21}^4 and thus kept as it is. Similarly, at x_2^3, the state x_3^4 is dropped.

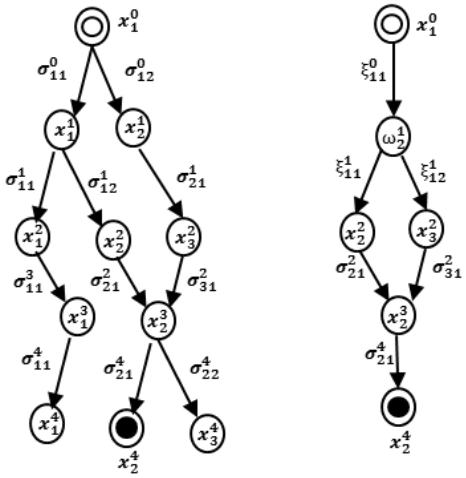


Figure 2.6: Reachability perspective of synthesis procedure

The error of synthesized model is evaluated by approximate alternating simulation relations in section 5.3 of chapter IV. It must be noted however that this approach may not be adequate to guarantee termination for stringent fidelity requirements coupled with very large state space models. Intuitively, mandating a near representative model implies inclusion of all elements of the system and this natural tendency of 'over specification' results in over detailed model with respect to the test scenarios. Thus a reasonable definition of fidelity tolerances is essential to obtain a model with adequate complexity and usually such requirement is given based on capitalization of previous experience and engineering judgment.

3. CONCLUSION

A semi-formal approach based on domain modeling and a formal approach based on quantitative reachability could lead to a unified framework encompassing high level fidelity needs capture to low level implementation. It may be noted that as the simulation product development process progresses, the method and tool used will become more formal and this multi modal i.e. unified approach of using a combination of formal and semiformal techniques will help managing the fidelity of models better. However, application of such formal and semi-formal methods to large scale industrial systems which are typically system of systems with different layers of abstraction will be incremental and the proposition described in this thesis is one such method for simulation model development. In reality, simulation as an enabling method for design and development of systems will not be replaced by formal verification such as reachability, at least not in the near future and not in some specific domains (e.g.: Human Machine Interaction, Failure Detection Isolation etc.). Hence, the onus should be on model development through component based approach where each component models are verified independently with an associated metric before composition and subjected to some use cases by classical simulation or state of the art co-simulation techniques using Functional Mockup Interfaces [FMI,2010] standards etc.

BIBLIOGRAPHY

[Airbus] Airbus, Internal Standard - Develop, Integrate and Validate Shared Simulation Models Methods and Guidelines, 2009.

[Albert,2009] Albert V., 2009. *Simulation validity assessment in the context of embedded system design*, Thesis (Phd). LAAS-CNRS, University of Toulouse, Unpublished.

[Albert,2016] Albert, V., Ponnusamy, S.S., 2016, *Codage de CDEVS et de PDEVS en réseau de Petri Temporisé: Théorie et Application*, Journées DEVS Francophones 2016, N° 15597, 9 pages, France.

[Alfaro,2009] Alfaro, L., Faella, M., Stoelinga, M., 2009, *Linear and Branching System Metrics*. IEEE Trans. Software Eng. 35(2), pages. 258–273, DOI:10.1109/TSE.2008.106.

[Alfaro,2005] Alfaro, L., Henzinger, T., 2005, *Interface-based design.* Engineering theories of software intensive systems, pages. 83–104, DOI:10.1007/1-4020-3532-23.

[Alfaro,2001] Alfaro, L., Henzinger, T.A, 2001, *Interface automata*. In: FSE, pages 109–120, ACM.

[Alfaro,2003] Alfaro, L., 2003, *Game Models for Open Systems, Verification: Theory and Practice*, Lecture Notes in Computer Science, Volume 2772, pages269-289.

[Alur,1998] Alur, R., Henzinger, T., Kupferman, O., and Vardi, M., 1998, *Alternating refinement relations*, CONCUR 98: Concurrency Theory. 9th Int. Conf., volume 1466, Lect. Notes in Comp. Sci., pages 163-178. Springer-Verlag.

[Alur,2015] Alur, R., Henzinger, T., and Vardi, M., 2015, *Theory in practice for system design and verification*, ACM SIGLOG News archive, Volume 2 Issue 1, January 2015, Pages 46-51.

[Alur,1994] Alur, R., Dill, D.L., *A theory of timed automata*, 1994, Theoretical Computer Scence., Volume 126, no. 2, pages 183-235,

[Baader,2005] Baader, F., Horrocks, I., Sattler, U., 2005, *Description Logics as Ontology Languages for the Semantic Web*, Mechanizing Mathematical Reasoning, Volume 2605 of the series Lecture Notes in Computer Science pages 228-248.

[Bair,2013] Bair, L.J., Tolk, A., 2013, *Towards a unified theory of validation*, Proceedings of the 2013 Winter Simulations Conference, Washington, DC, USA.

[Balci,2000] Balci, O., Ormsby, W.F., 2000, *Well-Defined Intended Uses: An Explicit Requirement for Accreditation of Modeling and Simulation Applications*, Proceedings of the 2000 Winter Simulation Conference, Orlando, FL, USA.

[Baleani,2005] Baleani, M., Ferrari, A., Mangeruca, L., Sangiovanni-Vincentelli, A. L., 2005, *Correct-by-construction transformations across design environments for model-based*

embedded software development, Design, Automation and Test in Europe, Vol. 2, pages 1044 - 1049.

[Basile,1992] Basile, G., Marro, G., 1992, *Controlled and Conditioned Invariants in Linear System Theory*, Prentice Hall.

[Bengtsson,1996] Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., Yi, W., 1996, *UPPAAL - a tool suite for automatic verification of real-time systems*, Springer.

[Benjamin,2009] Benjamin P., Akella K., 2009. *Towards ontology-driven interoperability for simulation-based applications.* Proceedings of Winter Simulation Conference, pages 1375-1386, December 13-16, Austin, USA

[Benviste,2012] Benveniste, A., Caillaud, B., Nickovic, D., Passerone, R., Raclet, J.B., 2012, *Contracts for System Design*. Research Report RR-8147, INRIA, pages 65.

[Ben,2003] Ben-David, S., Eisner, C., Geist, D., and Wolfsthal, Y., 2003, Model checking at IBM, Formal Methods in System Design, Volume 22, Issue , pages 101–108.

[Bethomieu,1991] Berthomieu, B., Diaz, M., 1991, *Modeling and verification of time dependent systems using time Petri nets*, IEEE Trans, on Software Engineering 1991, Volume 17-3, pages. 259-273.

[Berthomieu,2004] Berthomieu, B., Ribet, P.O., Vernadat, F., 2004, *The tool TINA–construction of abstract state spaces for Petri nets and time Petri nets*, International Journal of Production Research, vol. 42, no. 14, pages 2741–2756.

[Berthomieu,2014] Berthomieu, B., Zilio, S.D., Fronc, L., Vernadat, F., 2014, *Time petri nets with dynamic firing dates: Semantics and applications*, Formal Modeling and Analysis of Timed Systems, pages 85–99, Springer.

[Berthomieu,2008] Berthomieu, B., Bodeveix, J.P., Farail, P., Filali, M., et al, 2008, *Fiacre: an intermediate language for model verification in the TOPCASED environment*. Embedded Real Time Software and Systems, Toulouse, France.

[Brade,2004] Brade D, *VV&A Global Taxonomy (TAXO)*, 2004, Common Validation, Verification and Accreditation Framework for Simulation, REVVA.

[Brooks,1996] Brooks R.J., Tobias A.M., 1996. *Choosing the best model: Level of detail, complexity, and model performance.* Journal of Mathematical and Computer Modeling, Volume24 Issue 4, Pages 1-14.

[Cellier,1991] Cellier, F.E., 1991, *Continuous System Modelling*, Springer Verlag, New York.

[Cellier,2008] Cellier, F.-E., Kofman, E., Migoni, G., Bortolotto, M., 2008, *Quantized state system simulation*. In: Proc. GCMS 2008, Grand Challenges in Modeling and Simulation, Scotland, pages 504-510.

[Cerny,2014] Cerny, P., Henzinger, T. & Radhakrishna, A., 2014, *Interface simulation distances*. Theoretical Computer Science, Volume 560, pages 348-363.

[Cerny,2010] Cerny, P., Henzinger, T. & Radhakrishna, A., 2010, *Simulation Distances*. Lecture Notes in Computer Science, Volume 6269, pages 253–268.

[Chandrasekaran, 1993] Chandrasekaran, B., Goel, A., & Iwasaki, Y., 1993, Functional Representation as a Basis for Design Rationale, IEEE Computer, Volume 26, Issue 1, pages 48-56.

[Chatain,2009] Chatain, T., David, A., Larsen, K.G., 2009, *Playing games with timed games*, Proceedings of the 3rd IFAC Conference on Analysis and Design of Hybrid Systems, Zaragoza, Spain.

[Chatterjee,2015] Chatterjee, K., Prabhu ,V.S., 2015, *Quantitative Temporal Simulation and Refinement Distances for Timed Systems*, IEEE Transactions on Automatic Control, Volume 60, Issue 9, pages 2291-2306.

[Chatterjee,2012] Chatterjee, K, Sabourian, H., 2012, *Game Theory and Strategic Complexity*, Computational Complexity, pages 1292-1308.

[Chatterjee,2005] Chatterjee, K., Henzinger, T. A., Jurdzinski, M., 2005, *Mean-payoff parity games*, Proceedings of IEEE LICS, pages. 178-187.

[Cimatti,2005] Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M., 2005, *NUSMV: a new symbolic model checker*, International Journal on Software Tools for Technology Transfer, Volume 2, Issue 4, pages410-425.

[Clark,2013] Clark, M., Koutsoukos, X., Kumar, R., Lee, I., Pappas, G. J., Lee, P., Porter, J., Sokolsky, O., 2013, *A Study on Run Time Assurance for Complex Cyber Physical Systems*, Interim Technical Report, AFRL/RQQA, NTIS Issue No 13.

[Clarke,2000] Clarke, E.M., Grumberg,O., Peled, D.O., 2000, *Introduction to Model Checking*, MIT Press Cambridge, MA, USA, ISBN:0-262-03270-8.

[Clayton,1999] Clayton M.J., Teicholz P., Fischer, M., Kunz, J., 1999, *Virtual components consisting of form, function and behaviour*, Automation in Construction, Volume 8, Issue 3, pages 351–367.

[Colombo,2007] Colombo, G., Mosca, A., Sartori, F., 2007, *Towards the design of intelligent CAD systems: An ontological approach*, Advanced Engineering Informatics, Volume 21, Issue 2, pages 153–168.

[Cousot,1992] Cousot, P, 1992, *Abstract Interpretation Frameworks*, Journal of Logic and Computation, Volume 2, pages 511-514.

[Cowder,2003] Crowder, R., Bracewell, R., Hughes, G., Kerr, M., Knott, D., Moss, M., Clegg, C., Hall, W., Wallace, K., and Waterson, P., 2003, *A Future Vision For The Engineering Design Environment: A Future Sociotechnical Scenario*. 14th International Conference on Engineering Design, The Design Society, pages 249-250.

[CRESCENDO, 2009] CRESCENDO, 2009, *Requirements Enabling Virtual Testing as Means of Compliance for Certification-CRESCENDO* FP7 234344 Report.

[Dahab,2008] Dahab, M.Y., Hassan, H., Rafea, A., 2008, *TextOntoEx: Automatic ontology construction from natural English text*, Expert Systems with Applications, Volume 34, Issue 2, Pages 1474–1480.

[Deng,2002] Deng, Y., M., 2002, *Function and behaviour representation in conceptual mechanical design*, Artif. Intell. Eng. Des. Anal. Manuf, Volume 16, Issue 5, pages 343-362.

[DO178C] DO-178C, 2011, *Software Considerations in Airborne Systems and Equipment Certification*, RTCA DO-178C, December 2011.547.

[Donzè,2014] Donzè, A., 2014, *On Signal Temporal Logic*, Lecture Notes in Computer Science, Volume 8174, pages 382-383.

[Donzè,2013] Donzè, A., 2013, *Specification Mining of Industrial-scale Control Systems*, Talk, University of California, Berkley.

[DOORS] DOORS, 2014, *IBM Rational DOORS Requirements Management* Framework Add-on User manual, Release 6.1.

[Duffy,1991] Duffy, D. A., 1991, *Principles of Automated Theorem Proving*, John Wiley & Sons.

[Durak,2016] Durak, U., Ören, T., 2016, *Towards an ontology for simulation systems engineering*, Proceedings of the 49th Annual Simulation Symposium (ANSS '16), Society for Computer Simulation International, San Diego, CA, USA, , Article 13, 8 pages.

[FAA,2002] FAA, 2002, *Joint Aircraft System/Component Code*, Table and Definitions, Light Standards Service Regulatory Support Division, Aviation Data Systems Branch, AFS-620, FAA Flight Standards Service, 72 pages.

[Fishwick,1993] Fishwick, P.A., Narayanan N.H., Sticklen J., Bonarini A., 1993. *A Multimodal Approach to Reasoning and Simulation*. IEEE Transactions on Systems, Man and Cybernetics, Volume 24, No 10.

[Fitzgerald,2013] Fitzgerald, J., Larsen, P. G., Pierce, K., Verhoef, M., 2013, *A Formal Approach to Collaborative Modelling and Co-simulation for Embedded Systems*, Mathematical Structures in Computer Science, vol. 23, no. 4, pages 726-750.

[FMI,2010] FMI, 2010, *Functional Mock-up Interface for Co-Simulation*, MODELISAR, ITEA 2 – 07006.

[Foures,2013] Foures, D., Albert, V., Nkesta, A., 2013, *Simulation validation using the compatibility between simulation model and experimental frame*, Proceedings of the 2013 Summer Computer Simulation Conference, Society for Modeling & Simulation International, Vista, CA, Article 55.

[Foukou,2016] Foukou, G., Jean, S., Hadjali, A., Baron, M., 2016, *Handling Failing RDF Queries: From Diagnosis to Relaxation*, Knowledge and Information Systems (KAIS 2016), pages 1-29.

[Frantz,1995] Frantz, F. K., 1995, *A taxonomy of model abstraction techniques*, Proceedings of the 27th conference on winter simulation, pages 1413-1420, Arlington, Virginia, United States.

[Gaines,1979] Gaines, B., 1979, *General systems research: quo vadis?,* General Systems: Yearbook of the Society for General Systems Research, Vol.24, 1979, pages 1-9.

[Ganter,2005] Ganter, B., Stumme, G., Wille, R., 2005, Formal Concept Analysis. Foundations and Applications, Springer, 2005.

[Gero,2004] Gero, J.S., Kannengiesser U., 2004. *The situated function-behaviour-structure framework*. Design Studies, 25(4), 373–391.

[Giambasi,2003] Giambiasi, N., 2003, *From Timed Automata to DEVS models*, Proceedings of the Winter Simulation Conference, Volume 1, pages 923-931.

[Graves,2008] Graves, H., Horrocks, I., 2008, *Application of OWL 1.1 to Systems Engineering*, OWL Experiences and Directions April Workshop.

[Girard,2005] Girard, A., 2005, *Reachability of Uncertain Linear Systems Using Zonotopes*, Hybrid Systems: Computation and Control, Volume 3414 of the series Lecture Notes in Computer Science pages 291-305.

[Girard,2007] Girard, A., 2007, *A composition theorem for bisimulation functions*, Technical Report, Université Joseph Fourier, France.

[Girard,2007] Girard, A., Pappas G J, 2007, *Approximate bisimulation relations for constrained linear systems*, Automatica, Volume 43 Issue 8, pages 1307-1317.

[Girard,2007] Girard, A., Pappas G J, 2007, *Approximation Metrics for Discrete and Continuous Systems*, IEEE Transactions on Automatic Control, Volume 52, Issue 5, pages 782-798.

[Goel,2009] Goel, A., Rugaber, S., Vattam, S., 2009. *Structure, behaviour, and function of complex systems: The structure, behaviour, and function modeling language*, Journal of Artificial Intelligence for Engineering Design, Analysis and Manufacturing archive, Volume 23, Issue 1, pages 23-35.

[Grädel,2002] Grädel, E., Thomas, W., Wilke, T., 2002, *Automata Logics, and Infinite Games - A Guide to Current Research*, Lecture Notes in Computer Science, Volume 2500.

[Greves,2009] Greves, H., 2009, *Integrating SysML & OWL*. Proceedings of OWL: Experiences and Directions, Volume 529, pages 117-124.

[Grosof,2003] Grosof, B.N., Horrocks, I., Volz, R., Decker, S., 2003, *Description Logic Programs: Combining Logic Programs with Description Logic*. In Proc. of the Twelfth International World Wide Web Conference, pages 48-57.

[Gross,1999] Gross, D., 1999, *Report from the Fidelity Implementation Study Group*, Simulation Interoperatbility Workshop, USA.

[Haghverdi,2005] Haghverdi, E, Tabuada, P and Pappas, G.J, 2005, *Bisimulation relations for dynamical, control, and hybrid systems*, Theoretical Computer Science, Volume 342, pages 229-261.

[Haskins,2011] Haskins, C., Forsberg, K., 2011, *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities,* INCOSE-TP-2003-002-03.2.1.

[Hays,1980] Hays, R.T., 1980, *Simulator Fidelity: A concept Paper*, Technical Report 490, US Army Research Institute for Behavioural & Social Sciences, DTIC.

[Henzinger,2013] Henzinger, T.A., 2013, *Quantitative reactive modeling and verification*, Journal of Computer Science, Volume 28 Issue 4, Pages 331-344.

[Henzinger,2005] Henzinger, T.A., Majumdar, R., Prabhu, V., 2005, *Quantifying similarities between timed systems*. Proceedings of the Third International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), Lecture Notes in Computer Science 3829, Springer, 226-241.

[IEEE,2007] IEEE, 2007, 1362-1998 - *IEEE Guide for Information Technology - System Definition - Concept of Operations (ConOps) Document*.

[IEEE,1990] IEEE, 1990, 610.12-1990 - *IEEE Standard Glossary of Software Engineering Terminology*.

[Ilieva,2005] Ilieva M.G., Ormandjieva O., 2005. *Automatic Transition of Natural Language Software Requirements Specification into Formal Presentation*, Natural Language Processing and Information Systems. Lecture Notes in Computer Science Volume 3513, pages 392-397.

[INCOSE,2011] INCOSE, 2011, *Systems Engineering Handbook*, Version 3.2.2.

[Jahanian,1994] Jahanian F., Mok A.K., 1994. *Modechart: a specification language for real-time systems*, IEEE Transactions on Software Engineering, vol.20, no.12, pages 933-947. DOI: 10.1109/32.368134.

[Jahanian,1988] Jahanian F., Mok A.K., Stuart, D.A., 1988, *Formal Specification of Real-time Systems*, Technical Report.

[Jenkins,2012] Jenkins S., Rouqette N., 2012. *Semantically-rigorous systems engineering using SysML and OWL*. International Workshop on System & Concurrent Engineering for Space Applications, Lisbon, Portugal.

[Julius,2004] Julius, A.A., Schaft van der, A.J., 2004, *State maps of general behaviours, their lattice structure and bisimulations*, 16th International Symposium on Mathematical Theory of Networks and Systems, Leuven, Belgium.

[Kezadri,2012] Kezadri, M., Pantel, M., 2012, *First steps toward a Verification and validation ontology*, International Conference on Embedded Real Time Software and Systems (ERTS2 2012), Toulouse, France.

[Kim,2004] Kim, H., 2004, *Reference model based high fidelity simulation modeling for manufacturing systems*, Phd Thesis, Georgia Institute of Technology.

[Kinkade,1972] Kinkade, R.G. Wheaton, G.R., 1972, *Training device design, Human Engineering Guide to Equipment Design*, Washington, D.C.: American Institutes for Research.

[Kuipers,2001] Kuipers, B., 2001. *Qualitative Simulation. Encyclopedia of Physical Science and Technology*, Third Edition, Academic Press.

[Kupferman,2000] Kupferman, O., Vardi, M.,Y., Wolper, P., 2000, *An Automata Theoretic Approach to Branching-Time Model Checking*, JACM, Volume 47, Issue 2, pages 312–360.

[Kwiatowska,2001] Kwiatkowska, M., Norman, G., Parker, D., 2001, *PRISM: Probabilistic Symbolic Model Checker*. Proceedings of PAPM/PROBMIV'01 Tools Session, pages 7-12.

[Labiche,2005] Labiche, Y., Wainer, G., 2005, *Towards the verification and validation of DEVS models*, Proceedings of 1st Open International Conference on Modeling & Simulation, Pages 295-305.

[Lacy,2004] Lacy, L. W., Gerber, William J., 2004, *Potential Modeling and Simulation Applications of the Web Ontology Language – OWL*, Proceedings of the Winter Simulation Conference, Washington D.C., DOI: 10.1109/WSC.2004.1371325

[Lehmann,2013] Lehmann, A., 2013, *Introduction to the special issue on verification, validation and accreditation in modeling and simulation*, The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, 10(4): p. 345-346.

[Levy,1997] Levy, A.Y., Iwasaki Y., Fikes R., 1997. *Automated model selection for simulation based on relevance reasoning.* Artificial Intelligence, Volume 96, Issue 2, pages 351–394.

[Lickly,2011] Lickly, B., Shelton, C., Latronico, E., and Lee E, 2011, *A Practical Ontology Framework for Static Model Analysis*, In Proceedings of the ninth ACM international conference on Embedded software, New York, USA, pages 23-32.

[Lynch,1988] Lynch, N.A., Tuttle, M.R., 1988, *An introduction to input/output automata*, CWI-Quarterly, Volume 2, Number 3, pages 219–246.

[Maler,2002] Maler, O., 2002, *Control from Computer Science*, IFAC Annual Review in Control 26(2), pages 175-187.

[Maler,2008] Maler, O., Batt, G., 2008, *Approximating Continuous Systems by Timed Automata*, Lecture Notes in Computer Science, Volume 5054, pages 77-89.

[Man,2009] Man-Kit-Leung, J., Mandl, T., Lee, E., Latronico, E., Shelton, C., Tripakis, S., Lickly, B., 2009. *Scalable semantic annotation using lattice based ontologies*. Lecture Notes in Computer Science, Volume 5795, pages 393-407.

[Marninchi,1998] Maraninchi, F., Rémond, Y., 1998. *Mode-Automata: About Modes and States for Reactive Systems*. Proceedings of European Symposium on Programming (ESOP), Lisbon, Portugal.

[Mazo,2010] Mazo, M., Davitan, A., Tabuada, P., 2010, *PESSOA: a tool for embedded controller synthesis*, Proceedings of the 22nd International Conference on Computer Aided Verification, pages 566-569.

[Miller,2004] Miller, J., A, Baramidze, G., T., Sheth, A., P., Fishwick, P., 2004, *Investigating Ontologies for Simulation Modeling*, In Proceedings of the 37th annual symposium on Simulation, IEEE Computer Society, Washington, USA.

[Miller,1999] Miller, J.,A., Sheth, A., P., Kochut, K., J., 1999, *Perspectives in Modeling: Simulation, Database, and Workflow*, Conceptual Modeling, Lecture Notes in Computer Science, Volume 1565, pages 154-167.

[Milner,1989] Milner, R., 1989, *Communication and Concurrency*, Prentice Hall.

[Monceaux,2007] Monceaux, A., Callot, M., 2007, *Use Case: Ontology to Support System Verification Process*, OWLED, Innsbruck, Austria, Volume 258.

[Myerson,1991] Myerson, R., B., 1991, *Game Theory: Analysis of Conflic*t, Harvard University Press.

[Novák,2011] Novák, P., Šindelářo, R., 2011, *Applications of Ontologies for Assembling Simulation Models of Industrial Systems*, On the Move to Meaningful Internet Systems: OTM 2011 Workshop, Lecture Notes in Computer Science, Volume 7046, pages148-157.

[Noy,2001] Noy, N.F., McGuinness, D.L., 2001, Ontology development 101: A guide to creating your first ontology, Stanford knowledge systems laboratory technical report KSL-01-05.

[OMG,2006] OMG SysML, 2006, *OMG Systems Modeling Language*.

[Ören,2014] Ören, T.I., 2014, *The Richness of Modeling and Simulation and an Index of its Body of Knowledge*, Advances in Intelligent Systems and Computing, Vol. 256, Springer, pages 3-24.

[Pace,2004] Pace, D.K., 2004, *Modeling and Simulation Verification and Validation Challenges*, Johns Hopkins APL Technical Digest, Volume 25(2), pages 163-172.

[Pappas,2003] Pappas, G. J., 2003, *Bisimilar linear systems*, Automatica, Volume 39, Issue 12, pages 2035-2047.

[Pappas,2000] Pappas, G.J., Lafferriere, G and Sastry, S, 2000, *Hierarchically consistent control systems*, IEEE Transactions on Automatic Control, Volume 45, No 6, pages 1144–1160.

[Peterson,1981] Peterson, J.L., 1981, *Petri net theory and the modeling of systems*, Prentice Hall PTR Upper Saddle River, NJ, USA.

[Piersall,2014] Piersall, III, C.H., Grange, F.E., 2014, *The Necessity of Intended Use Specification for successful Modeling and Simulation of a System-of-Systems*, Crosstalk, the Journal of Defense Software Engineering, pages 20-24.

[Pnueli,1977] Pnueli, A., 1977, *The temporal logic of programs*, Foundations of Computer Science, The 18th Annual Symposium on Foundations of Computer Science, pages 46–57.

[Pola,2007] Pola, G., Girard, A., Tabuada, P., 2007, *Symbolic models for nonlinear control systems using approximate bisimulation*, The 46th IEEE Conference on Decision and Control, pages 4656-4661.

[Ponnusamy,2016] Ponnusamy, S.S., Albert, V., Thebault, P., 2016, *Consistent behavioural abstractions of experimental frame*, AIAA Modeling & Simulation Technologies Conference, AIAA Scitech, USA, DOI: 10.2514/6.2016-1923.

[Ponnusamy,2016] Ponnusamy, S.S., Albert, V., Thebault, P., 2016, *Towards an Ontology Driven Simulation Model Development*, Proceedings of the 8th European Congress on Embedded Real Time Software and Systems, 11 pages, France.

[Ponnusamy,2016] Ponnusamy, S.S., Albert, V., Thebault, P., 2016, *Simulation Fidelity Distance: A Game-Theoretic Framework*, Proceedings of the Symposium on Theory of Modeling & Simulation, Spring Simulation Multi-Conference, USA.

[Ponnusamy,2016] Ponnusamy, S.S., Thebault, P., Albert, V., 2016, *Quantifying Fidelity for Timed Transition Systems*, Proceedings of the 6th International Conference on Simulation and Modeling Methodologies, Technologies and Applications, Portugal, pages 318-326.

[Ponnusamy,2015] Ponnusamy, S.S., Albert, V., Thebault, P., 2015, *A Meta-Model for Consistent & Automatic Simulation Model Selection*, Proceedings of the 29th European Simulation and Modelling Conference, UK, pages 31-36.

[Ponnusamy,2015] Ponnusamy, S.S, *A Syntactic & Semantic Abstraction Framework*, 2015, The 16th Congress of Ecole Doctorale Systémes, Toulouse.

[Ponnusamy,2014] Ponnusamy, S.S., Albert, V., Thebault, P., 2014, *A Simulation Fidelity Assessment Framework*, Proceedings of the International Conference on Simulation and Modeling Methodologies, Technologies and Applications, Austria, pages 436-471.

[Ponnusamy,2014] Ponnusamy, S.S., Albert, V., Thebault, P., 2014, *Modeling & Simulation Framework for the Inclusion of Simulation Objectives by Abstraction*, Proceedings of the

International Conference on Simulation and Modeling Methodologies, Technologies and Applications, Austria, pages 385-394.

[REQTIFY,2015] REQTIFY, 2015, REQTIFY FD01 User Manual, Dassault Systems.

[Retho,2013] Retho, F., Smaoui, H., Vannier, J.C., Dessante, P., 2013, *A model-based method to support complex system design via systems interactions analysis*, CSDM 2013 Poster Workshop, CEUR -WS Vol-1085.

[Robinson,1997] Robinson, S., 1997, *Simulation Model Verification & Validation: Increasing the User's Confidence*, WSC'97 Proceedings of the 1997 Winter Simulation Conference, IEEE Computer Society Washington, USA, pages 53-59.

[Roques,2016] Roques, P., 2016, *MBSE with the ARCADIA Method and the Capella Tool*, The 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), Toulouse, France.

[Roza,1999] Roza, M., 1999, *Fidelity Requirements Specification: A Process Oriented View*, Fall Simulation Interoperability Workshop.

[Roza,2012] Roza, M., Voogd, J., Sebalj, D., 2012, *The Generic Methodology for Verification and Validation to support acceptance of models, simulations and data*, The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, Volume 10, Issue 4, pages 347-365.

[Roza,2004] Roza, M., 2004, *Simulation Fidelity Theory and Practice: A unified approach to defining, specifying and measuring the realism of simulations*, PhD Thesis, Delfts University Press, The Netherlands.

[Saadawi,2009] Saadawi, H., Wainer, G., 2009, *Verification of Real Time DEVS models*, Proceedings of the 2009 Spring Simulation Multiconference, Artcile No: 143, USA.

[Sage,2000] Sage, A.P., Armstrong, J.E., 2000, *Introduction to systems engineering*. Wiley series in systems engineering. Wiley.

[Sarjoughian,2004] Sarjoughian, H.S and Singh, R.K, 2004, *Building Simulation Modeling Environments Using Systems Theory and Software Architecture Principles*, Advanced Simulation Technology Conference, Washington DC., pages 99-104.

[Sebti,2013] Sebti, M., Girard, A., Gregor, G., 2013, *CoSyMA: a tool for controller synthesis using multi-scale abstractions*, 16th international conference on Hybrid Systems: Computation and Control, HSCC'13, pages 83–88.

[Seidwitz,2003] Seidewitz, E., 2003, *What models mean*, IEEE software, Volume 20, No. 5, pages 26–32.

[Sancandi,2011] Sancandi, M, 2011, *The Validation issue in Modeling and Simulation*, CEA-EDF-INRIA Summer School.

[Simon,1969] Simon, H., 1969, *The Sciences of the Artificial*, MIT Press.

[Sintek,2002] Sintek, M., Decker, S., 2002, *TRIPLE A query, inference, and transformation language for the semantic web*, The Semantic Web ISWC, pages 364–378. Springer.

[SISO,2013] SISO, 2013, GM-VV Vol. 3: *Reference Manual, Reference for Generic Methodology for Verification and Validation (GM-VV) to Support Acceptance of Models*, Simulations and Data, SISO-REF-039-2013, Simulation Interoperability Standards Organization.

[Smits,2013] Smits, G., Pivert, O., Hadjali, A., 2013, *Fuzzy Cardinalities as a Basis to Cooperative Answering*, Flexible Approaches in Data, Information and Knowledge Management, Computational Intelligence, Volume 497, pages 261-289.

[CRYSTAL,2014] State of the art for Aerospace ontology. 2014. CRYSTAL Project, D209.010.

[Stolle,1998] Stolle, R., Bradley E., 1998. *Multimodal Reasoning for Automatic Model Construction*, Proceedings Fifteenth National Conference on Artificial Intelligence, July, 1998, pages 181-188, Madison, Wisconsin, USA.

[Stursberg,2003] Stursberg, O., Krogh, B.H., 2003, *Efficient Representation and Computation of Reachable Sets for Hybrid Systems*, Hybrid systems: Computation and Control, Lecture Notes in Computer Science Volume 2623, pages 482-497.

[Tanner,2003] Tanner, H., Pappas, G. J, 2003, *Abstractions of constrained linear systems*, Proceedings of the American Control Conference, Denver, CO, Volume 4, pages 3381-3386.

[Thebault,2014] Thebault, P., Suquet T., Duffy M., Viaud B., Will P., Cordon J., Lamoliate S., 2014, *Engineering of complex avionics systems simulations using a model based approach.* Proceedings of Embedded Real-time Software and Systems (ERTS) Toulouse, France.

[Thebault,2015] Thebault, P., Ponnusamy, S.S., Albert, V., 2015, *A Multimodal Approach to Simulation Fidelity*, Proceedings of the 12th International Multidisciplinary Modeling & Simulation Multi conference, Italy, pages 34-43.

[Thomas,2002] Thomas, G., E., Wilke, W., T., 2002, *Automata, logics, and infinite games,* Springer, Berlin.

[Tiwari,2003] Tiwari, A., Shankar, N., and Rushby, J., 2003, *Invisible Formal Methods for Embedded Control Systems, Proceedings of the IEEE*, Volume 91. No. 1, pages 29–39.

[Tomlin,2003] Tomlin, C.J., Mitchell, I., Bayen, A., Oishi, M., 2003, *Computational Techniques for the Verification of Hybrid Systems*, Proceedings of the IEEE, Vol. 91, No. 7, pages 986-1001.

[Traoré,2006] Traoré, M.K., Muzzy A., 2006. *Capturing the dual relationship between simulation models and their context*. Simulation Modeling Practice and Theory. Volume 14, Issue 2, pages 126–142.

[Tripakis,2016] Tripakis, S., 2016, *Compositionality in the Science of System Design*, Proceedings of the IEEE Volume 104, Issue 5, pages 960 – 972, DOI:10.1109/ JPROC.2015. 2510366.

[Vardi,2009] Vardi, M., 2009, *Model Checking as A Reachability Problem*, Volume 5797, Lecture Notes in Computer Science, pages 35-35.

[Van,2004] Van der Schaft, A.J., 2004, *Equivalence of dynamical systems by bisimulation*, IEEE Transactions on Automatic Control, Volume.49, No.12, pages 2160-2172.

[Van,2006] Van Breugel, F., Worrell, J., 2006, *Approximating and computing behavioural distances in probabilistic transition systems. Theoretical Computer Science*, Volume 360(1-3), pages 373–385.

[Vincent,2012] Vincent, L., Dunyach, J.-C., Huet, S., Pelissier, G., Merlet, J., 2012, *Towards Application of NASA Standard for Models and Simulations in Aeronautical Design Process*, Proceedings of the DAta Systems In Aerospace, Dubrovnik, Croatia.

[Vu,2015] Vu, L.H., Foures, D., Albert, V., 2015, *ProDEVS: an event-driven modeling and simulation tool for hybrid systems using state diagrams*, Proceedings of the 8th International Conference on Simulation Tools and Techniques, Greece, pages 29-37.

[Wagner,2012] Wagner, D.A., Bennett, M.B., Karban, R., Rouquette N., Jenkins S., Ingham M., 2012. *An ontology for State Analysis: Formalizing the mapping to SysML*. IEEE Aerospace Conference, USA, 10 pages.

[Wainer,2009] Wainer, G., 2009, *Discrete-Event Modeling and Simulation: A Practitioner's Approach*, CRC Press, 520 Pages, ISBN 9781420053364.

[Wainer,2002] Wainer, G., 2002, *CD++: a toolkit to develop DEVS models*, Journal of Software: Practice and Experience, Volume 32, Issue 13, pages 1261-1306.

[Zayas,2010] Zayas, D.S., Monceaux A., Ameur Y.A., 2010. *Models to Reduce the Gap between Heterogeneous Models: Application to Aircraft Systems Engineering*. Proceeding sof ICECCS, pages 355-360.

[Zeigler,2000] Zeigler, B.P., Praehofer H., Tag G.K., 2000, *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic press.

[Zeigler,2000] Zeigler, B.P., Praehofer H., Tag G.K., 2000. *Theory of modeling and simulation*, San Diego, California, USA: Academic Press.

WEB REFERENCES

[TURTLE]	https://www.w3.org/TR/turtle/
[HERMIT]	http://hermit-reasoner.com/
[OWL]	http://owl.man.ac.uk/factplusplus/
[Protégé]	http://protege.stanford.edu/
[NOMAGIC]	http://www.nomagic.com/
[TOPQUADRANT]	http://www.topquadrant.com/tools/ide-topbraid-composer-maestro-edition/
[ISO15926]	http://15926.org/references/
[GATE]	https://gate.ac.uk/
[INTERCAX]	http://intercax.com
[Pidlock]	http://infogrid.org/trac/wiki/Reference/PidcockArticle
[OSKI]	http://www.oskitechnology.com/solutions

LIST OF PUBLICATIONS

The following publications were a part of this thesis and are listed below based on their partial or full contribution to this thesis chapters.

Chapter II – section 3.1, Chapter IV – section 2, Annex – section 6

Ponnusamy, S.S., Albert, V., Thebault, P., 2016, *Consistent behavioural abstractions of experimental frame*, AIAA Modeling & Simulation Technologies Conference, USA, DOI: 10.2514/6.2016-1923.

Chapter I – section 1, 2, Chapter II – section 3.1, Chapter III – section 1, 2, 5.1, 6, Chapter V – section 1

Ponnusamy, S.S., Albert, V., Thebault, P., 2016, *Towards an Ontology Driven Simulation Model Development*, Proceedings of the 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), 11 pages, France.

Chapter II – section 4.1, Chapter IV – section 2, 4, 5.1, Chapter V – section 2.1-2, 6

Ponnusamy, S.S., Albert, V., Thebault, P., 2016, *Simulation Fidelity Distance: A Game-Theoretic Framework*, Spring Simulation Multi-Conference, USA.

Chapter II – section 4.1, Chapter IV – section 2, 4, 5.2, Chapter V – section 2.1, 2.3

Ponnusamy, S.S., Thebault, P., Albert, V., 2016, *Quantifying Fidelity for Timed Transition Systems*, Proceedings of the 6th International Conference on Simulation and Modeling Methodologies, Technologies and Applications, Portugal, pages 318-326.

Chapter I – section 1, 2.1, 2.2, Chapter II – section 2, Chapter III – section 1, 2, 5.1, 6

Thebault, P., Ponnusamy, S.S., Albert, V., 2015, *A Multimodal Approach to Simulation Fidelity*, Proceedings of the 12th International Multidisciplinary Modeling & Simulation Multi conference, pages 34-43, Italy.

Chapter I – section 1.2, Chapter III – section 1, 4.3, 5.2

Ponnusamy, S.S., Albert, V., Thebault, P., 2015, *A Meta-Model for Consistent & Automatic Simulation Model Selection*, Proceedings of the 29th European Simulation and Modelling Conference, pages 31-36, UK.

Chapter I – section 1.1-2, 2, Chapter II – section 1, 3.4, 4, Chapter III – section 1, Annex – section 6.1

Ponnusamy, S.S, *A Syntactic & Semantic Abstraction Framework*, 2015, The 16th Congress of Ecole Doctorale Systémes, Toulouse.

Chapter II – section 4.1, Chapter IV – section 2, 5.3, Chapter V – section 2.4

Ponnusamy, S.S., Albert, V., Thebault, P., 2016, *Experimental Frame Fidelity Distances*, Journal of Simulation Modeling Practice & Theory, Under Preparation.

Chapter I – section 1.1, 1.2, 2, Chapter II – section 1, 3.3-4, Chapter III – section 1, Annex – section 6.1

Ponnusamy, S.S., Albert, V., Thebault, P., 2014, *A Simulation Fidelity Assessment Framework*, Proceedings of the International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), pages 436-471, Austria.

Ponnusamy, S.S., 2014, *A Simulation Fidelity Framework*, Poster, Airbus Phd Day, Bremen, Germany.

Ponnusamy, S.S., 2013, *Simulation Fidelity*, Poster, Airbus Phd Day, Bristol, UK.

Chapter I – section 1.1, 1.2, Chapter II – section 3.3, Chapter IV – section 2

Ponnusamy, S.S., Albert, V., Thebault, P., 2014, *Modeling & Simulation Framework for the Inclusion of Simulation Objectives by Abstraction*, Proceedings of the International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), pages 385-394, Austria.

The following publication is not part of the thesis but the principles were used in **Chapter IV – section 2, 4, 6**.

Albert, V., Ponnusamy, S.S., 2016, *Codage de CDEVS et de PDEVS en réseau de Petri Temporisé: Théorie et Application*, Journées DEVS Francophones 2016, N° 15597, 9 pages, France.

ANNEXURE

1. DOMAIN MODEL IMPLEMENTATION

The domain model concepts and relationships are briefly presented in this section. Some of the important concepts of our ontology introduced in section 4 of chapter III used to build our semi-formal domain model approach is listed below. The description is by no means complete and is given only for the sake of illustration. As remarked in chapter III, the approach defines a methodology to build a MR consistent with TR and SD rather than validate the domain model itself. The concepts, relationships, constraints and axioms used in the approach is preliminary and need to be further studied and agreed upon by the concerned stakeholders. The flexibility of the approach allows to have this validation progressively with ease. However, it must be emphasized that the inclusion approach proposed in this thesis will remain the same irrespective of the ontology contents along with design space exploration mechanisms using queries presented in our approach.

The domain model construction, as remarked in section 4 of chapter III, is based on the follwing

i. Academic state of art such as SBF framework [Gero,2004], [Graves,2008], ontologies in V&V [Kezadri,2010], and state of art system engineering languages such as SysML [OMG], CAPELLA [Roques,2016].
ii. Industrial state of art such as the use of ontologies in (aerospace) industry [Jenkins,2012], in industrial M&S, MBSE approaches in industry especially in V&V activities [Monceaux,2007], [Zayas,2010], [CRYSTAL,2014].
iii. Survey of the existing practices at Airbus on standards, methods, documentations and processes such as
    a. M&S standards for model development and sharing between stakeholders
    b. Multisystem and overall aircraft V&V strategy and processes
    c. Reports from inhouse fidelity improvement projects and working groups with their lessons learnt and recommendations
iv. Questionnaires and interviews with stakeholders such as various system design teams, simulation user teams and model development team including model specialists.
v. Application of SPARQL query and reasoning of the domain model approach, discussed futher in section 2 of this annexure, is driven by the user need. In our study, existing MR for different such systems were studied to find the commonly recurring elements such as requirements and constraints on functions, modes, equipments, their operating condition etc. Then specific queries are developed to extract such information to build a MR. This approach allows saving time in reusing existing queries for different requirements and at the same time is flexible to incorporate a new query.
vi. In addition, scenarios from different TR and design from different SD were studied to include only frequently recurring elements since in reality it is impractical to convert all the knowledge of system design and its operation into a domain model.

The following figures 1.1 to 1.3 give a brief overview of the resulting ontology concepts. The figure 1.1 taken from the ontology editor Protégé illustrates the SBFIO framework.

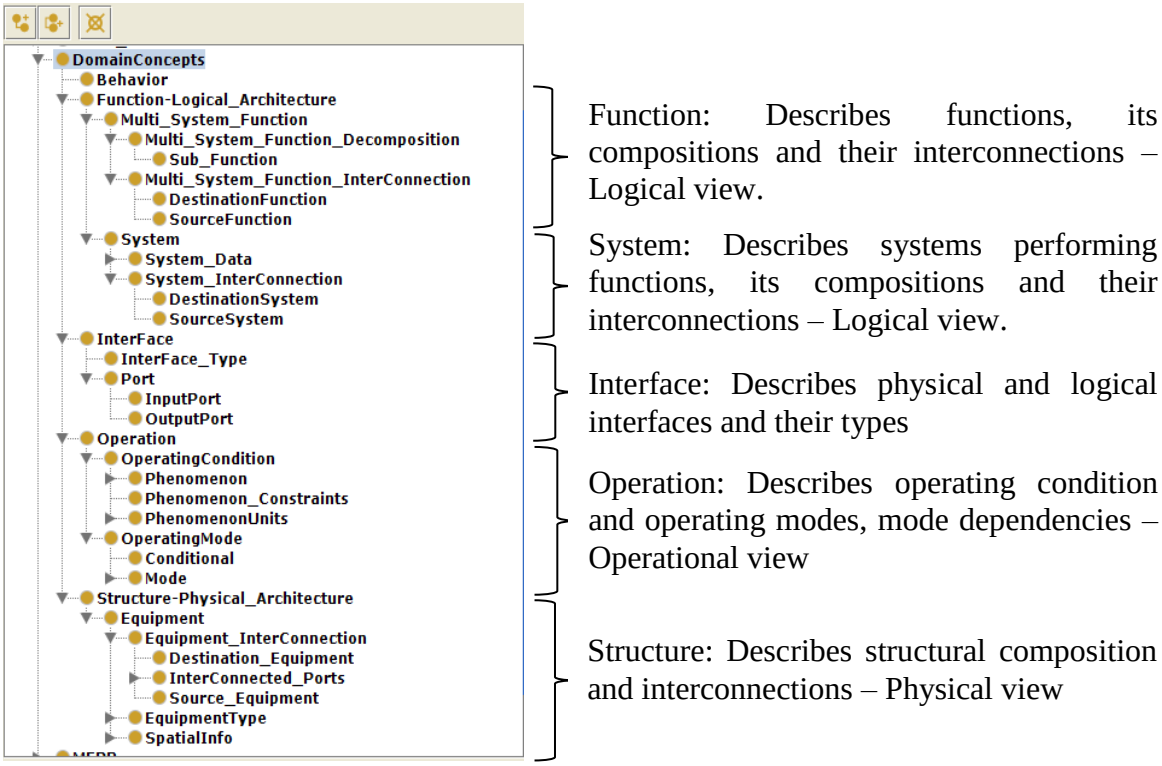


Figure 1.1: SBFIO Ontolgy

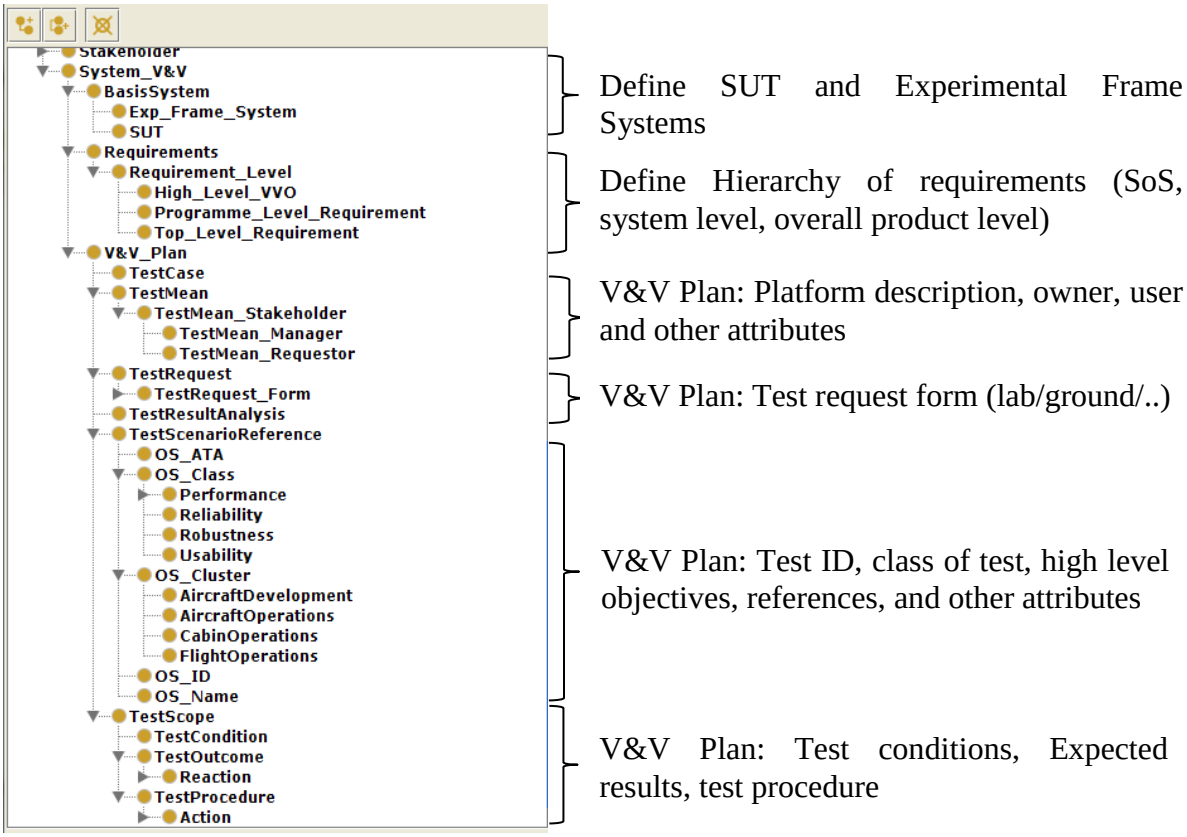


Figure 1.2: Ontology Classes I

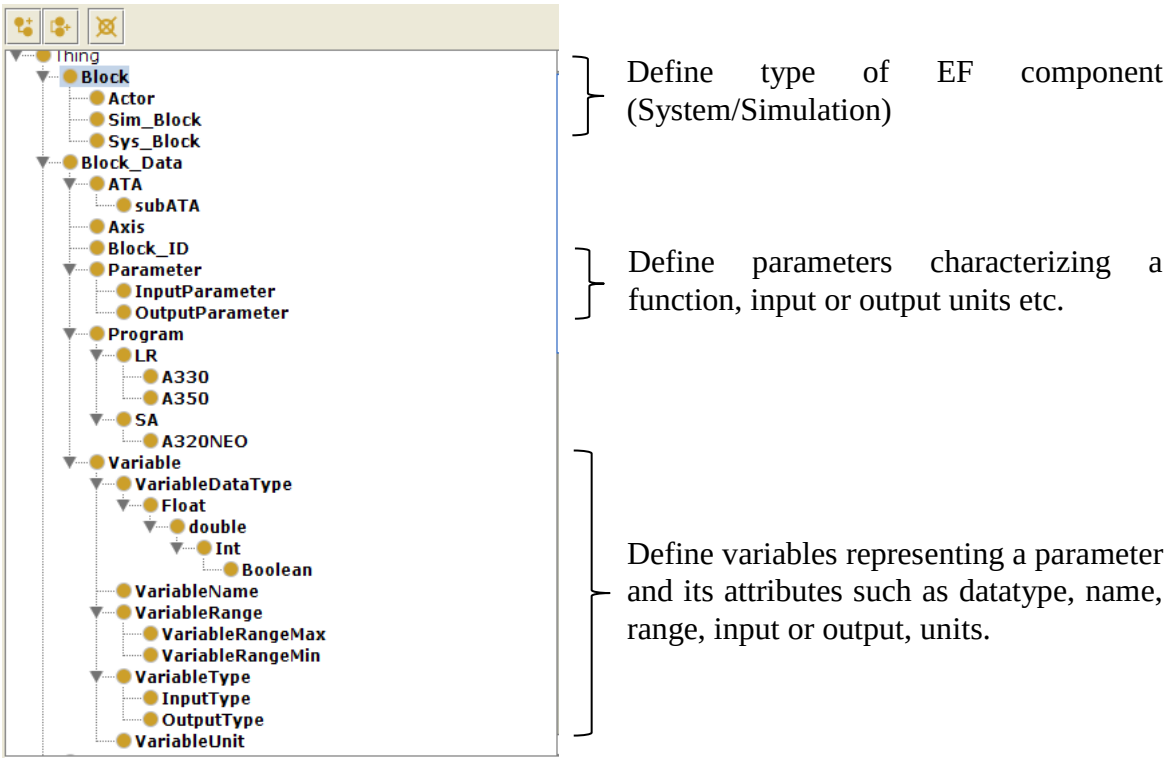


Figure 1.3: Ontology Classes II

The following figure presents properties used in our study. From chapter III and definitions of ontology in Protégé, a property might be object property or a data property. The object properties are illustrated in figure 1.4 whereas some data properties are in figure 1.5. For example, recalling the notation of relationship between concepts as $\mathfrak{E} \xrightarrow{\mathfrak{r}} \mathfrak{E}$, where $\mathfrak{E}$ and $\mathfrak{r}$ are concepts and relationship respectively, then for example, a statement 'a parameter characterises one or more function(s)' is translated as an object property association, *Parameter* $\xrightarrow{characterises}$ *Function*. Such property association could be of type functional (one-to-one relation), transitive, inverse. Similarly, data properties can be defined, for example, a statement 'an equipment located in rear of an engine' is translated as, *Equipment* $\xrightarrow{is\ located\ in\ zone}$ *string* where 'rear of engine' string associates the spatial location of the equipment.

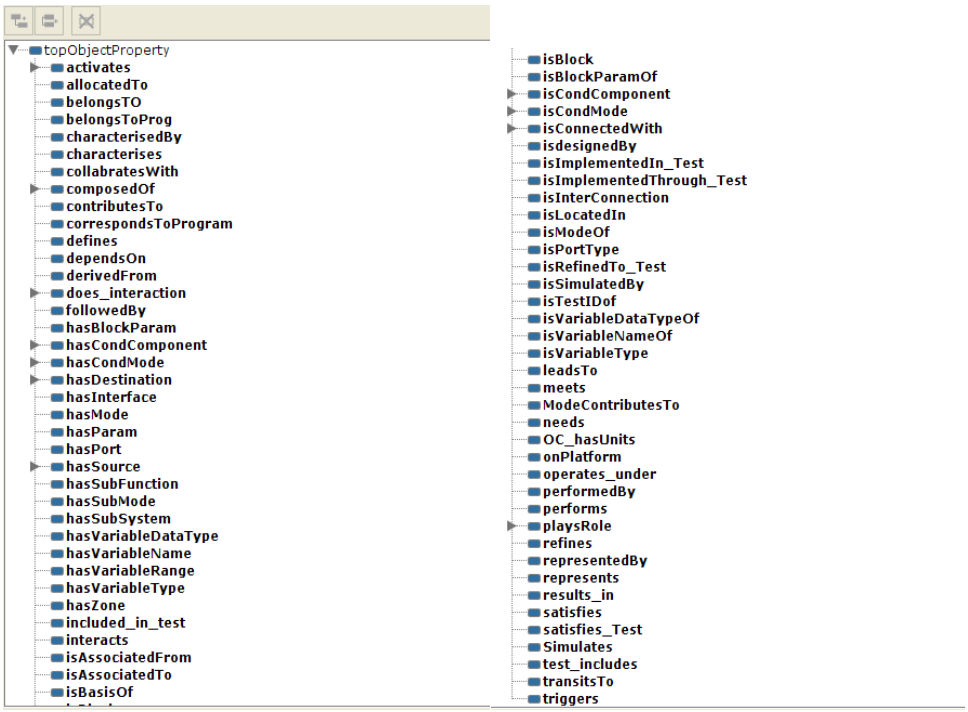


Figure 1.4: Object Properties

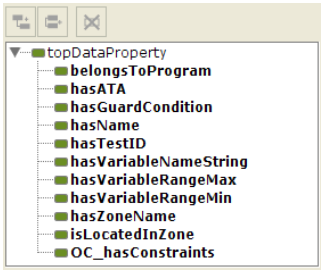


Figure 1.5: Data Properties

The property associations with classes are briefly illustrated in figures 1.6 to 1.7. The classes are marked in yellow circles with predicates in arrow and individuals of classes in violet diamonds.

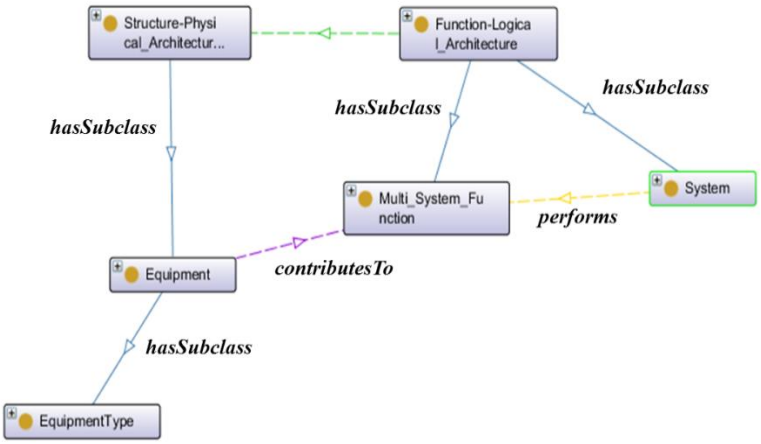


Figure 1.6: System-Function-Equipment Ontology

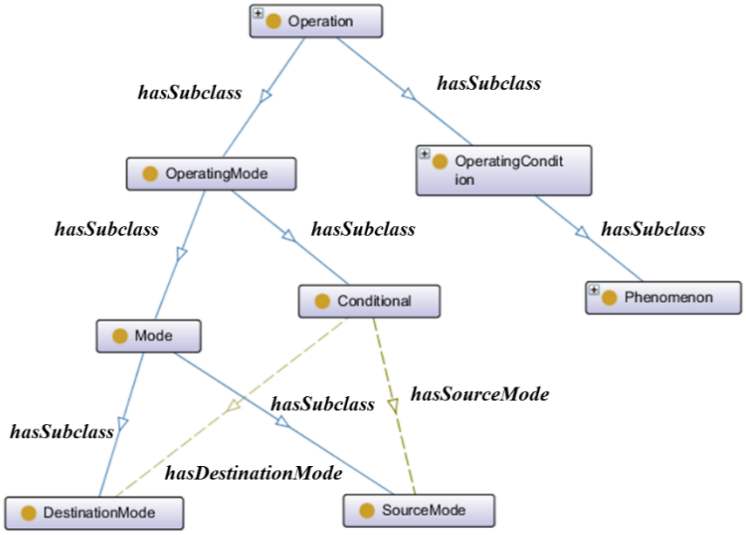


Figure 1.7: Operation Ontology

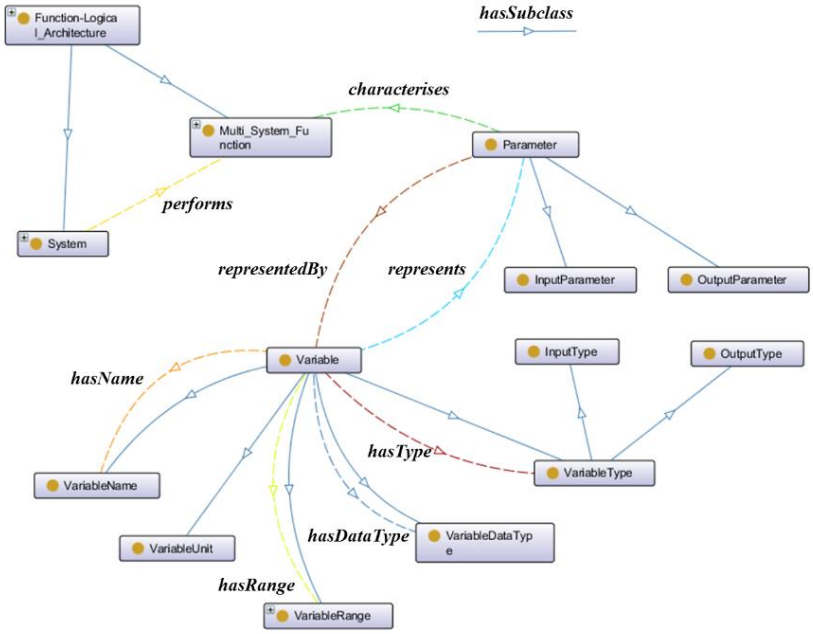


Figure 1.8: Parameter-Variable-System-Function Ontology

Some metrics on the part of our domain model obtained from the Protégé tool is listed below. Some key figures are listed in left pane and associated metrics in the right pane. It can be seen that the size of this part of domain model is relatively small with about 160 classes and 116 properties which may change according to its (in)adequacy on the application case. The size of the instantiated domain model vary between about 3600 to 9300 triples with 3620 triples for unfilled domain model followed by 4797 for filled i.e. instantiated domain model and 9295 after reasoning. Another measure yields 998 for unfilled, 1027 for unfilled and reasoned, 1239 for filled and 2046 for filled and reasoned instances for NAIS model. However, it must be noted that despite this being a measure it changes and evolves over time according to added or modified knowledge.

It is worth noting from section 1 of chapter V that though the innate scalability of ontology is appealing, care must be taken to build an ontology with sufficient tractable complexity. A highly complex ontology and domain model will result in practical difficulties in instantiation, query development and ontology management thus obviating the very purpose of reducing the complexities in building a MR with sufficient fidelity.

Axiom: 1850 Logical axiom count: 1430 Declaration axioms count: 356 Class count: 160 Object property count: 105 Data property count: 11 Individual count: 92	DL expressivity: SROIQ(D) SubClassOf: 196 EquivalentClasses: 2 DisjointClasses: 7 GCI count: 0 Hidden GCI Count: 2 SubObjectPropertyOf: 29 EquivalentObjectProperties: 0 InverseObjectProperties: 18 DisjointObjectProperties: 0

	FunctionalObjectProperty: 15 InverseFunctionalObjectProperty: 8 TransitiveObjectProperty: 17 SymmetricObjectProperty: 1 AsymmetricObjectProperty: 1 ReflexiveObjectProperty: 0 IrrefexiveObjectProperty: 1 ObjectPropertyDomain: 72 ObjectPropertyRange: 74 SubPropertyChainOf: 0 SubDataPropertyOf: 0 EquivalentDataProperties: 0 DisjointDataProperties: 0 FunctionalDataProperty: 6 DataPropertyDomain: 9 DataPropertyRange: 11 ClassAssertion: 415 ObjectPropertyAssertion: 526 DataPropertyAssertion: 21 NegativeObjectPropertyAssertion: 0 NegativeDataPropertyAssertion: 0 SameIndividual: 1 DifferentIndividuals: 0 AnnotationAssertion: 64 AnnotationPropertyDomain: 0 AnnotationPropertyRangeOf: 0

Figure 1.9: Domain Model Properties

2. SPARQL QUERY IMPLEMENTATION

In this section, some of the sample queries used for design space exploration and verification is listed.

2.1 MODEL ASSEMBLY EXAMPLE

A sample query to compare three simulation models input interface with system model input interface similar to figure 5.4 of chapter III. In this case, two of the models have same parameter (e.g. Force, F) but different datatypes (e.g: double, int); The query below first matches the models having same parameters then list the lattice length so that the element having closest distance can be chosen as the best abstraction [Ponnusamy,2016].

```
PREFIX mm:<http://instantiated model name***.owl#>

PREFIX nn:<http:// instantiated model name ***_infered.owl#>

#the query needs to be customized to suit the respective class, object and data properties respectively

SELECT  DISTINCT ?iclist ?source ?dest ?system_var_out_name ?system_var_in_name ?sim_var_in_name ?sim_block
(COUNT(DISTINCT ?sim_var_class) AS ?sim_var_class_no)

WHERE

(

#List all system Interconnection

 ?iclist rdf:type mm:Block_InterConnection;

      mm:connectsFrom ?source;

      mm:connectsTo ?dest.

#check Source Port and Destination Port have same variable names eg:F for force

?source_port a mm:SourcePort;                    owl:sameAs ?system_port_out.?system_port_out mm:isAssociatedTo
?system_param_out. ?system_param_out mm:representedBy ?system_var_out. ?system_var_out mm:hasVariableName
?system_var_name. ?system_var_name mm:hasVariableNameString ?system_var_out_name.

?dest_port a mm:DestinationPort;                    owl:sameAs ?system_port_in.

?system_port_in mm:isAssociatedTo ?system_param_in.?system_param_in mm:representedBy ?system_var_in. ?system_var_in
mm:hasVariableName ?system_var_name1. ?system_var_name1 mm:hasVariableNameString ?system_var_in_name.

#check variable datatypes

FILTER(CONTAINS(?system_var_out_name, ?system_var_in_name))

?sim_block mm:Simulates ?source;          mm:hasBlockParam ?q.

 ?q a mm:InputParameter. ?q mm:representedBy ?b. ?b mm:hasVariableName ?d.  ?b mm:hasVariableDataType ?jj. ?ii
rdfs:subClassOf* mm:VariableDataType. ?jj a ?sim_var_class. ?d mm:hasVariableNameString ?sim_var_in_name.

FILTER(CONTAINS(?sim_var_in_name, ?system_var_in_name) )

?ii rdfs:subClassOf* mm:VariableDataType.

?jj a ?sim_var_class.

)

GROUP BY ?sim_block ?iclist ?source ?dest ?system_var_out_name ?system_var_in_name ?sim_var_in_name
```

2.2 DESIGN SPACE EXPLORATION EXAMPLE

A query corresponding to the design space exploration described in section is given below.

```
PREFIX mm:<http://www.simulation_library.org/simulation_library_ontology.owl#>
SELECT DISTINCT ?c_req ?c_avb  ?c_avb_1 ?m_avb_1
WHERE {
?m a mm:TestScope.
?m a mm:OperatingMode.
?c_req mm:hasMode ?m.
?c_req mm:hasName ?n1.
?s mm:composedOf ?c_avb.
?s mm:isdesignedBy ?sd.
?c_avb mm:hasName ?n2.
?c_avb mm:hasMode ?m_avb.
?c_avb mm:isConnectedFrom ?c_avb_1.
?c_avb_1 mm:hasMode ?m_avb_1.
?s mm:composedOf ?c_avb_1.
?s mm:isdesignedBy ?sd.
}
```

3. AUTOMATED MODEL SELECTION ALGORITHM IMPLEMENTATION

The model selection algorithm of [Levy,1997] is implemented in SysML for the automated model selection presented in the domain model approach section 5.2.1 of chapter III. SysML is chosen since it may become a defacto standard in MBSE framework and in future our ontology based approach serves as complimentary to such framework. It could be possible to transform ontology models to SysML and vice versa, such as using OWL2UML plugin in Protégé 4.1 or by other transformation mechanisms which are being studied [Jenkins,2012] and then initiating the algorithm. The SysML implementation consists of block diagrams to define the domain model and activity diagrams for the description of the algorithm. The modeling tool used is MagicDraw SysML [NoMagic] with its Cameo Simulation Toolkit plugin for the execution of built models, in our case execution of activity diagram over the instantiated domain model. It may be noted that an activity diagram specifies input to output transformation through controlled sequence of actions and the model selection algorithm is formalized in it and executed. An iterative expansion region is used for list iterations, 'readStructuralFeature' and 'addStructuralFeature Value' actions for attribute's getters and setters of classes, call behaviour actions for modularity and reusability of functions, merge and decision nodes for choices and conditions. Though the activity diagram is not presented, model selection and the domain theory as block diagram is illustrated below [Ponnusamy,2015].

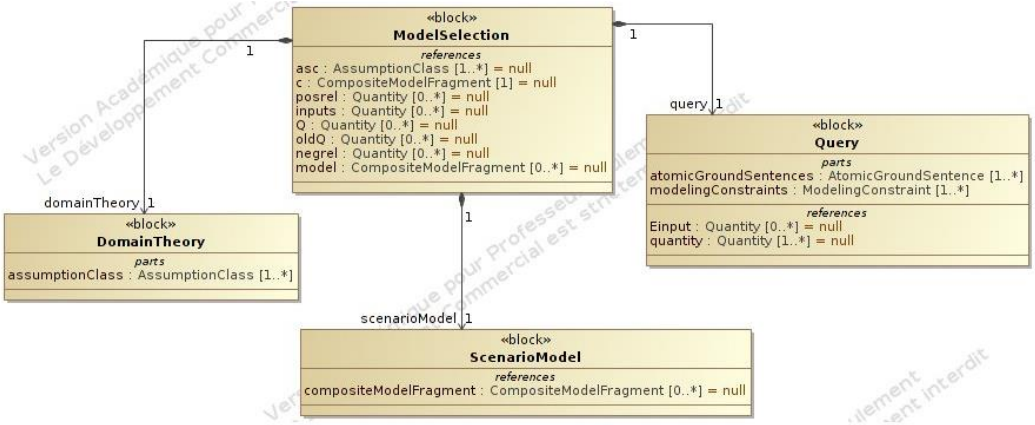


Figure 3.1: Model Selection SysML Block Diagram

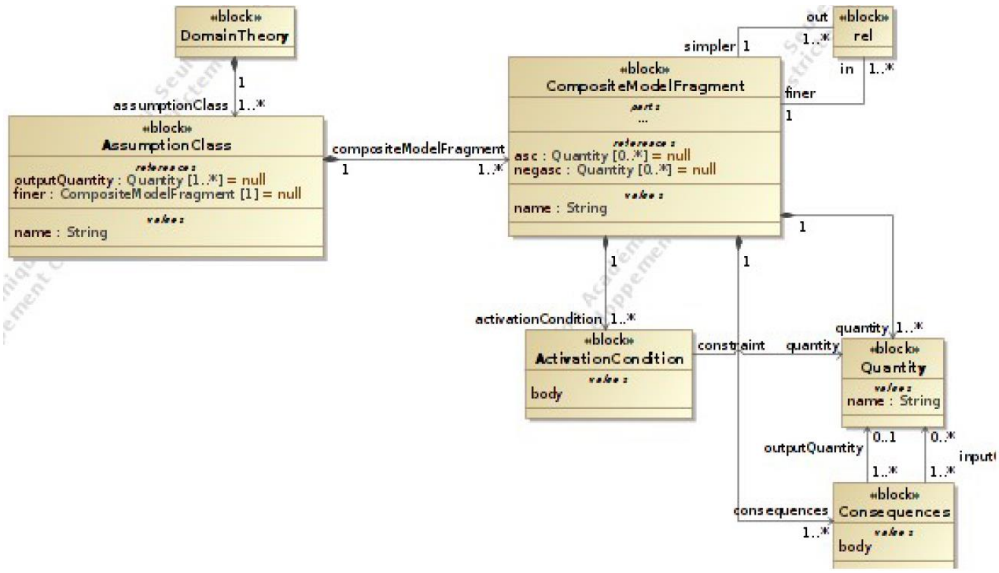


Figure 3.2: Domain Theory SysML Block Diagram

However, it must be noted that the implementation is not validated especially for its complexity, scalability and further works needed in this aspect. Alternatively, other general purpose languages could be used to implement such an algorithm.

4. PETRINET IMPLEMENTATION

4.1 AUTOMATED GAME CONSTRUCTION

The construction of games manually is a cumbersome and often error prone process especially when the models are large. The ProDEVS software, introduced in section 6 of chapter IV, allows the automated construction of games between the system and simulation models, or generically between any two models modeled under the same formalism. This is a process of autofusion where the two models are fused with the game model represented by $\mathfrak{g}$. The label containment verification step allows to find the intersection between two models labels which will later be used to build psuedo models (for untimed games). These two pseudo models are chained with the game model resulting in $\mathfrak{g}(M_{sys}, M_{sim})$. Since the distance is calculated indirectly based on the comparison between data value of each transition, the associated data file is also constructed automatically using random or user defined values for each transition. From the label containment verification, only the common labels between two models are assigned the same value. Then the data embedding is done by compiling the *.c file to *.dll file which is then used by TINA to initiate the game and generate the quantitative reachability graph. The output will be a *.txt file which is then fed back to ProDEVS parser to reconstruct the reachability tree and perform some analytics

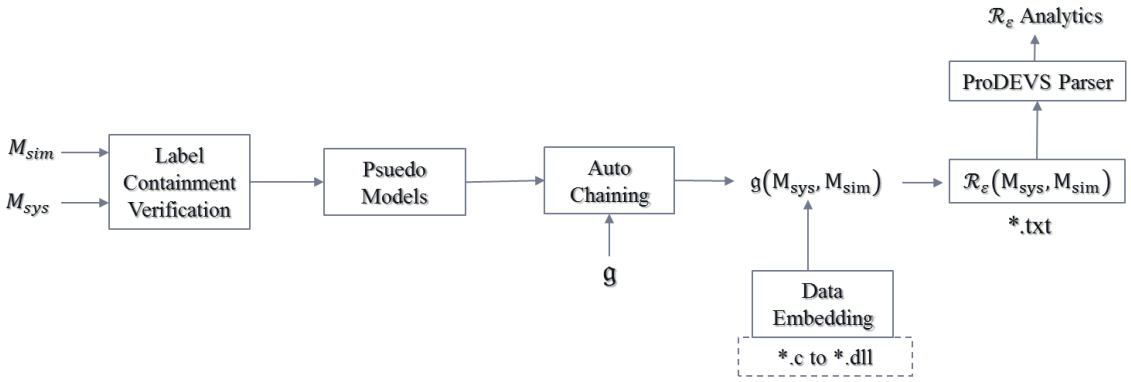


Figure 4.1: Automated Game Construction Process

The untimed games implementation for both the classical and interface automata in Petrinet formalism in explained necessitates vioding spurious or redundant plays by the player 2 in cases of branching in system and/or simulation model. This is taken into account by building an intersection set which gets updated dynamically. The intersection is introduced as follows, from the definitions of $T_{sys} =< \Sigma_{sys}, X_{sys}, x^0_{sys}, \delta_{sys} >$ and $T_{sim} =< \Sigma_{sim}, X_{sim}, x^0_{sim}, \delta_{sim} >$, let us define the intersection set between the set of alphabets as $\Sigma_{int} = \Sigma_{sys} \cap \Sigma_{sim}$ and the set of transitions from a given state $x^i_{sim} \in X_{sim}$ as $\tau_{x^i_{sim}} \in \delta_{sim}$. Then, denoting a label and state of a transition τ by $\xi \in \Sigma$ and $x \in X$ respectively leads to,

$$\forall \xi_{sys} \in \tau_{sys} \wedge \xi_{sys} \in \Sigma_{int}, e(\rho_i) = \begin{cases} 0 & \text{if } \xi_j \in \tau_{x^i_{sim}} \wedge \xi_j = \xi_{sys} \wedge \xi_j \in \Sigma_{int} \\ \infty & \text{if } \xi_j \in \tau_{x^i_{sim}} \wedge \xi_j \neq \xi_{sys} \wedge \xi_j \in \Sigma_{int} \end{cases} \quad (1)$$

where j refers to the number of transitions from a given state x_{sim}^{i} and i refers to number of such states. The intersection is managed in Petrinet by adding extra i.e. psuedo transitions and places for the transitions at the intersection. The static transition label intersection set, Σ_{int} is given as an input along with two models which then gets updated as the game progresses. This helps in avoiding redundant plays where the simulation model can match the system model but in addition plays the other available transition which in reality is redundant. Such conditions may also be encoded in the underlying data files as guard conditions on simulation model transitions. Though these methods may reduce the number of redudant trajectories being generated, adding pseudo transitions increases the compexity of the game. However, there is no other method known at this stage to manage this problem. This is applicable to untimed games only and since for timed games the distance is calculated only on transition timings.

4.2 UNTIMED GAMES IMPLEMENTATION

The fidelity quantification for untimed simulation models based on automata games is presented in this section. Let us recall the notions, system model, M_{sys} and simulation model, M_{sim} and the two players be player 1 and player 2. The game is described as follows,

1. Player 1 plays on M_{sys} and player 2 plays on M_{sim}
2. Player 1 first plays its transition, then hands token to player 2
3. Player 2 plays its transition and then hands back token to player 1. The play is complete now.
4. Error is calculated at the end of each play and then next play begins.

There is a place for each player with player 1's place marked initially along with a marked place to denote the start of turn. The *cheat* or *nocheat* transitions are mutually exclusive taken according to the equivalence of labels at the end of a given play. This is amenable to calculate the mean distance such as in Eq.(7,8) of section 4.2 in chapter IV.

Table 4.1: Untimed Game Model Description

Description	Player 1	Player 2	Play Start	Play End	Matching transition	Non-matching transition
Type	Place	Place	Place	Place	Transition	Transition
Name	*player1*	*player2*	*turn_start*	*turn_end*	*nocheat*	*cheat*

The rules for automatically chaining the game model illustrated in figure 4.2 with user defined system and simulation models is given as follows,

1. All transitions of system model, M_{sys} has two input and one output.
2. All transitions of simulation model, M_{sim} has one input and two output.

Every transition of M_{sys} and M_{sim} are connected to this game model as follows, for example every transition of M_{sys} is connected *from* two places namely 'player1' and 'turn_start' i.e. two inputs and connected *to* a place 'player2' i.e. one output. This is described in the following table.

Table 4.2: Untimed Game Auto-chain Rule

	M_{sys}	M_{sim}
From	player1, turn_start	player2
To	player2	player1, turn_end

The resulting auto-chained model where the game is played is illustrated for the $g(M_{sys}, M_{sim}^{1})$ described in the application case in section 2.1-2 of chapter V.

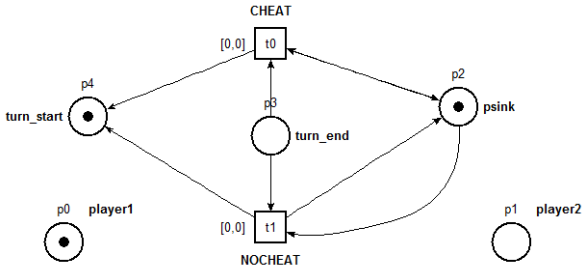


Figure 4.2: Untimed Game Model

In the following figure, an implementation of the game of the application case is shown. The system and simulation model can be seen chained along with the game model which controls the turn based semantics of the game.

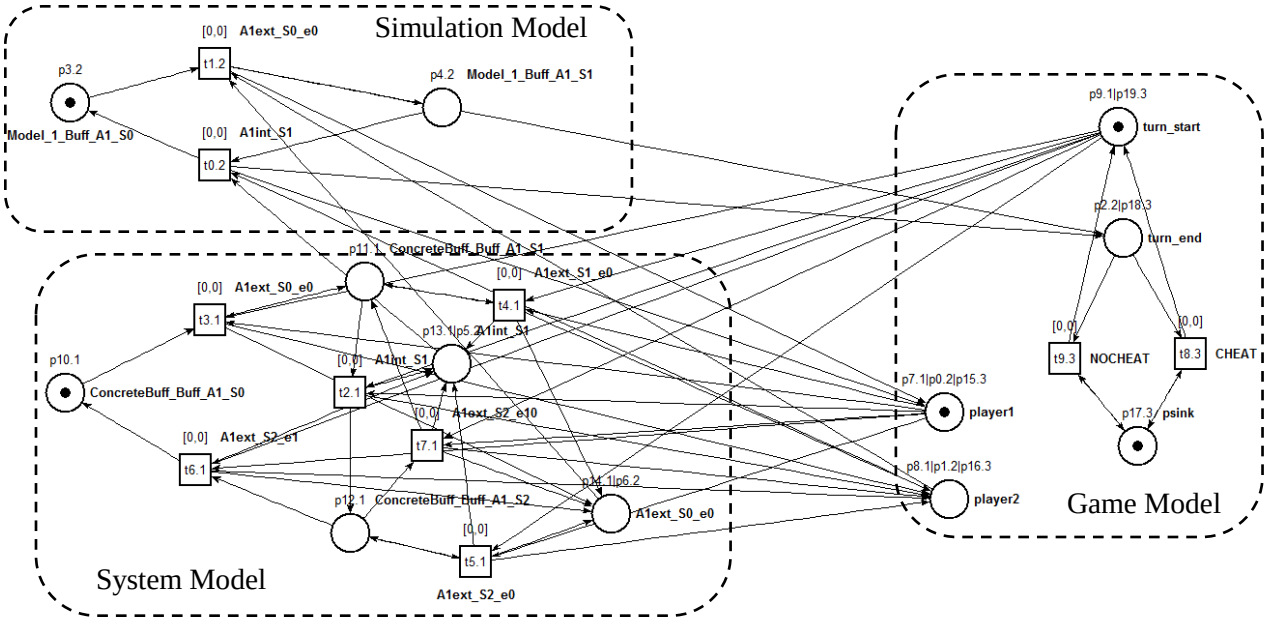


Figure 4.3: Untimed Game Implementation

4.3 TIMED GAMES IMPLEMENTATION

The timed games implementation and the quantitative reachability generation is presented in this section. Let us recall the notions system model, M_{sys} and simulation model, M_{sim} with players be player 1 and player 2. Then the play is sequentially defined in the following steps as defined in section 5.2 of chapter IV.

1. Player 1 first plays its transition on M_{sys} and then hands token to player 2 **and** keeps playing
2. Concurrently, Player 2 plays its transition on M_{sim} iff the label matches and then hands back token to player 1.
3. The play is complete now. Error is calculated at the end of each play and then next play begins. Note player 1 could be ahead by many transitions.

The description for the places and transitions of this game implementation in TINA-ND are given below,

Table 4.3: Timed Game Model Description

Description	Player 1 place	Player 2 place	Play End place	Non-matching transition	Total Play Count
Type	Place	place	place	transition	place
Variable name	*player1*	*player2*	*turn_end*	*cheat*	*total_play_count*

The game model is automatically chained with the user defined M_{sys} and using the following rules.

1. All transitions of system model, M_{sys} has one *read* type input and one output.
2. All transitions of simulation model, M_{sim} has one *regular* type input and two output.

Table 4.4: Untimed Game Auto-chain Rule

Transitions	M_{sys}	M_{sim}
Inputs	*player1*	*player2*
Outputs	*player2*	*player1, turn_end*

3. The transition *cheat* has one input from place *turn_end* and one output to place *total_play_count* with time [0,0].

It may be noted that unlike untimed games there is no need for *turn_start* as player 1 plays continuously. Informally, a game is lost when the player 2 has no moves or cannot match labels. And the game is won when the player 2 has a matching move exactly i.e. no time difference or approximately i.e. with time difference where the error starts increasing. In the game, player 2 first checks whether player 1 move's label is available, if exists, it always takes it and this checking is based on the intersection verification described in previous section. The game model and the auto-chained model for the case $\mathfrak{g}(M_{sys}, M_{sim}^2)$ of section 2.3 of chapter V are illustrated in figures 4.4 and 4.5 respectively.

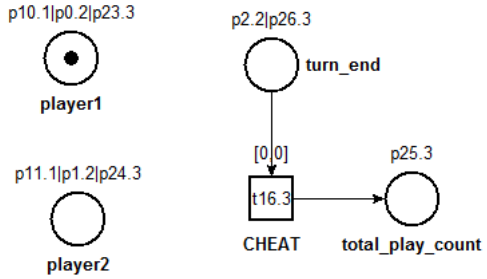


Figure 4.4: Timed Games Model

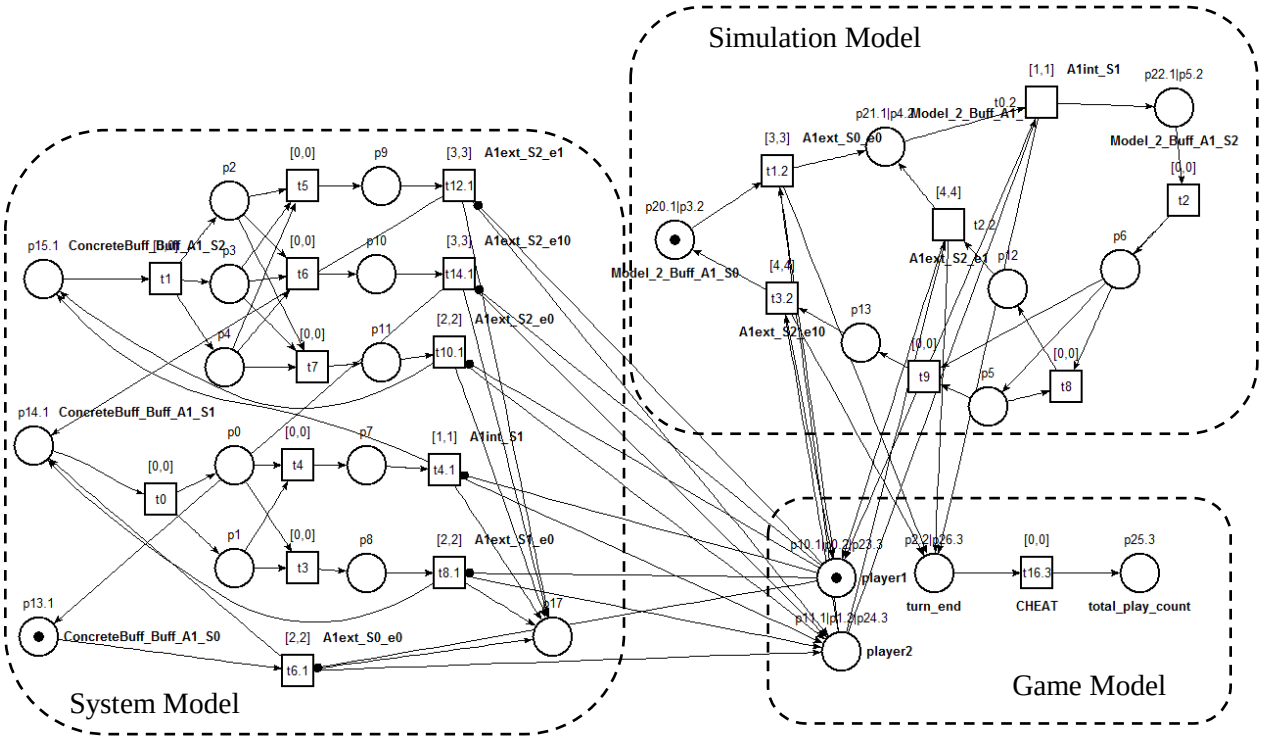


Figure 4.5: Timed Games Implementation

4.4 UNTIMED INTERFACE GAMES IMPLEMENTATION

The untimed interface game distinguish between input and output transitions and hence the game is alternating play as discussed in section 5.3 of chapter IV. This game is defined sequentially as follows,

1. Player 1 can play on input of M_{sys} or output of M_{sim} and hands over the turn to player 2.
2. Player 2 can play on input of M_{sim} or output of M_{sys}
3. Error is calculated at the end of each play and then next play begins.

In addition to places in the table 4.1 for untimed automata games, two more places in and out connected to input and output transitions of models respectively in the interface games. The game model can be seen in the following figure,

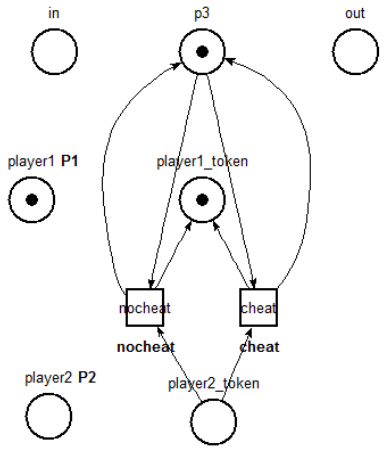


Figure 4.6: Untimed Interface Game Model

The rules for the auto-chain with the user defined suystem and simulation model are given below

1. All transitions of M_{sys} and M_{sim} has two input and two output.

Table 4.5: Untimed Interface Game Auto-chain Rule

	M_{sys}		M_{sim}	
	Input	Output	Input	Output
From	player1 turn_start	player2 out	player2 in	player1 turn_start
To	player2 in	player1 turn_end	player1 turn_end	player2 out

2. At start, only player1 and turn_start has token.
3. The transition nocheat and cheat has one input from place turn_end and one output to place turn_start
4. In addition, the transition nocheat and cheat has one input and output to a single place called p_sink

The fidelity distance i.e. error calculation is similar to untimed games. An illustration of the NAI application case for the game $\mathfrak{g}(\mathrm{M_{sys}}, \mathrm{M_{sim}^1})$ discussed in section 2.4 of chapter V is given below.

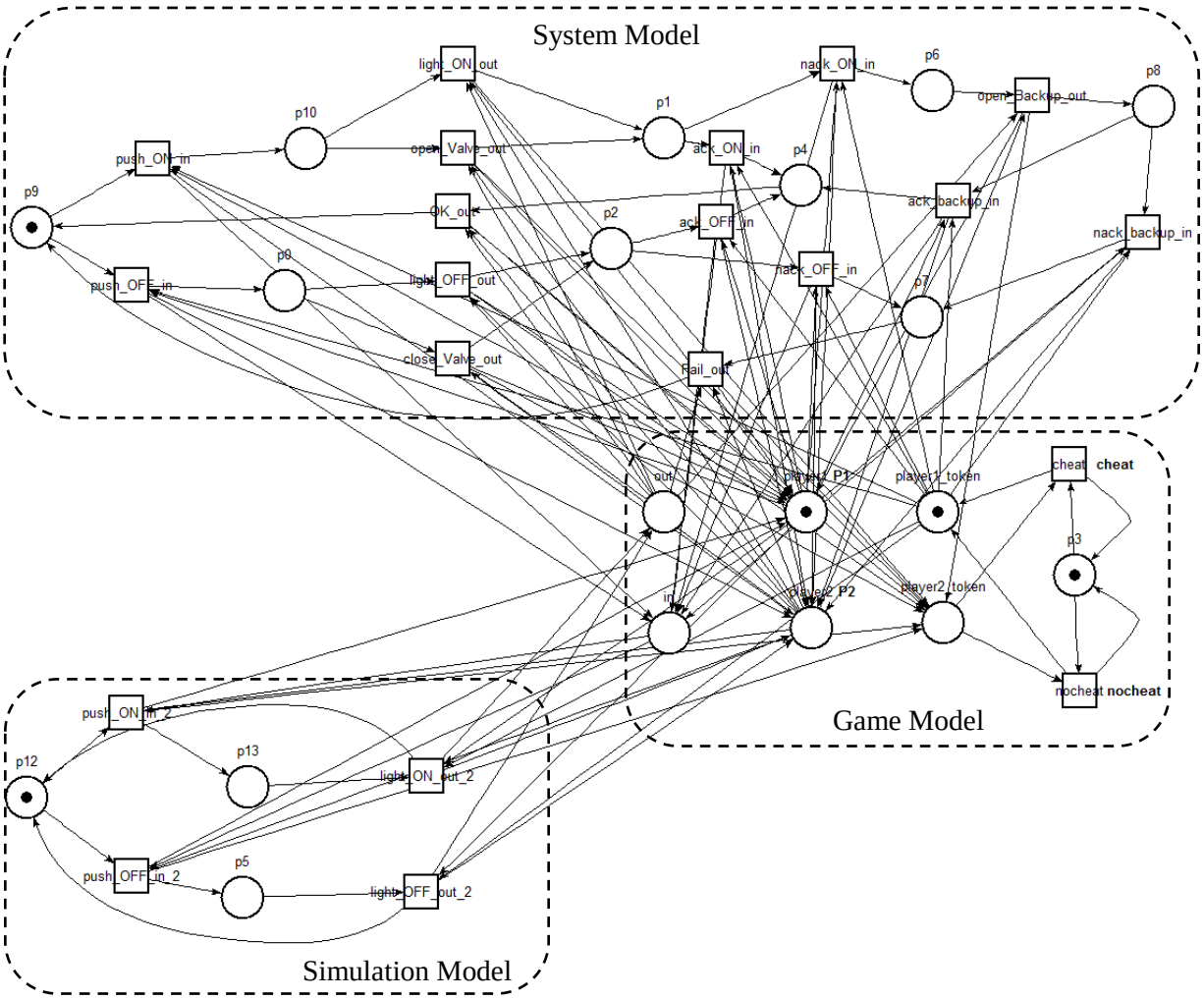


Figure 4.7: NAIS Interface Fidelity Game Implementation

5. REACHABILITY TREE RECONSTRUCTION

The parser used to reconstruct the output reachability graph from TINA is briefly presented in this section. The (pseudo) reachability tree reconstruction is given followed by a part of parser analytics. The parser reconstructs the reachability tree until termination i.e. available data generated from TINA or until user defined depth. Informally, the exploration starts from initial state defined by 'trans_init' and the successor transition label identified as 'label' and located at 'label_node' in the data is iteratvely read and stored in reachability tree variable 'R' along with the corresponding error 'error'. In the course of exploration, locations where branches occur are stored and once the exploration reaches the end, from the branch location data 'branch_list' the algorithm switches the initial location to the last location where the branch occurred and from it the next branch is taken and explored further. This process is repeated until all the branches are explored. The pseudo-algorithm implemented as linked list is given below. It must be noted that the performance of the algorithm is not evaluated and benchmarked yet.

```
// Initial values
Init: trace=0, trans=1,trans_init=1,j=1;k=2;id=1;branch_list_end(1)=0;dum=2;branch_list=();

//Iterative exploration

While trace>0
  for trans=trans_init:depth
     if (isempty(label)==true)
        terminate
     end
     list.node(trans+1).value=label;
     label_node = label_node_list(id);
     if (isempty(label_node)==true)
        output('insufficient classes : try generating more classes ')
        terminate;
      end
      list.node(trans+1).next=label_node+1;
      label_node_len=length(label_node);
     if(label_node_len>1)&&(trans_init!=trans)
        branch_list(j)=[trans length(label_node):-1:2];
         j=j+1;
      end
      R(trans+1,trace,1)=d1(id);
       R(trans+1,trace,2)=error(1);
```

```
 end

 end

  if(isempty(branch_list)==true)
     output('R graph generation over ')
     break
  end

 branch_list_end=branch_list(end);
  if(length(branch_list_end)>1)
    id=branch_list_end(end);
    trans_init=branch_list_end(1);
    trace=trace+1;
    R(1:trans_init,trace,1)=R(1:trans_init,trace-1,1);
    R(1:trans_init,trace,2)=R(1:trans_init,trace-1,2);
    branch_list(end)=branch_list_end(1:end-1);
    if(length(branch_list(end))==1)
      branch_list(end)=[];
      j=j-1;
    end

  end
 end
 trace=trace+1;
end

// Analytics
Output('Total no of trajectories generated  :   ', Column_size(R));
Output('Length of trajectories generated  :  ', Row_size(R));
Output('Number of plays :  ',Row_size(R)/3)); // only for untimed games
epsilon=100-R(3:3:end,:,2)); // extract error data from R tree
plot(epsilon);
plot(minimum_epsilon_values);
```

```
plot(maximum_epsilon_values);
C_mean = epsilon(end);
//fidelity distribution
max_fid=max(Cmeans);
min_fid=min(Cmeans);
histogram(Cmeans);
```

6. BEHAVIOURAL FIDELITY FOR CONTINOUS SYSTEMS

6.1 APPROXIMATE BISIMULATION RELATIONS

Bisimulation, originally introduced in computer science has been brought to control [Van,2004] and it is a notion that unifies state space equivalence and reduction. Approximate Bisimulation relations developed by Pappas, Girard et al in [Girard,2007] within the framework of metric transition systems extends this notion of bisimulation to continuous systems similar to definitions of Eq(7,8).

A metric transition system is a transition system whose outputs are equipped with a metric such as the Euclidean distance. Consider two transition systems $\prod_{1,2}$ which essentially refer to a system specification and its abstraction defined in Eq.(6) as,

$$\prod_n =< S_n, X_n, \tau_n, S_n^0, Y_n, O_n >, n=1,2 \tag{1}$$

where
S_n are the set of states
X_n are the set of inputs
τ_n are the transition maps, $\tau_n : S_n \times X_n \rightarrow 2^{S_n}$
S_n^0 are the set of initial states $S_n^0 \subseteq S_n$
O_n are the output maps $O_n : S_n \rightarrow Y_n$ equipped with a metric d.

The Approximate Bisimulaion (AB) relations are intended to capture the most significant characteristics of a system dynamics and neglect the less important ones [Girard,2007] similar to quantitative simulation relations. The degree of approximation is given by the precision of the AB function (ε) and this precision provides a bound of the distance between the output trajectories of a system and its abstraction. The set of output trajectories, $\{(Y,X) \mid Y=O(S)\}$, denoted by $\mathcal{L}(T)$ is called the language of the transition system, $\prod$. The behavioural equivalence between two systems described by homomorphism relation is given here in terms of the observational equivalence i.e. language inclusion and equivalence.

From [Girard,2007], two metric transition systems $\prod_1$ and $\prod_2$ are said to be bisimilar with a precision ε, if there exists bisimulation relation, f_ε and for all $(s_1, s_2) \in f_\varepsilon$

$$d\big(O_1(s_1), O_2(s_2)\big) \leq \varepsilon \tag{2}$$
$$\{ \forall x \in X, \forall s_1' \in S_1(s_1,x), \exists s_2' \in S_2(s_2,x) \mid (s_1',s_2') \in f_\varepsilon \}$$
$$\{ \forall x \in X, \forall s_2' \in S_2(s_2,x), \exists s_1' \in S_1(s_1,x) \mid (s_1',s_2') \in f_\varepsilon \}$$

Such bisimulation relations could be expressed as bisimulation function, f_B. The function $f_B: S_1 \times S_2 \rightarrow \mathbb{R}^+$ is a bisimulation function between $\prod_1$ and $\prod_2$, if for all $(s_1, s_2) \in S_1 \times S_2$

$$f_B(s_1, s_2) \geq \max \left\{ \begin{array}{c} d\big(O_1(s_1), O_2(s_2)\big), \\ \sup_{x \in X} \inf_{s_1' \in S_1(s_1,x)} \inf_{s_2' \in S_2(s_2,x)} f_B(s_1', s_2'), \\ \sup_{x \in X} \inf_{s_2' \in S_2(s_2,x)} \inf_{s_1' \in S_1(s_1,x)} f_B(s_1', s_2') \end{array} \right\} \quad (3)$$

where the bisimulation function f_B bounds the distance between the observations for a couple (s_1, s_2) by precision $\varepsilon \geq 0$ such that $f_B(s_1, s_2) \leq \varepsilon$ and non-increasing under operational dynamic conditions [Girard,2007].

Thus the AB notions for continuous systems are similar to quantitative (alternating) simulation relations with only difference being how the state space is enumerated i.e. explicitly in discrete case and through geometric over approximation in the case of continuous systems. While the focus for discrete systems is developing a mechanism to formally and explicitly quantify the fidelity, both globally and locally, the focus for continuous systems is formalization of such (bi)simulation relations in the V&V context globally and specifically for a class of linear continuous systems. In the following sections, some consistency conditions for continuous systems abstraction is briefly presented.

6.2 CONSISTENT BEHAVIOURAL ABSTRACTIONS

The fidelity distance, also known as abstraction precision in section 4.1 of chapter II between a concrete system and its abstraction is related to fidelity. This implies that abstraction is finding a surjection map and valid abstraction is finding the surjection map consistent with the SOU. To recall, Surjection, also called abstraction in general, is an onto mapping where the codomain of a function is also its range. A function $f: A \rightarrow B$ is a surjection means that every $b \in B$ is in the range of f. Surjective functions here are used in mapping entities from high dimensional space to a low dimensional subspace. Based on the propositions given in section 2, the following proposition can be stated [Ponnusamy,2016].

Proposition 6.1: For a given fidelity requirement, defined over some metric, δ_F, the set of valid abstractions are given by

$$\alpha_{SDU} \mapsto \varepsilon_{SDU} \mid \varepsilon_{SDU} \leq \varepsilon_{SOU} \quad (4)$$

and the representative abstractions are given by

$$\alpha_{SDU} \mapsto \varepsilon_{SDU} \mid \varepsilon_{SDU} \sim \varepsilon_{SOU} \quad (5)$$

From the definition of abstraction as surjection, it follows that valid and/or representative abstractions are member of the lattice, $\mathcal{L}_{\alpha_i}$.

It is well known from the behavioural systems theory that, the concept of states, similar to the one described in Eq.(1), is used to capture the necessary information about the evolution of a system. In other words, the problem stated above is how to partition state space of a dynamic system

such that its abstracted semantics allows to conclude about its concrete semantics. Here the concrete semantics is assumed to represent real system. In [Julius,2004], Julius et al presented systems behaviour through state maps and states that relationship between bisimulation and lattice structure of the state maps. The abstractions or state map reductions are discussed through equivalence classes as position in the lattice. This study can be seen as continuation of this proposition and extends this bisimilarity preserving abstraction operation to the experimental frame formalism.

An important challenge of the system abstraction is the computation of bisimulation functions $f_B(s_1, s_2)$ between the system and its abstraction. This problem was addressed for constrained linear time invariant systems through the introduction of quadratic and truncated quadratic bisimulation functions [Girard,2007]. It was remarked that the choice of surjection map to form an abstraction prior to constructing the approximate bisimulation function is heuristics based, provided, it respects the observation preserving and controlled invariant properties [Girard,2007]. However, different such surjections are possible for the same level of abstraction leading to different precisions and if a certain precision is being desired by the user of the model, then it becomes important to relate this expected precision with this choice of surjection.

Thus, instead of choosing an admissible surjective map such that the precision of abstraction formalized as a semi definite optimization problem is minimal, the surjection map must be chosen such that the precision of abstraction is arbitrarily closer to the required precision. The existence of such an admissible surjection map gives the Necessary & Sufficient (N&S) abstraction consistent with the simulation objectives defined through precision, ε_{SOU}. From Eq.(4), the valid or necessary abstractions (N) i.e. the set of all admissiable surjections necessary for validity assessment is rewritten in terms of bisimulation function yielding precision ε_{SDU} as follows

$$\alpha_N := \alpha_{SDU} \mapsto \{ \varepsilon_{SOU} - f_B(s_1, s_2) \} \geq 0 \tag{6}$$

In conjunction with Eq.(5) & (6), the sufficiency condition gives the abstraction with minimal granularity i.e. level of detail with respect to the objectives. Thus the Necessary & Sufficient (N&S) is given by

$$\alpha_{N\&S} = \min(\alpha_N) \tag{7}$$

6.2.1 Model Composition

It is known from EF definitions in chapter II that the models are usually composed hierarchically with interconnections between them and it is necessary to quantify the propagation of the effect of abstraction in composition. Similar to (alternating) simulation relation definitions in chapter IV, due to the transitive nature of bisimulation functions, composition of bisimulation functions is possible, which in turn is amenable to standard engineering approach of hierarchical composition in system development.

Consider two systems which are approximately bisimilar and it follows that their respective abstractions are also bisimilar [Pappas,2003]. This entails that composition of resulting abstracted systems also will be bisimilar [Girard,2007] and thus providing bounds on the EF composed of

different such systems. This morphism relation, similar to the one shown in [Julius,2004], can be seen from the following figure,

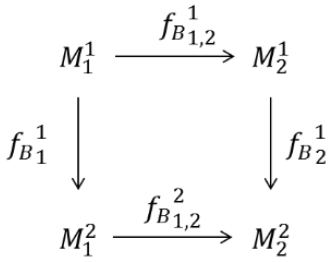


Figure 6.1: Composition of Bisimulations

It can be seen that the morphism relations are established between two models ($f_{B_{i,j}}^{n}$) and also among the hierarchies of models ($f_{B_i}^{n}$), where i, j are models and n refers to abstraction operation. These compositional concepts will be further discussed in the context of applicability and derivability of the experimental frames in section 3.2.2-3, of chapter II.

6.3 CONSISTENT ABSTRACTIONS OF LTI SYSTEMS

In this section, a class of dynamic systems, namely, Linear Time Invariant systems with (LTI) constraints on input and states is taken and abstractions consistent with simulation objectives are explained. Consider linear system with constraints on input, $x_{1,2} \in [x_{min}\ x_{max}]$ and state, $s_{1,2} \in [s_{min}\ s_{max}]$ described by state transition matrix $A_{1,2}$ input matrix, $B_{1,2}$ and output matrix $C_{1,2}$ as

$$\dot{s}_1=A_1s_1+B_1x_1 \qquad (8)$$
$$y=C_1s_1$$

and its abstraction

$$\dot{s}_2=A_2s_2+B_2x_2 \qquad (9)$$

where the abstraction is given by the linear surjective map, H, as $s_1 = Hs_2$

In such an abstraction operation, the map must be chosen such that it ensures propagation of desired properties such as stability, controllability and observability between the concrete system and its abstraction. Pappas et al in [Pappas,2000] & Van der Schaft in [Van,2004] proposed the extension of computer science notions of bisimilarity to dynamical systems and characterized it with controlled invariant concepts originally developed in the context of differential geometric control theory. Recalling the conditions for observation preserving partition [Pappas,2003],

$$\mathrm{Ker}(H) \sqsubseteq \mathrm{Ker}(C_1) \qquad (10)$$

The bisimulation conditions for linear continuous systems are given in the following equation

$$A_1\mathrm{Ker}(H) \sqsubseteq \mathrm{Ker}(H) + \mathrm{Im}(A_1,B_1) \qquad (11)$$

where $\mathcal{R}(A_1,B_1)$ gives reachability.

It is remarked that in order for the partition to have the bisimulation property and thereby preserving transition, its null space must be a controlled invariant subspace. Controlled invariant space, introduced in the seminal papers of [Basile,1992], is a stability like notion where in the evolution of constrained dynamical system, constraint satisfaction can be guaranteed for all time *iff* the initial state is contained inside a control invariant set. However, as explained in section, this bisimilarity preserving surjections needs to be augmented with inclusion of objectives in the framework of designed fidelity.

6.3.1 Finding Consistent Surjective Maps

As explained in section 4 of chapter II, Fidelity or more specifically relative Fidelity refers to finding the necessary and sufficient abstraction with respect to requirements. Implicitly, such an abstraction must ensure semantics compatible with behavioural requirements (ε_{SOU}). This section explores the possibility of finding bisimilarity preserving surjections with fidelity using geometrical systems theory concepts.

It has been shown that computing the coarsest bisimulation resulting in maximum complexity reduction corresponds to computing the maximal controlled or reachability invariant subspace inside the kernel of the observations map [Pappas,2000]. A coarsest partition of state space gives maximal controlled invariant set, H_{max} and there exists a fix point algorithm to do it. The key question is the computation of a partition yielding desired precision instead of maximum complexity reduction by coarsest bisimulation. Since maximum controlled invariant set by its definition is the smallest possible set having all the invariant set, the computation of such map will result in an inclusion relation [Ponnusamy,2016]

$$\mathrm{Ker}(H_{SOU}) \sqsubseteq \mathrm{Ker}(H_{max}) \tag{12}$$

where H_{SOU} is the reference user surjection which is mostly unknown and if known, fidelity becomes simply a problem of verification i.e. implementation.

The valid surjection maps are given by Eq.(4). However, quite often, fidelity needs to be optimal and the corresponding abstraction is called is Necessary & Sufficient abstraction. The necessary conditions are given by Eq.(6) and the sufficiency conditions described in Eq.(7) then becomes,

$$\mathrm{Ker}(H_{SOU}) \sim \mathrm{Ker}(H_{SDU}) \tag{13}$$

Besides, a straight forward computation of H_{SOU} may not be feasible since it is very difficult to obtain an abstraction without any knowledge of system functionality which in turn is different in different modes of operation. Instead, the approach should be finding criteria of maps which are valid with respect to requirements in addition to Eq.(12). However, there exists no fix point method to determine minimal controlled invariant set [Pappas,2003] and similar argument can be applied to an invariant set existing in between the two extremum invariant sets.

The surjective criteria in Eq.(12) & also limits the definition of SOU i.e. it gives a feasibility criteria. Considering the set of valid abstractions given by $H_{i=1..n}$ does not include ε_{SOU}, it follows that,

$$H_{SOU} \notin \{H_{i=1..n}\} \tag{14}$$

In other words, the abstraction is not feasible and the desired precision cannot be achieved. It may be noted that, within an abstraction hierarchy, *i*, different abstractions, j=1..l corresponding to different invariant sets respecting bisimilarity conditions too can form a lattice similar to one described in section. But finding the coarsest bisimulation for each order $H_{i,j+1} < H_{i,j}$ might yield an idea about feasible precision such that if $\varepsilon_{max} \sim \varepsilon_f$, the partition is sufficient and if $\varepsilon_{max} > \varepsilon_f$, refinement is needed. The problem of finding surjections whose null space is controlled invariant subspace with respect to fidelity requirements needs to be explained in the M&S standard of experimental frame formalism.

The question then becomes, similar to maximal controlled invariant set, how far a modeler can abstract the components such that the resulting EF is still applicable to SUT (whose validity is being assessed by simulation). In terms of surjection, what are the allowable surjections such that the resulting EF is still applicable to a SUT [Ponnusamy,2016]. In other words, what is the coarsest bisimulation possible between a model and its abstraction such that the abstracted EF is applicable to SUT. Implicit in this statement is the allowable bounds on the abstracted behaviour i.e. abstraction precision. Applicability of an EF can thus be related by quantifying the abstraction.

Thus the problem explained in the previous section can be rephrased as a problem of finding controlled invariance of EF Model being abstracted (i.e. nullspace of surjection map) such that the resulting EF holds applicability property of the concrete EF. A method based these preliminary relations between fidelity, abstractions in EF formalism needs to be developed to address these issues to build models with assured behaviour on bounds and thereby ensuring fidelity.

THE END