A METHODOLOGY FOR CROSS-RESOLUTION MODELING IN DEVS USING EVENT-B REFINEMENT

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

AHMET KARA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

FEBRUARY 2014

Approval of the thesis:

A METHODOLOGY FOR CROSS-RESOLUTION MODELING IN DEVS USING EVENT-B REFINEMENT

submitted by **AHMET KARA** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Halit Oğuztüzün _____
Supervisor, **Computer Engineering Department, METU**

Dr. M. Nedim Alpdemir _____
Co-supervisor, **TÜBİTAK BİLGEM İLTAREN**

Examining Committee Members:

Prof. Dr. Faruk Polat
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering Department, METU _____

Prof. Dr. Veysi İşler
Computer Engineering Department, METU _____

Assist. Prof. Dr. Selim Temizer
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Hürevren Kılıç
Computer Engineering Department, Gediz University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: AHMET KARA

Signature :

ABSTRACT

A METHODOLOGY FOR CROSS-RESOLUTION MODELING IN DEVS USING EVENT-B REFINEMENT

Kara, Ahmet

Ph.D., Department of Computer Engineering

Supervisor : Assoc. Prof. Dr. Halit Oğuztüzün

Co-Supervisor : Dr. M. Nedim Alpdemir

February 2014, 98 pages

This thesis proposes a software engineering solution for implementing simulations via composition of models at different resolution levels with the help of formal methods. Our solution provides a systematic methodology that offers a well-defined sequence of stages to obtain executable converters for entity resolution mapping, given the types of entity attributes that are exchanged at model interfaces and the mapping specifications. Our methodology relies on Event-B as the formal specification language and DEVS as the model composition framework; utilizes refinement relations between Event-B machines for specification, verification and generation of the data conversion steps between models, and employs a code generator that inputs Event-B machine definitions to generate converter code that connects two model ports. Resolution converters for model compositions allows an introduction to use of connector paradigm in modeling and simulation environment. We use our achievements in DEVS converters for implementing DEVS simulations in heterogeneous environments with the help of connectors in the sense of component based software

engineering. This solution involves implementing connectors as atomic models to be used in mediation of data type and time resolution mismatches. Employing atomic models as connectors allows connector composition in the style of Reo and promotes higher level of reuse in simulation construction.

Keywords: Cross-Resolution Modeling; Model Composability; DEVS; Event-B; Connectors; Modeling and Simulation

ÖZ

DEVS İÇİN MELEZ-ÇÖZÜNÜRLÜKLÜ MODELLEMEDE EVENT-B ARITIMI KULLANAN BİR YÖNTEMBİLİMİ

Kara, Ahmet
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Halit Oğuztüzün
Ortak Tez Yöneticisi : Dr. M. Nedim Alpdemir

Şubat 2014 , 98 sayfa

Bu tezde benzetim uygulamalarında farklı çözünürlükteki modellerin biçimsel yöntemler yardımıyla bileştirimi konusunda bir yazılım mühendisliği çözümü sunulmaktadır. Çözümümüz varlıkların çözünürlüklerinin eşleştirilmesi için iyi tanımlı ardışık aşamalar ile çalıştırılabilir dönüştürücü elde eden düzenli bir yöntembilimi sunmaktadır, öyle ki varlıkların cinslerine ait öznitelikleri model arayüzleri ve eşleştirme tanımları ile takas edilmektedir. Yöntembilimimiz Event-B adındaki biçimsel tanımlama dili ve DEVS adındaki model bileştirme çerçevesini ile Event-B makineleri arasındaki arıtma ilişkilerine dayanarak modeller arasındaki veri dönüştürüm adımlarının doğrulanması ve üretilmesi, arıtma ilişkilerinin dönüştürümün tanımlanması ve doğrulanması için kullanılması ile Event-B makine tanımlarını kod üretiminde kullanarak iki modelin birbirine bağlanması için dönüştürücü kodu üretilmesine dayanmaktadır. Model bileştirimi için çözünürlük dönüştürücüleri, modelleme ve benzetim ortamında bağlaştırıcı örneklemi kullanılmasına giriş yapmaktadır. DEVS bağdaştı-

rıcılarından elde ettiğimiz başarıları ayrıcinsten ortamlarda DEVS benzetim uygulamalarının bileşen tabanlı yazılım mühendisliği algısı içerisindeki bağlaştırıcıların yardımıyla geliştirilmesi için de kullandık. Bu çözüm bağlaştırıcıların atomik model olarak uygulanması ile veri cinsi ve zaman çözünürlüğü uyuşmazlıklarına arabuluculuk sunmaktadır. Atomik modellerin bağlaştırıcı olarak uygulanması Reo tarzı bağlaştırıcı bileştirimine olanak vererek benzetim üretiminde daha yüksek seviye yeniden kullanılabilirlik geliştirmektedir.

Anahtar Kelimeler: Melez-Çözünürlüklü Modelleme; Model Bileştirimi; DEVS; Event-B; Bağlaştırıcılar; Modelleme ve Benzetim

Dedicated to my wife, son and daughter

ACKNOWLEDGMENTS

First and foremost I would like to thank my supervisor Assoc. Prof. Dr. Halit Oğuztüzün for all the support and encouragement he gave me. His guidance and positive approach throughout this time period made possible to complete this work.

I would like to express my gratitude and profound respect to my co-advisor Dr. M. Nedim Alpdemir for his suggestions, efforts, morale support and patience during this study. He has made invaluable contributions to my both academic and professional studies.

I acknowledge with thanks and appreciation to the thesis monitoring committee members, Prof. Dr. Veysi İşler and Assoc. Prof. Dr. Hürevren Kılıç for their support and constructive comments on my thesis work.

I would like to thank to the thesis defense jury members, Prof. Dr. Faruk Polat and Assist. Prof. Dr. Selim Temizer for reviewing and evaluating my thesis.

I would like to express my appreciation to TÜBİTAK BİLGEM İLTAREN for the understanding and support during my academic studies. I also want to thank to people I work with for the joy they bring to my life making my business life together with academic life a pleasureful experience.

I thank to my wife, who became the light of life with her patience and morale support during my studies. And I thank to my son and daughter for their patience during my thesis study.

Finally, I would like to express my thanks to my parents, brother and sister, making me who I am now with their love, trust, freedom understanding and every kind of support throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ADG	Activity Dependency Graph
API	Application Programming Interface
BİLGEM	Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezi (Informatics and Information Security Research Center)
CBSE	Component Based Software Engineering
CCG	Converter Code Generator
CPU	Central Processing Unit
CRM	Cross-Resolution Modeling
DCC	Data Conversion Connector
DEVS	Discrete Event System Specification
DFT	Direct Feed Through
DoD	Department of Defense
EMG	Event-B Machine Generator
HLA	High Level Architecture
HRE	High Resolution Entity
İLTAREN	İleri Teknolojiler Araştırma Enstitüsü (Advanced Technologies Research Institute)
KODO	SiMA Kod Oluşturucu (SiMA Code Generator)
LRE	Low Resolution Entity
METU	Middle East Technical University
MR	Multi-Resolution
MRE	Multi-Resolution Entity
MREI	Multi-Resolution Event Interface
MRM	Multi-Resolution Modeling
MRMF	Multi-Resolution Model Family
MRS	Multi-Resolution Space
ODTÜ	Ortadoğu Teknik Üniversitesi (METU)
PO	Proof Obligation
RGB	Red, Green, Blue

RTI	Run-Time Infrastructure
SiMA	Simulation Modeling Architecture
TRC	Time Resolution Connector
TÜBİTAK	Türkiye Bilimsel ve Teknolojik Araştırma Kurumu (Scientific and Technological Research Council of Turkey)
UK	United Kingdom
USA	United States of America
WCF	Windows Communication Foundation
WSN	Wireless Sensor Network
XML	eXtensive Mark-up Language

CHAPTER 1

INTRODUCTION

Simulation can be defined as generation of the behavior or characteristics of one system through the use of another system, especially a computer program. Models and simulations are developed for many reasons such as supporting the decision-making activities of people faced with complex problems. Models can be used to make predictions, evaluate alternatives, gain insight into phenomenology and test different scenarios about real problems. The complexity of the system under study increases the complexity of the modeling, and so does the importance of utilizing ready, re-usable models. The motivation for using ready models is mainly due to a possible reduction in the development effort (and so less time to start experimentation), and also, more importantly, due to appealing comfort of using verified and validated (and if available accredited) models where possible.

Constructing complex software systems through the integration and coordinated interactions of simpler components has long been the focus of many research efforts. As the complexity of software systems increases, the need for systematic approaches increases accordingly. The issue at hand has a number of more specific problem areas, each of which deserves a distinct line of research. In the work presented in this thesis our interest lies in a particular manifestation of this problem that of constructing complex simulation applications by composition. We focus on a relatively subtle but an important challenge for such complex simulation applications which are the cross-resolution simulations involving multi-resolution models [19]. Although different viewpoints may be attached the associated set of problems by different research communities, we argue that the issue of resolution mapping pertaining to externally

visible properties (i.e. in terms of inputs and outputs) of cross-resolution models is essentially a software engineering problem. In fact, the concept of *connectors* [46] in the context of Component-Based Development [38] can be seen as a more general case of *resolution converters* introduced in this thesis. Resolution conversion has peculiarities, mainly characterized by the multi-facets that induce distinct variations to the handling of the more general problem, these will be briefly discussed below. Not surprisingly, those peculiarities also allow us to contemplate the more focused and concrete solutions. As such, based on the analysis and selective targeting of those peculiarities, we will argue that the process of specifying, verifying and generating those converters can be supported by the use of a convenient formal language to develop a rigorous software engineering methodology. Along these lines, our primary motivation at the outset of our research was to develop a well-defined (i.e. based on formal specifications) and repeatable methodology with necessary tool support that diminishes the shortcomings of ad-hoc coding practices mostly adopted for such conversion tasks. Building on our previous research background [7, 26] in terms of both the formal model specification and underlying model composition framework, our solution aims to cover aspects ranging from model and mapping specification to mapping verification and application construction by component composition.

To facilitate a better understanding of the concepts of multi-resolution modeling and the peculiarities mentioned above, we first provide an introductory summary, followed by our key contributions. The reader is referred to our Chapter 2 for further information on those concepts.

1.1 Cross-Resolution Modeling in the Context of Complex Simulation Applications

A modeling enterprise involves capturing characteristics and behavior of systems via mathematical or logical abstractions. Intrinsically, the same real world system can be represented by various abstractions. Depending on the requirements of different stakeholders, those abstractions may particularize specific aspects of the entities being represented or they may delineate varying levels of structural or behavioral information across the same set of aspects. The level of detail at which system components

and their behaviors are represented is generally agreed to indicate the *resolution* of a model which represents that system.

The issue of multiple resolution models pertaining to the same real world system manifests itself in three distinct forms: i - When constructing the model of a system the *modeller* may have to develop multiple representations of the same system, each with a different level of resolution. This is known as *Multi-Resolution Modeling (MRM)*. ii - When developing simulation applications, the *simulationist* may have to compose or orchestrate models with different resolution levels, paying particular attention to resolution conversion requirements. This is known as *Cross Resolution Modeling (CRM)*. iii - When running a simulation scenario and analysing the results of a simulation run, the *analyst* may need to be aware of the existence of multiple resolution models and take the possible cross resolution mapping issues into account when interpreting the results.

The characterization of these issues and the problems that may be attributed to them has already been explored in the relevant literature. For instance, Powell [54] provides a semi-formal description of the central concepts such as resolution, aggregation, disaggregation and consistency maintenance. Davis et al. [23, 22, 24, 25] provide an in-depth coverage of the key terminology and discuss a wide range of aspects delivering considerable insight into the complexity of MRM. An extensively studied problem in the literature is the need for resolution mapping in a CRM setting. The root of this problem is that resolution is a relative concept and exhibits variations across all attributes of a model. So the term 'level of resolution' is not congruously applicable to all the models involved in CRM. Thus, model developers need a methodology to compose models with different levels of resolution. Another root difficulty lies in the fact that the term resolution is a multifaceted concept (Figure 2.2) [22] and to find a solution that is applicable to all facets is a formidable task. Of the six facets given in Figure 2.2, the process, spatial and temporal facets are related to the internal behavioral logic of a model and therefore it is less convenient to deal with them. This is primarily because the model internals can be arbitrarily complex and would typically depend on many artefacts starting from conceptual models and requirements down to the implementation techniques across the development life cycle of the model. Composing cross resolution models based on these facets will require

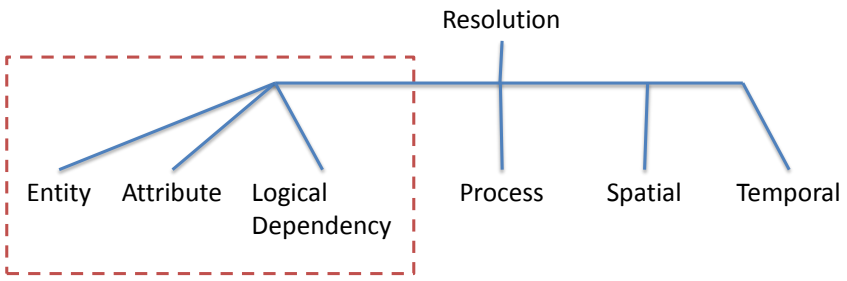

Figure 1.1: Aspects of model resolution (adapted from [22])

coordination between the model designers and developers during development cycle, which, in general, is not practical, since models might be developed at different sites and at varying times, using a wide range of technologies, standards and tools.

The entity, attribute and logical dependency facets of resolution refer to the externally manifested properties of models such as input/output variables and their data types. We claim that the resolution of a model can more conveniently be manifested by its visible properties. As such, we can document a model through its input/output variables and use this information to facilitate the composition with other models of different resolution levels; however, to ensure a consistent data exchange between composed models a mapping of these variables is needed. A purely syntactic approach to define the mappings between these variables may not result in a successful model composition, since the semantics of data types should be preserved through resolution conversions. To cater for such semantics-preserving conversions we propose to map the inputs and outputs of models via special connectors, called *converters*.

1.2 Our Contribution

Clearly, ad hoc forms of converter building processes would not contribute much to the current state of CRM since many simulation application developers inevitably implement some form of conversion logic to compensate for the resolution mismatch between models. We argue that a rigorous resolution mapping methodology should be founded on a formal basis to address the four inter-related issues of (i) entity definition, (ii) mapping specification, (iii) verification and (iv) model composition.

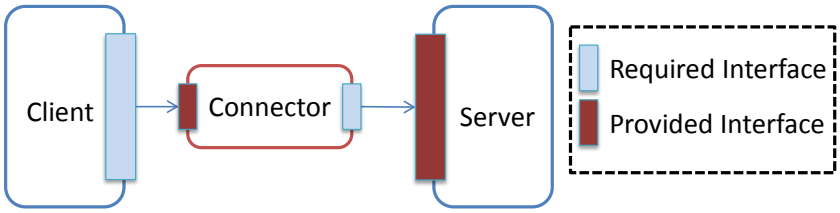

Figure 1.2: A sample application using connectors (adapted from [13])

Further, in order to be practical, this methodology should have tool support.

Our contributions to the work in the field of complex simulation software construction involving CRM are closely aligned with the four aspects listed above. We provide a methodology for resolution mapping in a CRM setting where we use Event-B [3] as a formal specification language, fulfilling i) given above; we use *refinement relations* between Event-B machines for the validation and generation of the data conversion steps between models, fulfilling ii), we use tools to verify the transformations and provide a code generator that uses the Event-B definitions and refinement relations to generate the converter code, with the option of generating monitor code based on machine invariants in support of runtime verification, fulfilling iii); we also propose a formal definition of a resolution converter in the DEVS setting, to incorporate it into a well-established model composition paradigm, and fulfilling iv).

1.3 Connectors for Heterogeneous DEVS Simulations

A resolution converter in the DEVS setting is in fact implementing a connector approach in a simulation environment. Component reuse [59] and their adaptation for multiple reuse contexts via first-class constructs called connectors [13] (Figure 1.2) are well explored concepts in component based software engineering (CBSE) literature. Furthermore, formal approaches such as Reo [8] are developed to support the construction of complex connectors through the composition of simpler ones.

The Discrete Event System Specification (DEVS), on the other hand, is a formalism introduced by Zeigler (1976) to describe discrete event systems with component-based model composition and reuse. Simulation models are constructed with compo-

sition of two kinds of models; atomic and coupled. An atomic model is the smallest entity that contains the model behavior, and coupled models are containers that are responsible from the composition of atomic and coupled models.

DEVS allows reuse of atomic or coupled models in a relatively homogeneous environment, where models produce and consume data with compatible data types and logically operate on compatible time resolutions. However DEVS, in its basic form, does not address the problems emanating from data type and time resolution incompatibilities. The term "time resolution incompatibility" here refers to cases where state computation (or update) intervals of interacting models do not guarantee an overall consistent behavior for the whole simulation.

Our proposal is that the connector paradigm in CBSE, and in particular complex connector composition paradigm supported by Reo [8], can be used for composition of heterogeneous DEVS models to bridge the model incompatibilities and solve some of the interoperability problems. We propose to develop connectors from DEVS atomic models, compose those connectors (as in Reo channels) to get more complex connectors in the form of DEVS coupled models if necessary, and use those to compensate for data syntax and time resolution incompatibilities in a DEVS setting. Thus, we seek to develop a systematic approach to model adaptation and reuse for complex simulation applications, by combining the power of two formal approaches (i.e. DEVS and Reo).

The remainder of this thesis is organized as follows. Chapter 2 outlines the background to our research; Chapter 3 details our proposed approach and Chapter 4 provides a description of our software tools to demonstrate the results of our work. Then in Chapter 5 present our discussions regarding the important aspects of this research. In Chapter 6 we present our approach on using connectors for heterogeneous DEVS simulations. Finally, Chapter 7 contains our concluding remarks and future work propositions.

CHAPTER 2

BACKGROUND

This chapter provides information that is essential for a better understanding of our work. First we provide the definitions of multi-resolution and cross-resolution modelling, including the concept of aggregation/disaggregation which is particularly important for entity resolution mapping. Then we present introductory information on Event-B and refinement which together constitute the formal basis of this research. Consecutively, we provide basic material on the DEVS formalism which constitutes the basis for our world view on model composability via port connections. Finally, we give a concise summary of SiMA, our simulation framework that implements an extended form of DEVS and serves as a basic software platform for the implementation of our CRM work. Further we provide brief information about component based software engineering and connector model, then introduce Reo coordination model for component composition, which we use for our connector solution approach.

2.1 Multi-Resolution Modeling (MRM)

All simulation models are abstractions of a reality but some are more abstract, in the sense that they are less detailed than others representing the same reality. Resolution is the level of detail at which system components and their behaviors are depicted [23]. The subject that deals with multiple levels of resolution for simulation models is called Multi-Resolution Modelling(MRM).

A comprehensive definition of multi-resolution modeling is given by Davis in [23] but the basic points are that it comprises:

Table2.1: Illustrative Range of Nonintegrated DoD Models with Varied Scope and Resolution [51]

Level of Model	Scope	Level of Detail	Time Span	Outputs	Illustrative Uses	Examples
Theater/ campaign	Joint and combined	Highly aggregated	Days to weeks	Campaign dynamics	Evaluation of force structures, strategies, balance; war-gaming	CEM, TACWAR, Thunder, JICM
Mission/ Battle	Multi-platform	Moderate aggregation, with some entities	Minutes to hours	Mission effectiveness	Evaluation of alternative force employment concepts, forces, systems; war-gaming	Eagle, Vector II Suppressor, EADSIM, NSS
Engagement	One to a few friendly entities	Individual entities, some detailed sub-systems	Seconds to minutes	System effectiveness	Evaluation of alternative tactics and systems; training	Janus, Brawler, ESAMS
Engineering	Single weapon systems and components	Detailed down to piece parts, plus physics	Sub-seconds to seconds	Measures of system performance	Design and evaluation of systems and subsystems; test support	Many, throughout R&D centers

(i) building a single model with alternative user modes involving different levels of resolution for the same phenomena;

(ii) building an integrated family of two or more mutually consistent models of the same phenomena at different levels of resolution; or

(iii) both i) and ii).

For example, the sensor model in a wireless sensor network simulation [64] may be implemented at different levels of resolution. To analyze a routing protocol, a sensor model with a simple battery and wireless model is sufficient; however to analyze the monitoring capability of a sensor, details such as the sensing unit and its sensitivity to environmental conditions should be covered by the sensor model.

Table 2.1 shows that there are multiple resolution and scopes for simulation models

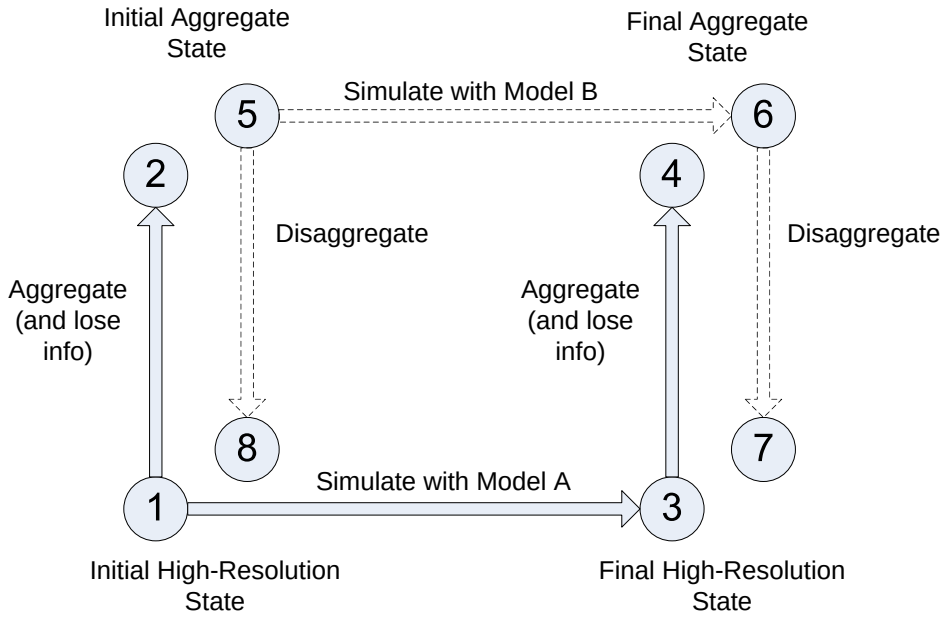

Figure 2.1: Usual Depiction of Weak and Strong Model Consistency in MRM [23]

and generating an integrated simulation of multiple models is not an easy problem.

Davis in [23] focuses on the consistency of MRM models. He pays attention on the term "Consistent in the Aggregate", which means execution of simulations in different resolutions should produce consistent results. As shown in Figure 2.1 results of an aggregated (low resolution) model should be consistent with the results of the disaggregated (high resolution) model after a conversion with an disaggregation operation and vice versa. Consistency of results may include equality or the state of "close enough", which is decided with a projection function that maps each result into some subset and measures the difference.

2.1.1 Why is MRM Important?

Both low and high resolution models are used in modeling and simulation environment but majority tend to place greater trust in high resolution. Many even say that as computing power grows, the need for low-resolution models will diminish, but the need for multiple levels of resolution, from high to low, will be as important as today and a hundred years from now. Here are some of the reasons [23]:

- *Cognitive Needs*: Reasoning occurs at different levels, and each level of reasoning has its own natural variables and concepts of cause and effect. For example in a war simulator, one may decide to reduce the attack power by %20. In a low-resolution model, probably there will be a single input variable, but in a high-resolution one there is not a sub-set of variables that can easily reduce the attack power.

- *Economy*: It is sometimes necessary to use low resolution models, because high resolution comes with a cost. Models become more complex, making them harder to program, debug and validate.

- *Explanatory Power*: Low-resolution models can provide transparent and persuasive explanations of the results, in contrast high-resolution models are often so detailed as to be opaque.

- *Uncertainty, Ignorance and Chaos*: Incorporating more detail in a model may merely spread the analyst's uncertainty across multiple factors.

2.1.2 Modeling Environment for MRM

McEver and coworkers [45] focuses on the modeling environment of an MRM simulation and lists the attributes that are needed if workers are to succeed rather than throw their hands up in frustration:

- *Visual modeling*: Modeling environments should allow users to see how processes and objects interact with each other.

- *Interactive languages*: The ability to easily experiment with models and their configuration parameters without having to recode and recompile is important.

- *Facilities for experimental design and configuration control*: Construction of skeletal hierarchies and a thoughtful design phase before coding begins greatly improve the quality and usefulness.

- *Statistical analysis tools*: These tools are required on the analysis of results to assess approximations and estimate data and process validity.

- *Graphical visualization tools*: To analyze the model output data these tools generate insight into the important process in the scenario space.

SiMA [41], which is a DEVS [65] based modeling framework, has many of these tools implemented and we use it to demonstrate our proposal in action.

2.1.3 Cross-Resolution Modeling

Cross-Resolution Modeling (CRM) [55] is applicable to the concept of simulations at different levels of resolution that are required to inter-operate. For CRM, it is important to understand the assumptions made about the levels of resolution of the simulation models. Two models that are required to work together might have different characteristics that would make inter-operation difficult. Ensuring that such discrepancies are resolved to allow simulations to interact with each other meaningfully is the heart of cross-resolution modelling.

To understand the CRM concept, a definition of resolution is needed:

> As can be seen in Figure 2.2, resolution is a multifaceted concept. Using a military example to make the distinctions, higher entity resolution might mean following units as small as battalions rather than divisions; a higher attribute resolution might mean following the number of various weapons held by each battalion rather than merely assigning the battalion a net "strength;" higher logical-dependency resolution might mean including constraints on the attributes and their interrelationships (e.g., the sum of the men in the units comprising a division should equal the number of men in the division). Higher process resolution might mean computing combat attrition at the battalion level, rather than computing it at the division level and then spreading the attrition equally across the division's battalions. Higher spatial and temporal resolution means using finer scales for space and time [22].

As seen in this definition of resolution, unless two models have been designed with CRM in mind, we cannot easily discuss the relative resolution between them, because

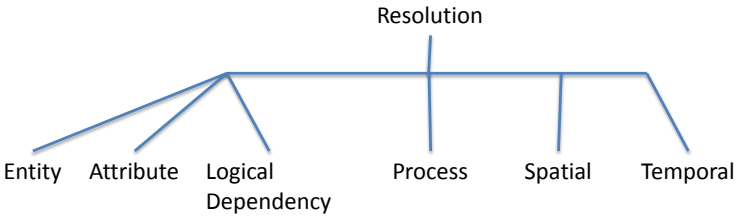

Figure 2.2: Aspects of Model Resolution [22]

they are likely to be complex and confusing since the resolution of one model compared with another can be higher in some respects, lower in others and the same in others.

2.1.4 Aggregation-Disaggregation

The common approach to CRM is aggregation/disaggregation and these twin processes ensure that entities interact with each other at the same level by forcing one entity to be formed at the level of the other. For example, in a wireless sensor network simulation [64] sensors and interactions between sensors can be modeled in terms of single units or bundles. A bundle is an aggregation that models a set of sensors with a single base unit. If a low-resolution entity (LRE) and a high resolution entity (HRE) have to interact, either the LRE will be decomposed into its constituents in a process known as disaggregation (LRE $\rightarrow$ HRE, bundles to units) or an aggregation process takes place (HRE $\rightarrow$ LRE, units to bundles). Aggregation-disaggregation have some issues to be discussed [55]:

- *Temporal Inconsistency* exists when two entities have conflicting or inconsistent representations of the state of a third entity at overlapping simulation times.
- *Mapping Inconsistency* occurs when an entity undergoes a sequence of transitions across levels of resolution, result in a state that could not be achieved by simulation.
- *Chain Disaggregation* occurs when an LRE is disaggregated to interact with an HRE, all other LREs that interact with that LRE would also be disaggregated.

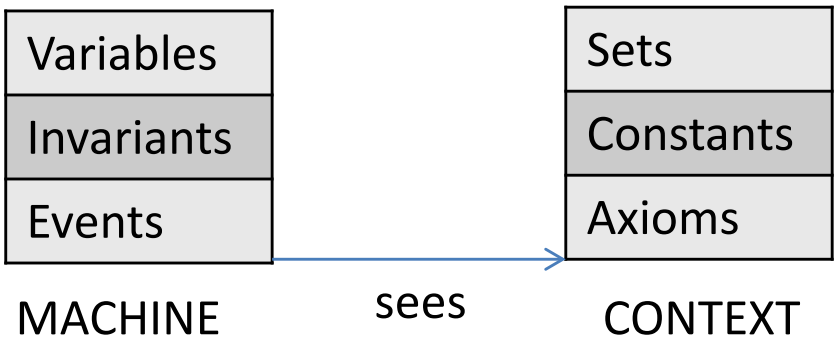

Figure 2.3: Machine and Context (adapted from [4])

- *Transition Latency* is the time to compute aggregated/disaggregated entities.
- *Thrashing* is the flip-flop of resolution levels for an entity that have dynamically changed interactions with other entities of different resolutions.
- *Network Flooding* is the overhead on network traffic if entities are distributed over a network, as each disaggregation operation adds new attributes and aggregation-disaggregation process require some control messages.

2.2 Event-B

Event-B [3] is a formal modeling method for discrete systems based on refinement [4], [29]. The main purpose of creating models in Event-B is to consider and understand the complete system starting from an abstract description.

When modeling a system, Event-B creates the formal model, such that constant and variable parts are kept in the distinct components of *contexts* and *machines* respectively. A machine consists of three distinct elements: (i) a set of state variables, (ii) a conjoined list of predicates, the invariants, and (iii) some transitions, called events. A context consists of objects (sets and constants) and the axioms that constrain these objects (Figure 2.3).

Events are operations that update the state variables of a machine. Each event is composed of guard and action statements. An event is allowed to execute an operation whenever all its guard statements return true. Action statements define the behavior

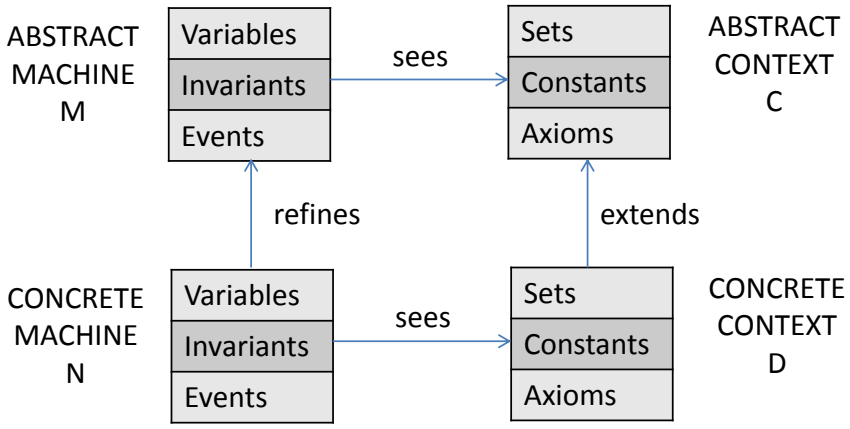

Figure 2.4: Machine Refinement and Context Extension (adapted from [4])

of the event operation and are required to update the state variables of the machine.

2.2.1 Refinement

Refinement [4] is applied to Event-B when there is an introduction of new machines and contexts that are related (refines, extends) to existing abstract ones (Figure 2.4). The sets and constants of an abstract context are kept in its extension. In other words, the extension of a context only consists of new sets and constants. However, in the refinement of the machines, the concrete machine N has a collection of state variables which might be completely distinct from its abstraction M. However, it is allowed that invariants of N can depend on variables of its abstraction M, which are called *glue invariants*. They “glue" the state of the concrete machine N to that of its abstraction M. Glue invariants are important from our point of view since we use them as the main constructs for the specification of the required data transformations between different entity resolution levels.

The new machine N has a number of events that have a corresponding event in the abstract machine M. New events, which do not exist in M, can also be introduced; they can be viewed as refining an implicit event which does nothing (skip).

2.2.2 Proof Obligations

To reason about a machine we consider its proof obligations, which are produced from the union of invariants, axioms, and guards. Proof obligations have two-fold purpose [29]:

- They show that a model is sound with respect to some behavioral semantics.
- They serve to verify the properties of the model.

There are several types of proof obligations, some of which are:

- *feasibility*: The body of an event should not be blocked when the event is enabled, for example, the before/after predicates should not prevent from continuing.
- *invariant preservation*: Action statements of events should preserve all invariants of the machine.

A machine in Event-B must be verified by discharging its proof obligations. Software tools are developed for both definition and verification of Event-B machines:

- Atelier B [18] is a commercial product, designed primarily for B Language [2], but it has an extension for Event-B.
- Rodin [5] is an open-source project and it is actively being developed by its community. It is developed over Eclipse framework [60] and supports plug-in development for both functionality and language extension. Due to its power in application programming interface (API) and since it has an open-source license we selected Rodin as our Event-B tool.

2.3 DEVS

The Discrete Event System Specification (DEVS) is a formalism introduced by Bernard Zeigler in 1976 [66] to describe discrete event systems. In this formalism there are

two types of models; atomic models and coupled models. The former have a behavioral logic and the latter consist of other models and connections between those models. An atomic model in the classical DEVS consists of a set of input events, a state set, a set of output events, an internal and an external transition function, an output function, and time advance function. The formal definition of an atomic model in the classical DEVS formalism is as follows:

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

where,
X is the set of input events,
S is the set of states,
Y is the set of output events,
$\delta_{int} : S \rightarrow S$ is the internal transition function,
$\delta_{ext} : Q \times X \rightarrow S$ is the external transition function such that:

$Q = \{(s, e) \,|\, s \in S, 0 \leq e \leq ta(s)\}$ is total state set,

$\lambda : S \rightarrow Y$ is the output function,
$ta : S \rightarrow R^{+}_{0,\infty}$is the time advance function.

In DEVS models communicate with each other using their ports, which are interfaces of models. External input events (X) are received by the input ports and output events (Y) are sent from the output ports. The state set S is valid for a time interval, which is determined by the time advance (ta) function. After the completion of each time interval the output function (λ) is executed to send the output events that belong to a current state and then the internal transition function(δ_{int}) is executed to calculate the new state. If a model receives an external event during this time interval(e) the external transition function(δ_{ext}) is executed and the current state is updated to reflect the effects of the incoming events.

Coupled models are a composition of components (atomic or coupled models) and the port couplings between these components. Coupled models do not contain any behavioral logic, states, or transition functions to be executed. They are intermediate structures that form the hierarchy in model structure. A coupled model in the classical DEVS formalism is formally defined as follows:

$$CM = \langle X, Y, D, \{M_i\}, EIC, EOC, IC, SELECT \rangle$$

where,
X is the set of input events,
Y is the set of output events,
D is the name set of sub-components,
$\{M_i\}$ is the set of sub-components where for each $i \in D$, M_i can be either an atomic DEVS model or a coupled DEVS model
EIC : external input coupling connects external inputs to the sub-component inputs,
EOC : external output coupling connects sub-component outputs to the external outputs,
IC : internal coupling connects sub-component outputs to the sub-component inputs,
$SELECT : 2^D \rightarrow D$ is the tie-breaking function which defines how to select the event from a set of simultaneous events.

A complete description of DEVS semantics can be found in [66], [65], and [10].

2.4 SiMA

SiMA (Simulation Modeling Architecture) [41] is a modeling and simulation framework, which is based on the DEVS [66] approach to provide a solid formal basis for complex model construction. SiMA Simulation Execution Engine implements the Parallel DEVS [65] protocol which provides a well-defined mechanism for model execution. SiMA builds on a specialized and extended form of DEVS formalism, namely SiMA-DEVS, which:

1. Formalizes the notion of "port types" leading to a strongly typed (and therefore type-safe) model composition environment. In this respect SiMA specializes the basic DEVS formalism by introducing type constraints on the port definitions.

2. Introduces a new transition function, called *Direct Feed Through Transition*, to account for model interactions involving state inquiries with possible algebraic transformations (but no state change) without simulation time advance. In this

respect we extend the basic DEVS formalism. This is similar to the notion of zero-lookahead in HLA [28] from a time-management point of view.

Strongly-typed data ports require a model developer using SiMA to define data types to be used for inter-port communication. SiMA uses port data types for several issues including serialization/deserialization of data values flowing between atomic models.

SiMA-DEVS formalism is given below:

$$SiMA - DEVS = \left\langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta, \delta_{df} \right\rangle$$

Where,
X: Set of input values arriving from set of input ports, P_{in},
Y: Set of output values sent from set of output ports, P_{out},
P_{in}, P_{out}: Set of input and output ports such that:

$P_{in} : \{(\tau, I_x) | \Gamma \mapsto \tau \wedge I_x \subseteq X \wedge \forall x \in I_x, \tau \mapsto x\}$,
$P_{out} : \left\{\left(\rho, O_y\right) | \Gamma \mapsto \rho \wedge O_y \subseteq Y \wedge \forall y \in O_y, \rho \mapsto y\right\}$,
Γ: XMLSchema [63] type system,
τ, ρ: data types valid wrt XMLSchema type system,

$\delta_{df} : PDFT_{in} \in P_{in} \times S \rightarrow P'_{out} \subseteq P_{out}$ is the direct feed through transition function.

Note that the set of input ports P_{in}, is formally defined as a set of pairs where each pair defines one input port of a model uniquely. The first element of each pair, τ, is a data type conforming to XMLSchema [63] type system (denoted with Γ) and the second element of the pair (I_x) denotes the set of input data values flowing through that port, where each element of the value set conforms to data type τ. Similar semantics also apply to output ports. Thus, SiMA makes *strong typing* and *type-system dependency* of the ports explicit in the formal model. Although introducing a run-time oriented property into the formal model may seem unusual, SiMA argues that there are a number of merits in doing so:

1. Introduction of a type discipline to the definition of the externally visible model interfaces (i.e. ports) leads to an information model for the overall system being modeled (coherency in modeling level information space), as well as for

the simulation environment (consistency and robustness in run-time-level data space).

2. SiMA facilitates Model-Driven Engineering through well-typed and type system dependent external plugs to enable automated port matching and model composition. In fact SiMA successfully implements a Model-Driven simulation construction pipeline, via a number of tools such as a code generator, a model builder and a model linker.

3. SiMA reduces the gap between modeling level logical composability constraints and run-time level pluggability constraints, thus forcing all implementations of specialized DEVS model to respect the type-system compatibility and to offer a strongly typed environment.

Note also that, in addition, SiMA introduces a new transition function, δ_{df}, that enables models to access the state of other models through a specific *type* of port, without advancing the simulation time. As such, it is possible to establish a path of connected models along which models can share parts of their state, use state variables to compute derived values instantly within the same simulation time step. As stated earlier, this is similar to the notion of zero-lookahead found in HLA [28]. One may argue that the zero-lookahead behavior could be modeled by adjusting the time advance function of an atomic model such that the model causes the simulation to stop for a while, do any state inquiry via existing couplings, then re-adjusting the time advance to go back to normal simulation cycle. Although this is possible, we argue that by introducing a transition function and a specific port type which is tied (through run time constraints imposed by the framework) to that particular transition function we gain several advantages:

1. The models can communicate and share state with each other without the intervention of the simulation engine thus providing a very efficient run time infrastructure.

2. Allowing such communications only to occur through a specific port type (compile time and run-time checks are carried out) the framework is able to apply

application independent loop-breaking logic at the ports to prevent algebraic loops, thereby ensuring model legitimacy.

SiMA is developed for a commercial project, but it has roots on academic community. It is continuously developed and improved for the requirements of new projects and scientific advancements. Some of the research that is already published is:

- Deniz et al in [26] added the functionality of dynamic DEVS formalism, which allows changing model structure while the simulation continues.
- Bozagac et al in [11] used SiMA in their simulation distribution architecture, which executes simulations of batch runs and monte-carlo trials on different computers and collects their results on a single computer.

2.5 HLA

The High Level Architecture (HLA) [20] is a set of specifications and rules within a common architecture supporting reuse and interoperation of simulations. It was developed for the need of interoperability among new and existing simulations within the U.S. Department of Defense.

In HLA each simulation is called a federate, and the composed simulation with different federates is called a federation. All federates are connected to a centralized server which is called RTI, and all communication and time management is done by the RTI.

The baseline definition of the HLA includes:

- *HLA Rules*: Summarizes the key principles behind the HLA. Like: federations should have a FOM (common data exchange document), federates should have documentation for HLA, etc. [36]
- *HLA Interface Specification*: The runtime services provided to federates by the RTI, and by federates to the RTI. Like: message sending/receiving, time management etc. [35]

- *HLA Object Model Template (OMT)*: The documentation templates that describe the set of objects, attributes and interactions used in an HLA simulation. There are two templates of OMT [37]:
  - Federation Object Model (FOM): Defines the set of objects, attributes and interactions shared across a federation.
  - Simulation Object Model (SOM): Same as FOM, but specifies only a single federate to define any requirements for future federations.

In HLA, the "Federation Object Model (FOM)" [37] allows models to define their syntactic requirements for composition into a federation. It includes the set of objects and attributes, their data type and quantity with a basic unformatted description. It allows modelers to understand and define the data exchange protocol inside a federation.

2.6 Composability

There are so many simulation models from different application domains, levels of resolution and time scales on the market. Model Composability is the ability to compose these models and generate a single simulation.

There are different factors that affect the variety of models, such as cost, ease of use and confidence in the value the simulation can provide. As each factor has its own solution, a single system that addresses all these issues is not possible, so developing a simulation infrastructure that simultaneously address as many of these issues as possible is important. The conceptual system from such an approach is that of a composable simulation framework.

With composability, simulation developers would no longer have to build large, inclusive, monolithic simulations. Instead they would build small modules with well defined functionality that are readily combined with other modules. And simulation users don't need any knowledge of specific module content, or how modules are selected, combined or run. [42, 56]

A composable simulation has six major steps [42]:

1. *Identify user requirements*: A system should assist the user in defining his requirements and constraints.
2. *Translation of user requirements and module identification*: Users specify the operational requirements, which must be mapped to functional descriptions.
3. *Library of simulation models*: A library of simulation "building blocks" is required. This library needs to be extensive enough to address the desired level of detail and needs of all application domains.
4. *Development of candidate simulations*: For a single simulation, system should define all possible combinations to allow selection of the best.
5. *Selection of best simulation candidate*: System should develop estimates of simulation performance and compare them to determine the one most appropriate for the simulation.
6. *System evaluations*: To meet the need of users' confidence, the system should have a series of evaluations with some sort of Figure and Merit.

As illustrated in Figure 2.5, the composable simulation problem have both horizontal and vertical dimensions [52]:

- In horizontal dimension, coupling components facilitate their interoperation and composability facilitates multi-resolution modeling.
- The vertical dimension involves the act of coupling two models for the sake of aggregation and hinder the application of other, more suitable, methods of abstraction.

Composability has many facets that need to be addressed; distinguishing among them, and their associated challenges is essential [25]:

- *Syntax*: Models should be able to operate together on the technical level, i.e., the digital output from one can be read as the digital input to the other.
- *Semantics*: The data should have the same meaning in the sending and the receiving model.

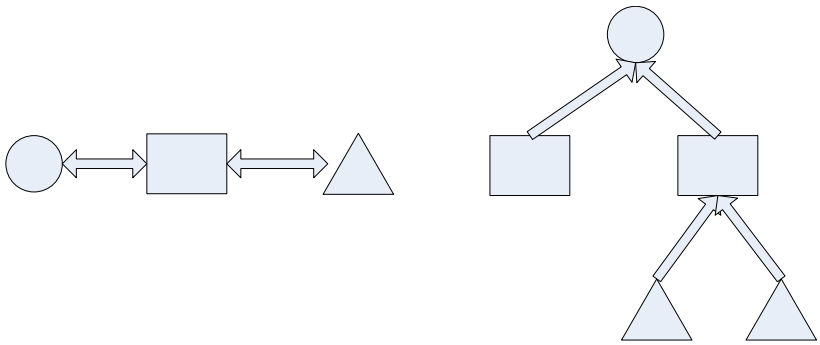

Figure 2.5: Horizontal and Vertical Dimensions of Simulation Composability

- *Pragmatics*: Meaning is not always straightforward, because a word means different things on different contexts. Pragmatics puts data into context of how they are used in the model.
- *Assumptions*: The way Model A calculates the data may not be suitable for what Model B needs. For example a temperature datum, might refer to a surface temperature, or an average temperature over some path etc.
- *Validity*: The question of whether a model is correct, a model that is adequately valid in one context may not be valid in another.

2.7 Components and Connectors

Component based programming [59] aims at building large scale systems by composing them from smaller parts. These parts (components) are well defined, they express explicitly what functionality they offer (provided interface) and what functionality they depend on(required interface). Components communicate with their environment only through these two sets of interfaces. This encapsulation reduces application complexity, eases the code reuse and allows better manageability and extensibility [13].

For heterogeneous environments, where components are developed with different requirements and executed in a single system, the required/provided interface paradigm is not sufficient for component composition. Because component interfaces generally do not have a common environment view that precludes the communication.

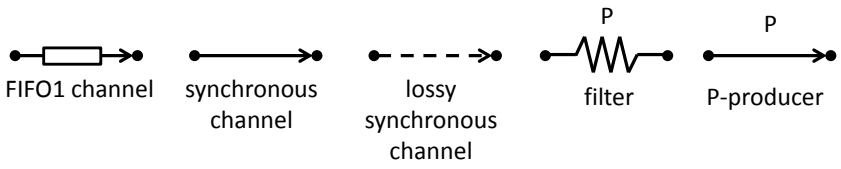

Figure 2.6: Some basic channels in Reo (reproduced from [46])

Connectors in component systems are the communication channels between components. They model and execute the communication between components and propose solutions for interface mismatch. A connector is often selected in design stage where it realizes the binding by representing an interaction of a set of components (Figure 1.2).

2.8 Reo

Reo [8] is a channel-based coordination model for component composition where complex coordinators, called connectors, are compositionally built out of simpler ones. In Reo, connectors are organized in a network of primitive connectors, called *channels*.

Each channel has two channel ends and there are two types of channel ends, namely, source and sink. A source channel end accepts data into the channel and a sink channel end dispenses data out of the channel. It is possible for the ends of a channel to be both sinks and both sources. Thus an open-ended set of channel types can be used together (Figure 2.6).

Complex connectors are constructed by composing channels via join operations. Channels are joined together in nodes where communication between channels are coordinated by letting sink channel ends provide data to source channel ends.

A complex connector has a graphical representation, called a Reo circuit, which is a finite graph where the nodes represent join operation on channel ends and the edges represent the connecting channels (Figure 2.7). The behavior of a Reo circuit is formalized by means of data-flow at its sink and source nodes.

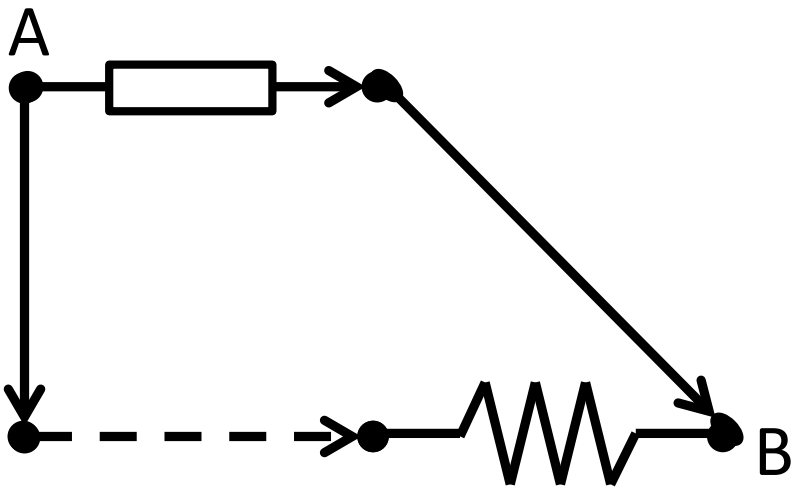

Figure 2.7: A sample reo circuit

CHAPTER 3

A METHODOLOGY FOR ENTITY RESOLUTION MAPPING

Our approach utilizes the data type information from the input/output variables used in models of different resolution levels to facilitate their composition. If two distinct models are to be composed via their input and output ports, they must either use identical input/output data types or a conversion is needed between the data types. Evidently, our assumption here is that composable models are *semantically related* but may have non-identical data representations due to the different entity resolution levels they employ.

To compose models at different resolution levels, our solution proposes the specifying, generation and use of converters between data types defining the ports of connected models. Put simply, these converters are connectors between output and input ports of models. Similar to models, converters have input and output variables, but unlike typical atomic models they do not have any behavioral logic. Contrary to the simplicity of the idea, achieving resolution mapping via converters following a systematic methodology is not straightforward. Thus, we state that systematic and repeatable resolution mapping via converters should have the following four aspects:

1. A formal language for specifying entities to be mapped,
2. An unambiguous, machine processable notation for specifying the set of transformations required to get a set of entities from another set of (input) entities,
3. A tool that assists with verification of the conversion specification and generates executable converters for a conversion,
4. A well-defined, formal basis for incorporating the notion of resolution convert-

ers into model composition schemes in a uniform way.

Although there are several proposals [55], [33], [34], [9] regarding the use of converters between simulation models in the literature, to our best knowledge they fall short of offering a comprehensive solution with respect to the aspects listed above. The main novelty of our approach is as follows:

1. It involves a formal proposal to fit the concept of converters into a well-established model composition paradigm (i.e. DEVS). Our proposal states that resolution conversion (of entities) can be specified uniformly via first class constructs called "connectors" that are inserted between couplings among atomic and/or coupled models in a DEVS setting.

2. It provides a systematic methodology that offers a well-defined sequence of stages to obtain executable converters for entity resolution mapping, given the appropriate descriptions of entities and refinement relations. Our methodology relies on Event-B as a formal specification language that utilizes *refinement relations* between Event-B machines for validation and generation of the data conversion steps between models, and employs a code generator that uses Event-B machine definitions and refinement relations to generate the converter code.

3. It enables the systematic re-use of converters in a uniform way. We define converters as a first class constructs within a model composition paradigm.

Section 3.1 provides a more detailed account of the first point, Section 3.2 gives details of the second point. Although the third point is a direct consequence of the first two, a detailed discussion is outside the scope of this thesis.

3.1 The Formal Representation of Converters in a DEVS setting

The formal representation of converters in DEVS terminology is important in order to forge the notion of converters into a coherent and well established model composition paradigm. In fact our SiMA-DEVS definition introduced in [7, 26] and reproduced below, provides a convenient formal basis for a converter definition.

M=$\langle X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \delta_{df}, \lambda, ta \rangle$
where
$S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta$ are as defined in the parallel DEVS formalism [65],
Γ: XMLSchema type system,
$InPorts = \{(\Gamma, \tau) \mid \Gamma \vdash v : \tau, v \in X\}$,
$OutPorts = \{(\Gamma, \rho) \mid \Gamma \vdash v : \rho, v \in Y\}$,
$X = \{(p, v) | p \in InPorts, v \in X_p\}$ is the set of input ports and values,
$Y = \{(p, v) | p \in OutPorts, v \in Y_p\}$ is the set of output ports and values,
X_p is the set of input values that can arrive at port, p
Y_p is the set of output values that can be published from port, p
$InPorts, OutPorts$: Sets of input and output ports such that:
τ, ρ: data types valid w.r.t. XMLSchema type system,
$\delta_{df} : PDFT_{in} \times S \rightarrow OutPorts'$ is the time invariant direct feed through function; where $PDFT_{in} \subseteq InPorts$ and $OutPorts' \subseteq OutPorts$.

Here, we propose that a *connector model* can be defined in two different forms:

(i) A *standard DEVS atomic model* can be restricted in its time advance function:

$ta : S \rightarrow \{0, \infty\}$ is the restricted time advance function.

This implies that the atomic model operates with only two states: 1. An idle state where the model waits for an input (i.e. next time interval is equal to infinity), 2. A transition state with zero duration which is triggered by the receipt of an input that produces the converted output at the end of the state.

(ii) A *SiMA-DEVS atomic model* can be restricted in its ports:

$PDFT_{in} = InPorts$ and $OutPorts' = OutPorts$.

This implies that the atomic model only has defined DFT ports, meaning that only δ_{df} would be operational at run time (i.e. other transition functions would do nothing). As mentioned in Section 2.4 the DFT ports and δ_{df} implement the zero lookahead and provide a convenient mechanism for simple processing within the same simulation time step.

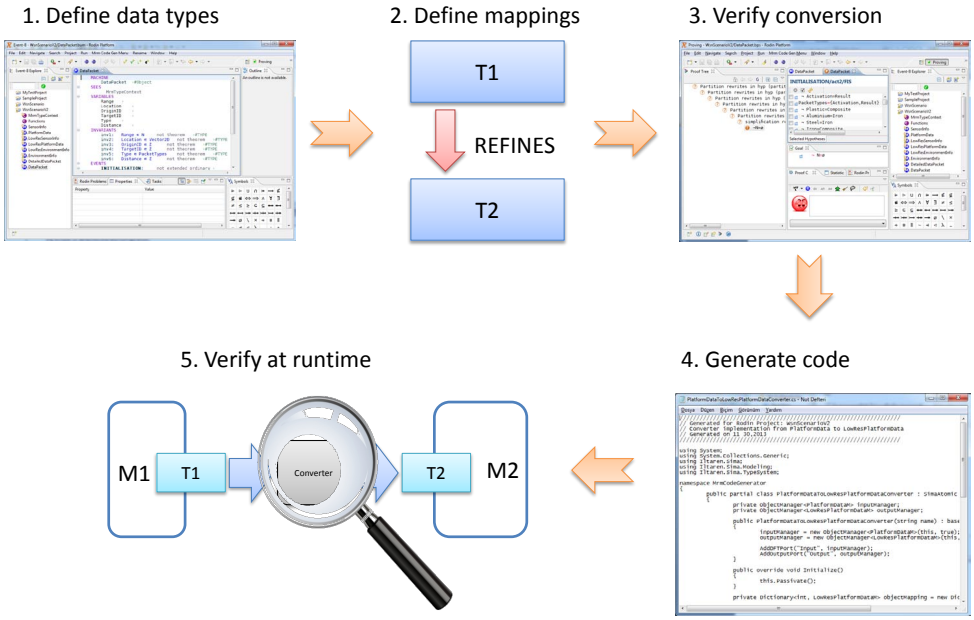

Figure 3.1: Illustration of entity resolution mapping process

Thus, it follows that the connector model does not violate the closure under coupling property of DEVS, since it is essentially an atomic model. In fact, in our case study the connectors are implemented as SiMA atomic models.

3.2 Process of Entity Resolution Mapping

Our methodology has five main stages:

1. Definition of data types for the variables of Event-B machines (Section 3.4.1)
2. Definition of mappings between data types of different resolution levels (Section 3.4.2)
3. Verification of conversion steps (Section 3.4.3)
4. Generation of the converter (Section 3.4.4)
5. Optionally, instrumenting the converters with pre- and post-condition checkers as an aid for runtime verification (Section 3.4.5).

We now present the details of our approach in the following sub-sections. Although our methodology supports both directions of mapping first we will discuss High Resolution Entity (HRE) to Low Resolution Entity (LRE) mapping. Subsequently, we will describe LRE to HRE and mixed mappings.

3.3 Case Study

To demonstrate our approach, we consider a simple wireless sensor network simulation [64] constructed using DEVS compliant models (an extended version of our previous example as presented in [26]). A top-level visual representation of the DEVS models developed to implement the proposed WSN is given in Figure 3.2. The WSN system consists of four components:

- *Sensors* detect the movement of objects in the environment and can communicate with other sensors within their range.
- The *sink* unit is the base unit that collects and fuses all the information supplied by the sensor network.
- The *platform* has a predefined path that it follows during the course of simulation. It represents the detectable object for the sensors in the environment.
- The *environment* model calculates the environmental information depending on the sensor locations, such as humidity and noise level that affect the signal transmission.

The simulation exercise is set to involve thousands of sensors that are spread over a wide area to track the path of a platform. The number of sensors means that the simulation requires a large amount of CPU power and to decrease the CPU cost, the resolution level of some of the sensors that are away from roads can be reduced, since their behavioral requirements allow for lower fidelity levels. A low resolution sensor model that was developed for previous simulations already exists, and will be integrated into the current simulation.

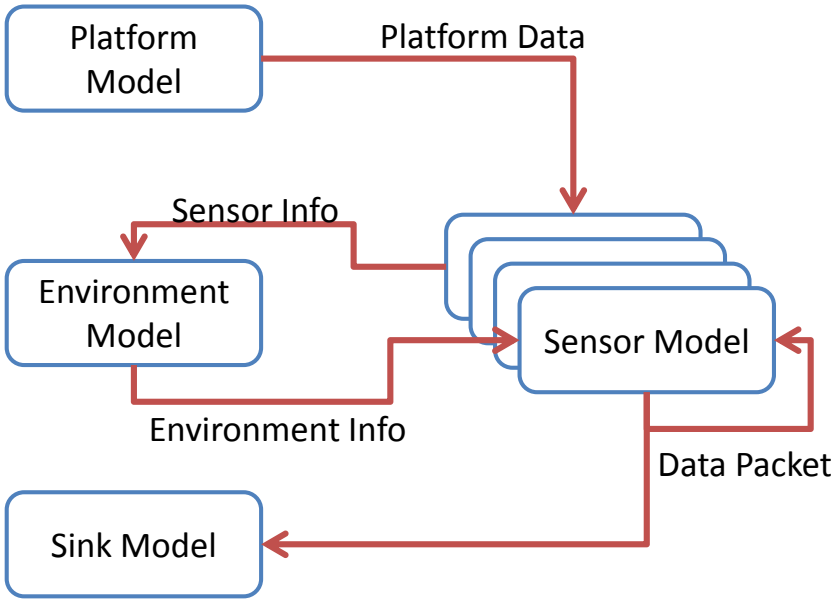

Figure 3.2: Models and entities in the sample scenario

3.4 HRE to LRE Mapping

The sensors transmit *DataPacket* values through their output ports. After introducing the low resolution sensor models, we need to integrate a converter between high resolution sensors and the low resolution sensors (Figure 3.3).

The output data type of high resolution model labeled the *DetailedDataPacket* contains a three-dimensional *Position* attribute with a *SignalStrength* value to be used by the receiver model for receivable signal limit with environmental information from the environment model. The *Direction* value contains a vectoral position of the target. However, the output data type of the low resolution model *DataPacket* contains a two-dimensional *Location* attribute with an exact *CommunicationRange* value which is updated by a percentage with environmental information. Furthermore, it contains only the *Distance* of the target without the direction.

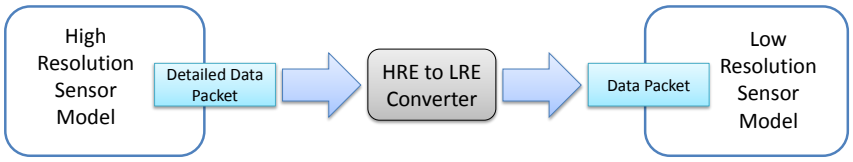

Figure 3.3: Composition of the new low resolution model

3.4.1 Using Event-B for Model Data Type Definitions

We use an Event-B machine as a container for the data type definitions for the input/output variables of a model. As mentioned above an Event-B machine is specified using three main constructs, namely a set of variables, a list of invariants and some events. We define a mapping of these constructs to three facets of the entity type specification in the following way:

- Variables: Each variable represents an attribute of an entity.
- Invariants: Invariants provide semantic information about variables that are crucial for type conversions.
- Events: Events provide modifiers for variables, but our methodology does not require any event definition.

The *DataPacket* entity can be represented with an Event-B machine as given in Figure 3.4. Note that the VARIABLES section of the machine definition includes the attributes of the complex type *DataPacket*, furthermore, the INVARIANTS section contains the types of the attributes. In fact, in an Event-B machine, invariants have two major responsibilities :

- Data Type Specification: such as integer, string or complex types such as array.
- Provision of Constraints: The relations among variables are specified. For instance, range restrictions such as $x > y$ and $x < 1000$, or more complex constraints such as $x \leq y/z * 100$ can be stated.

MACHINE DataPacket

SEES DataPacketContext

VARIABLES

$CommunicationRange$
$Location$
$SensorLocation$
$PacketID$
$OriginID$
$TargetID$
$Type$
$Distance$

INVARIANTS

inv1 : $CommunicationRange \in \mathbb{N}$ (cm)
inv2 : $Location \in Vector2D$ (cm)
inv3 : $SensorLocation \in Vector2D$ (cm)
inv4 : $PacketID \in \mathbb{N}$
inv5 : $OriginID \in \mathbb{N}$
inv6 : $TargetID \in \mathbb{N}$
inv7 : $Type \in PacketTypes$
inv8 : $Distance \in \mathbb{N}$ (cm)
...

END

Figure 3.4: Event-B machine for DataPacket entity

inv9 : $Type = Activation \Rightarrow Distance = 0$

inv10 : $Type = Result \Rightarrow Distance > 0$

inv11 : $Distance < 2000$

inv12 : $CommunicationRange \leq 1000$

inv13 : $OriginID \neq TargetID$

...

Figure 3.5: More invariants for the Event-B machine of DataPacket entity

Figure 3.5 shows the use of invariants for the provision of additional constraints for the *DataPacket* entity. These invariants are used for the dynamic checking of the converter output. Assertion statements are generated corresponding to those invariants as part of the code generation process.

3.4.2 Applying Event-B Refinement

Traditionally, refinement is used to develop a concrete system based on an existing abstract model [4]. For the purpose of resolution mapping, a concrete system corresponds to a high resolution (or disaggregated) entity, and an abstract system corresponds to a low resolution (or aggregated) entity. Thus refinement in this case can be viewed as obtaining a high resolution entity from a low resolution entity. Glue invariants of Event-B map the variables of a refined machine (i.e. the high resolution entity definition) to those of an abstract machine (i.e. low resolution entity definition). This effectively makes the glue invariants the primary vehicle for the specification of refinement relationships between two entities of different resolution levels. It should be the responsibility of the model developers to define the glue invariants for each refinement.

An example illustrating the use of glue invariants for the purpose of defining a refinement from a *DataPacket* entity to a *DetailedDataPacket* entity (as depicted in Figure 3.6) is given in Figure 3.7. Note that the *DetailedDataPacket* machine contains different variables and that it indicates, via the REFINES keyword, that it is a refinement for the *DataPacket* machine. Note also that the glue invariants define how the aggregated variables of the *DataPacket* entity can be obtained from the variables

of *DetailedDataPacket* entity. As a relatively subtle point, notice the use of accessor functions such as Vector3D_X for variables of complex types in the expressions of glue invariants. These functions enable access to the members of complex data types (X member of a Vector3D variable) and are used for both the proof system and code generation described in the following sections.

3.4.3 Proving Glue Invariants

The verification of the glue invariants requires extensive tool support. Fortunately, the Event-B community has developed such tools as Rodin [5] for that purpose. Rodin generates proof obligations (POs) for possible gaps to be filled to construct the proof. Then it tries to automatically prove all POs and if not possible requests the user to prove the remaining POs manually. Inside the proof editor, Rodin requests the user to select related invariants about his/her model and produce the required statements to prove the proof obligation. If the current list of invariants is not sufficient to prove the PO, the user should update his/her glue invariants or define more invariants to provide further constraints. Eventually, with all POs having been proved, the machine is verified and ready for converter generation.

For example, for the glue invariant *glue1* in Figure 3.7 the prover requires the definition of *CalculateRange* function (e.g. SignalStrength * 10) in order to verify the $CommunicationRange \leq 1000$ and $SignalStrength \geq 40$ invariants. The related POs could be resolved by using the function definition inside the Rodin editor.

3.4.4 Converter Generation

Once the glue invariants that define the conversion relationships between low-resolution and high-resolution entities are specified and are proved to preserve the constraints (as specified by the glue invariants), then the converters can be generated based on the statements of the glue invariants. Note that converter generation does not depend on glue invariant proofs. Therefore, converters can be generated while the glue invariants are unproven, but such a practice would not be advisable from the viewpoint of reliability.

MACHINE DetailedDataPacket

REFINES DataPacket

SEES DetailedDataPacketContext

VARIABLES

$SignalStrength$
$Position$
$SensorPosition$
$PacketID$
$OriginID$
$TargetID$
$Type$
$Direction$

INVARIANTS

inv1 : $SignalStrength \in \mathbb{N}$ (dB-microvolts)
inv2 : $Position \in Vector3D$ (cm)
inv3 : $SensorPosition \in Vector3D$ (cm)
inv4 : $PacketID \in \mathbb{N}$
inv5 : $OriginID \in \mathbb{N}$
inv6 : $TargetID \in \mathbb{N}$
inv7 : $Type \in PacketTypes$
inv8 : $Direction \in Vector3D$ (cm)
inv9 : $Type = Activation \Rightarrow ScalarLength3D(Direction) = 0$
inv10 : $Type = Result \Rightarrow ScalarLength3D(Direction) \geq 0$
inv11 : $SignalStrength \geq 40$
...

END

Figure 3.6: Event-B Machine for DetailedDataPacket entity

glue1 : $CommunicationRange = CalculateRange(SignalStrength)$

glue2 : $Location = mk\_Vector2D(Vector3D\_X(Position) \mapsto Vector3D\_Y(Position))$

glue3 : $SensorLocation = mk\_Vector2D(Vector3D\_X(SensorPosition) \mapsto Vector3D\_Y(SensorPosition))$

glue4 : $Distance = ScalarLength3D(Direction)$

Figure 3.7: Glue invariants for DetailedDataPacket entity

Glue invariants in general are of the form of $x = expr(a, b, ...)$, where x represents a variable of a lower resolution entity and $expr(a, b, ...)$ represents some algebraic expression in terms of variables, such as $a, b, ...$ of a higher resolution entity. To generate the converter code, we transform these statements to statements in a programming language and inject them into the related methods that are called during simulation execution.

As such, the generation of a converter from *DetailedDataPacket* entity to *DataPacket* entity, for instance, fills the gap between our high resolution sensor model and low resolution sensor model, and allows the simulation to execute as expected (Figure 3.3).

3.4.5 Monitor Generation

The invariant statements about the provision of constraint information as discussed in Section 3.4.1 can be used to verify the input and output of converters. They specify the relationships between variables and self-restrictions like $x > y$ and $x < 1000$. We generate assert statements for each invariant that produce an error upon receipt of invalid values and the converter input/output is checked against the source and target machine invariants, respectively, at run time, more specifically, before and after the conversion takes place.

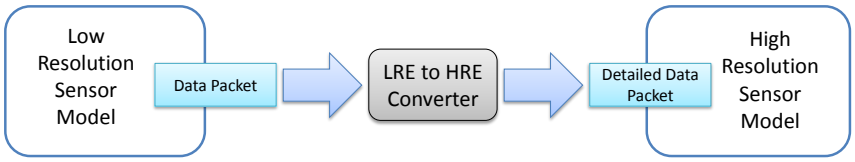

Figure 3.8: Composition of a new low resolution model with LRE to HRE mapping

3.5 LRE to HRE Mapping

Although, up to this point we have discussed the mapping of High Resolution Entities (HREs) to Low Resolution Entities (LREs), our proposed approach puts no limit on the model composer with respect to the direction of the mapping. Event-B refinements can be put to work in both directions with refinement, as usual and abstraction.

The methodology steps discussed thus far can be applied to LRE to HRE mapping without any modification. Our transformation routines introduced above employed only aggregation, therefore additional information was not needed. However, in a cross-resolution modeling scenario, connections from LRE to HRE are likely to require that additional information needs to be provided. This problem requires further consideration.

To illustrate the problem in the context of our example described in Figure 3.2, we require a converter from low resolution sensor models to high resolution models as the sensor network needs communication in both directions (Figure 3.8).

In general, the reverse direction requires extra information (to be inputted to the converter) or assumptions “to fill in the gap". Assumptions to map $CommunicationRange$ value to the actual $SignalStrength$ value can be produced by a model developer who knows the *high resolution sensor model* behavior and can interpret concrete values with respect to the abstract values.

For the sensor model example given above, we need to have some kind of mapping from LREs to HREs. There are many ways to implement such mappings, but we can put them into two main categories:

- Mapping based on assumptions: A conversion function that computes the outputs for inputs based on the assumptions made by the model composer.
- Mapping with the help of external data sources: The conversion function uses external data sources, such as statistical databases, to calculate the output.

We describe these categories in the following sections.

3.5.1 Mapping Based on Assumptions

If the model composer knows the details of the high resolution model behavior and he/she can be certain of the required HRE values for the LRE values; thus, the model composer can implement a function such as $f(i_1, i_2...i_m) = \{o_1, o_2...o_n\}$ while $i_1, i_2...i_m$ are the low resolution inputs and $o_1, o_2...o_n$ are the high resolution outputs.

For the *CommunicationRange* example we may have a mapping similar to the one below:

- $CommunicationRange = 1 \rightarrow SignalStrength = 42$
- $CommunicationRange = 2 \rightarrow SignalStrength = 44$
- ...
- $CommunicationRange = 1000 \rightarrow SignalStrength = 540$

3.5.2 External Data Sources

The *CommunicationRange* variable has a limited domain and can be implemented with simple mappings. More complex variables require external inputs such as statistical databases to fill the gap between the LRE and HRE values [23]. Hence, this reverse mapping of complex variables will require the model composer to attach external data sources to connectors to feed the transformation function.

Let E represent the external data source, then our mapping will look like $f(E, S, i_1, i_2...i_m) = \{S', o_1, o_2...o_n\}$. The state variable S is a reflection of values previously converted and it will let us find more appropriate results from our data source.

axm10 : $CalculateRange \in \mathbb{N} \rightarrow \mathbb{N}$

axm11 : $\forall x \cdot x \in \mathbb{N} \land CalculateRange(x) \leq 10$

Figure 3.9: Sample function declaration and its output constraint

For the sensor model example, the LRE side only transmits a *Distance* value, however the HRE side requires a vectoral *Direction* value which requires the exact position of the target relative to the sensor at the time of detection to compute. The LRE to HRE converter obtains the position of the target from a store that keeps the past values of all the outputs then it fills the *Direction* value with the *Distance* computed from the *Position* taken from that store.

3.5.3 Implementing Mapping on Event-B

Simple mapping functions such as those given in our examples can be easily implemented using Event-B machines however, more complex functions that cannot be expressed using the language capabilities of Event-B would require a different solution, and therefore, we suggest employing user-defined functions for that purpose. Event-B allows for the definition of a function declaration with its inputs and outputs and does not require other function details unless there is a proof obligation that needs to be proved. The constraints on inputs or outputs of such functions can be defined and implemented as shown in Figure 3.9.

Although we cannot formally verify the implementation of these converter functions, we still have the option to validate their input and check their output at runtime with the generation of monitors as discussed in Section 3.4.5.

3.6 Mixed Mapping

As discussed above, the resolution difference between two models might be mixed in the sense that some of the attributes within the same complex entity might be at low resolution and others at high resolution.

To illustrate this situation we can add a new variable to our *WSN* example. The low resolution sensor model uses the $TimeOfDay$ value of *DataPacket* to calculate the actual *Distance*, because sensing unit receives more noise during the day and *Distance* should be reduced to find the target location. However, the developer of a high resolution sensor model does not deal with the $TimeOfDay$ but uses $DayState$ which is an enumeration of *Night* and *Daytime*. Thus, our converter includes a mapping from $TimeOfDay$ to $DayState$ that involves aggregation into our LRE to HRE conversion.

As seen in our examples both HRE to LRE and LRE to HRE mappings can be specified for the generation of the same converter. Thus our solution supports conversion in both directions for the same pair of models. If the model composer defines the required Event-B machine refinements, the converters will be generated accordingly.

3.7 Summary

Summary of the required steps to generate our converters is as follows:

1. Entity types are defined as Event-B machines.
2. Constraints and conversion steps are defined using invariants.
3. Machines (entities) that represent different levels of resolution are linked to each other with refinement relationships.
4. All the machine invariants including glue invariants are proved.
5. Programming language statements of the converter are automatically generated from glue invariants.
6. Monitor statements that check the inputs and outputs of the converter are generated from invariants.

Chapter 4 contains an implementation of our approach using the SiMA [41] framework, followed by a discussion on our findings in Chapter 5.

CHAPTER 4

IMPLEMENTATION IN A DEVS SETTING USING SIMA

As pointed out in Section 2.4, SiMA [41] is a modeling and simulation framework developed by our research group at TÜBİTAK BİLGEM İLTAREN, it is based on DEVS [66] to provide a solid formal basis for building complex models through composition. SiMA provides a convenient software platform for our purposes since it inherently supports simulation construction through model composition. In that, the connectors fit well into the model coupling paradigm via input-output ports. SiMA also comes with a tool-chain that facilitates the employment of a simulation construction methodology which involves a distinct stage where all data types used for input/output variables of model ports are defined in XML conforming to an XML Schema. In a later stage, automated code generation based on these types is achieved by using the KODO tool. As explained in more detail in Section 4.2 this also is a good fit for our converter generation approach employed in this work.

In addition to SiMA we also used Rodin [5] as an appropriate and easily accessible tool for creating and verifying Event-B constructs. Our development efforts contributing to our existing software infrastructure followed two paths:

1. Implementing additional tools for Rodin to cater for the definition of Event-B machines via KODO type definitions and generation of source code from those definitions,

2. Devising a mechanism for the incorporation of converters into model composition, to enable the actual deployment and execution of those converters between SiMA models at simulation run time.

For the first path we have developed two tools that operate as plugins to Rodin:

- *Event-B Machine Generator (EMG)*: A tool to generate Event-B machines on Rodin that represent the KODO data types.
- *Converter Code Generator (CCG)*: A tool to generate converters from the refinement definitions of Rodin.

For the second path we used specific properties of the atomic model implementation provided by SiMA to treat them as state-less algebraic converters.

The overall converter generation process has four stages:

1. Create KODO XML definitions for data types with different levels of resolution (i.e. including abstract and refined counterparts of a certain entity),
2. Use Event-B Machine Generator Plug-in of Rodin (EMG) to produce an Event-B machine definition for KODO data types,
3. Decorate the Event-B machine definitions with glue invariants to specify conversion expressions between abstract and refined entities,
4. Use code generator plug-in of Rodin (CCG) to produce source code that implements the conversion logic corresponding to those Event-B machine definitions.

In the following sections we describe these stages in more detail.

4.1 Creating type definitions

As stated above, for a simulation construction we define our overall information space by creating XML definitions for entities flowing through input-output ports of our models. This creates a natural precursor for potential Event-B machines where resolution conversion is needed. At this point we utilize the KODO tool within the toolchain of SiMA that generates source code for input-output variable definitions for models. KODO has a well-defined schema and a type system to define data types in

```
<object name="DataPacket">
  <member typeName="uint" name="CommunicationRange" type="primitive"/>
  <member typeName="Vector2D" name="Location" type="struct"/>
  <member typeName="Vector2D" name="SensorLocation" type="struct"/>
  <member typeName="uint" name="PacketID" type="primitive"/>
  <member typeName="uint" name="OriginID" type="primitive"/>
  <member typeName="uint" name="TargetID" type="primitive"/>
  <member typeName="PacketTypes" name="Type" type="enum"/>
  <member typeName="uint" name="Distance" type="primitive"/>
</object>
```

Figure 4.1: Sample KODO data type definition

XML. KODO uses XML files that fully specify data type definitions for input/output variables to be used in SiMA models. These type definitions include primitive types and complex types for each data field of objects as illustrated in Figure 4.1.

4.2 Generating Event-B Machines

In this stage, we use the EMG tool developed for Rodin that uses the KODO data type definitions to generate Event-B machine definitions. For example, the EMG reads the KODO type definition given in Figure 4.1 to generate the Event-B machine in Figure 4.2, which is, in fact, equivalent to the machine definition given in Figure 3.4. Note that we generate some metadata in the form of comment tags (such as #Object, #Type etc.) that will allow us to determine object types, invariant categories and other information during the code generation. The generated machine defines all fields of the KODO data type as variables, and their data types as primitive invariants. We also generate and use a *Context* called *MrmTypeContext* to define the type representations of all complex data structures in Event-B.

4.3 Decorating the Event-B Machines with Glue Invariants

The next step is the insertion of refinement relations in the form of glue invariants. The user is required to select the related data types to be used in the converters for

MACHINE DataPacket # Object

SEES MrmTypeContext

VARIABLES

CommunicationRange

Location

OriginID

...

INVARIANTS

inv1 : $CommunicationRange \in \mathbb{N}$ # TYPE

inv2 : $Location \in Vector2D$ # TYPE

inv3 : $OriginID \in \mathbb{N}$ # TYPE

...

END

Figure 4.2: Event-B Machine generated from the KODO data type definition

model composition. Then he/she should define the *REFINES* keyword in the refinement machine (Figure 3.6) and place his/her glue invariants as depicted in Figure 3.7. Note that some variables exist on both LRE and HRE model and do not require a glue invariant.

Additional invariants that define constraint information and required for monitor code generation should be inserted manually. An example of such additional invariants is shown in Figure 3.5.

4.4 Generating the Converter Code

In this stage, the Event-B machines generated by the EMG and finalized by the manual decoration process as described in Section 4.2 are ready for generation of converters. We used the glue invariants in refined machines as the basis of our converters and other invariants for verification.

As SiMA is implemented with C# language [31], our code generator generates con-

verters in C# language. The converter code can then be compiled together with the other source code which implements the overall simulation.

4.4.1 Converter Implementation using SiMA

The central idea is that converters are placed along the coupling connections of input/output ports between models. To achieve this in a systematic and uniform way, we utilize an existing construct of DEVS, namely the atomic model. Although in its original form, an atomic model is used to implement the behavioral logic of a simulation model as a stateful component capable of operating on both discrete-event and discrete-time paradigms. Here, however, we use a specialized and somewhat reduced form of this model which operates as a state-less algebraic converter. To achieve this, we disable all other transition functions of the atomic model and use only the *Direct Feed Through Transition Function* mechanism (please see Sections 2.3 and 2.4 for more detail on the meaning of transition functions in DEVS and SiMA settings).

A *Direct Feed Through Transition Function* is executed on each arrival of a new data from input ports and is responsible for producing a response from output ports. Thus, we define a special atomic model, called *Converter Model* that implements only the *Direct Feed Through Transition Function (δ_{dft})* and executes the appropriate converter function for the received value of the input port (See Figure 4.3 for an example of the δ_{dft} code).

4.4.2 Converter Generation

After the generation of the model described in Section 4.4.1 the next stage is the generation of the converter code. Our *Converter Model* contains a function named *ConvertItem* that lists the conversion statements for each field of the destination data type based on their glue invariants (Figure 4.4).

A glue invariant has the form $x = f(y, z, ...)$, where x is the destination data type and $y, z, ...$ are sources. Most of the statements in f are defined in the Event-B Machine or its Context. So the code generation for the converter function depends on the specifi-

```
public override void DirectFeedThroughTransitionWithObjects(string portName)
{
    foreach(var item in inputManager.ExternallyCreatedObjects)
    {
        objectMapping.Add(item.ObjectID, outputManager.CreateObject());
        ConvertItem(item, objectMapping[item.ObjectID]);
    }

    foreach(var item in inputManager.ExternallyUpdatedObjects)
    {
        ConvertItem(item, objectMapping[item.ObjectID]);
    }
    ... // other DirectFeedThroughTransition actions not related
        // to converter logic
}
```

Figure 4.3: Sample Direct Feed Through Transition Function

cation of f and its implementation patterns. We adopt the approach of [47] regarding the target patterns in the C# language and generate statements in glue invariants (Figure 3.7) as statements of C# language (Figure 4.4).

4.4.3 Compensating for the Shortcomings of Event-B Type System

Event-B itself is not a fully-fledged programming language; it does not have control logic constructs such as loops or primitive mathematical functions (e.g. sin, sqrt etc.) however, some glue invariants require these constructs and to compensate for this shortcoming we support the manual implementation of complex glue-functions.

To elaborate further, for a glue-statement such as $x = f(y) + q(z)$ we can generate full details of f and leave the implementation of q to the user. The C# language supports *partial class* definitions that allow the user to define his/her functions in different files. Our code generator will generate f in main file and require the user to implement q in another file to be used in converter.

Although this extension is instrumental in allowing the definition of refinements involving complex expressions, it does limit our automatic verification capability since we can no longer use the implementation details of q in our proofs. However, since

```
private void ConvertItem(DetailedDataPacket source, DataPacket dest)
{
    MonitorSource(source);

    dest.Distance = ScalarLength3D(source.Direction);
    dest.Location = (new Vector2D((source.Position).X, (source.Position).Y));
    dest.OriginID = source.OriginID;
    ... // mappings for other variables of DataPacket object

    MonitorDestination(dest);
}
```

Figure 4.4: Sample glue converter function

this extension is implemented because of the limitations of Event-B language, it can be omitted if, in the future, the Event-B language is enriched to support our requirements.

4.4.4 Using Invariants for Runtime Verification of Converter Output

Allowing the manual insertion of conversion code appears to be a loophole in our automatic verification capability. However, our Event-B machines involve invariants that provide constraints on variables and the code generator can use these invariants for converter input and output checking.

We generate assert statements as given in Figure 4.5 and place them inside the *MonitorSource* and *MonitorDestination* methods and call them at the beginning and end of *ConvertItem* method in Figure 4.4. The assert statements can be executed in the debugging phase of software development and they can be omitted in the released executable. This allows the run-time verification to execute in debugging and does not reduce the performance of the deployed application.

```
Debug.Assert(item.Type != PacketTypes.Result || item.Distance > 0,
                "DataPacket: inv8 verification error");
Debug.Assert(item.Distance < 2000, "DataPacket: inv9 verification error");
Debug.Assert(item.OriginID != item.TargetID,
                "DataPacket: inv11 verification error");
... // monitor functions for other DataPacket object variables
```

Figure 4.5: Sample content for a monitor function

4.5 Details on Case Study

We used our case study (Section 3.3) as an illustrative example to explain our methodology. During the explanation we focused on methodology, hence skipped the details of the case study. However these details require some attention, because by implementing the case study we learned some pros and cons of our methodology. This section will describe these details along with the lessons learned from the case study.

Before going into details, we would like to inform the reader that the implementation of simulation models in the case study require deep information about sensor technologies and radio frequency transmission calculation. The main purpose of this thesis is not the sensing calculations, and the implementation and use of converters are sufficient. So we omitted these calculations in model implementations and used abstract calculation functions that probably generate false results.

As illustrated in Figure 3.2, our case study uses three more data types other then Data Packets. By introducing low resolution sensor model, these data types also need to have a converter to let the simulation execute.

4.5.1 Platform Data

PlatformData (Figure 4.6) values are produced by the Platform Model to transfer the information about the object that sensors are tracking. Sensors are using the position and extent information about the platform object to produce the results of their sensing computation. As indicated for *DataPacket* in Chapter 3 these results include distance of the platform to a sensor.

MACHINE PlatformData # Object
REFINES LowResPlatformData
SEES MrmTypeContext, Functions
VARIABLES

Position
Heading
Velocity
Extent
Material

INVARIANTS

inv1 : $Position \in Vector3D$ # TYPE
inv2 : $Heading \in Vector3D$ # TYPE
inv3 : $Velocity \in Vector3D$ # TYPE
inv4 : $Extent \in Extent3D$ # TYPE
inv5 : $Material \in MaterialTypes$ # TYPE
...

...
END

Figure 4.6: Event-B machine for PlatformData object

MACHINE LowResPlatformData # Object
SEES MrmTypeContext
VARIABLES

$Location$
$Velocity2D$
$Type$

INVARIANTS

inv1 : $Location \in Vector2D$ # TYPE
inv2 : $Velocity2D \in Vector2D$ # TYPE
inv3 : $Type \in PlatformTypes$ # TYPE

...
END

Figure 4.7: Event-B machine for LowResPlatformData object

glue1 : $Location = mk\_Vector2D(Vector3D\_X(Position) \mapsto Vector3D\_Y(Position))$
glue2 : $Velocity2D = Calculate2DVelocityWoHeading(Heading \mapsto Velocity)$
glue3 : $Type = MapPlatformType(Extent \mapsto Material)$

Figure 4.8: Glue invariants for PlatformData object

PlatformData (Figure 4.6) object has five attributes to be used by sensors; Position, Heading, Velocity and Extent attributes are defined in 3D space, and Material is an enumeration of Steel, Aluminum, Iron, Plastic and Composite that effects the sensor range.

However, low resolution sensors do not need these details and expect a *LowResPlatformData* (Figure 4.7) object with simpler attributes. First of all, Location and Velocity attributes are defined in 2D space, because low resolution sensors assume the world as a flat surface and don't expect a terrain altitude. Second, the Heading information is not used, because platforms are assumed to have heading to the direction of their velocity. And third, platforms are categorized into types, including Car, Truck, Tank etc., that reduces the need for Extent and Material attributes.

```
dest.Location = (new Vector2DM((source.Position).X, (source.Position).Y));
dest.Type = MapPlatformType(source.Extent, source.Material);
dest.Velocity2D = Calculate2DVelocityWoHeading(source.Heading,
                                                source.Velocity);
```

Figure 4.9: PlatformData converter routines

For converter generation we need to define the glue invariants for the attributes of *LowResPlatformData* (Figure 4.8):

Location Converting the 3D Position attribute to 2D Location attribute is a bit trivial; with the assumption of the simulation space is nearly flat surface, that does not affect the low resolution sensor results.

Velocity2D Calculating Velocity2D attribute requires a complex computation that includes: a) calculating the scalar length of Velocity, b) calculating scalar times of Heading with length of velocity, c) converting the resultant vector into 2D. Implementing these kinds of steps within Rodin editor is not always applicable because of the limited space. The editor is not designed to have so many equations in a single invariant, and it does not allow easy readable long equations. So we can leave the implementation post code generation.

Type To calculate Type attribute we need to build a mapping of Extent and Material attributes to the restricted set of PlatformTypes enumeration. This conversion does not have an accurate mapping, but we assume that low resolution sensors do not need that accuracy. Again this mapping requires a complex mapping function that is not applicable in Rodin editor, and left to implementation after code generation.

After definition of glue invariants, our code generator produces a code stub including the one in Figure 4.9. The code does not include monitoring statements, because PlatformData structures do not need a validation invariant.

MACHINE SensorInfo # Object
SEES MrmTypeContext
VARIABLES

Position
Extent
AntennaPower

INVARIANTS

inv1 : $Position \in Vector3D$ # TYPE
inv2 : $Extent \in Extent3D$ # TYPE
inv3 : $AntennaPower \in \mathbb{Z}$ # TYPE
...

...

END

Figure 4.10: Event-B machine for SensorInfo object

4.5.2 Sensor Information

SensorInfo (Figure 4.10) values are produced by Sensor Models to give the descriptive information of sensors. This information is required by the Environment Model to compute the environmental conditions of the sensors that effect sensing units and the radio signal transmission.

SensorInfo (Figure 4.10) object has three attributes to be used by the environment model; Position and Extent attributes are defined in 3D space, and AntennaPower is a numeric value that affects the sensor signal transmission range.

However, low resolution sensors do not produce same level of details and use a *LowResSensorInfo* (Figure 4.11) object with simpler attributes. First of all, Location is defined in 2D space, because of the same reason for *LowResPlatformData* objects (Figure 4.7). Second, sensors are categorized into types, including Class1, Class2 etc., that reduces the need for Extent and AntennaPower attributes.

For converter generation we need to define the glue invariants for the attributes of *SensorInfo* (Figure 4.12):

MACHINE LowResSensorInfo # Object
REFINES SensorInfo
SEES MrmTypeContext, Functions
VARIABLES

$Location$

$Type$

INVARIANTS

inv1 : $Location \in Vector2D$ # TYPE

inv2 : $Type \in SensorTypes$ # TYPE

...

...

END

Figure 4.11: Event-B machine for LowResSensorInfo object

glue1 : $Position = mk\_Vector3D(Vector2D\_X(Location) \mapsto Vector2D\_Y(Location) \mapsto 0)$

glue2 : $Extent = GetExtentOfSensor(Type)$

glue3 : $AntennaPower = GetAntennaPowerofSensor(Type)$

Figure 4.12: Glue invariants for LowResSensorInfo object

```
dest.AntennaPower = GetAntennaPowerofSensor(source.Type);
dest.Extent = GetExtentOfSensor(source.Type);
dest.Position = (new Vector3DM((source.Location).X, (source.Location).Y, 0));
```

Figure 4.13: SensorInfo converter routines

Position Converting the 2D Location attribute to 3D Position attribute is a bit trivial, with the same assumption of Location attribute of *PlatformData* object.

Extent and AntennaPower Sensors that are built with the same type has the same Extent and AntennaPower attributes. For example Class1 type of sensors has extent of 30mm Width, 20mm Height and 70mm Length, and 42dBm AntennaPower. Thus, converting Type attribute to Extent and AntennaPower attributes uses a simple mapping function.

After definition of glue invariants, our code generator produces a code stub including the one in Figure 4.13. The code does not include monitoring statements, because SensorInfo structures do not need a validation invariant.

4.5.3 Environment Information

EnvironmentInfo (Figure 4.14) values are produced by the Environment Model to give the environmental information for sensors. This information is required by the Sensor Models to compute the capabilities of sensing units and the radio signal transmission that is effected by the environmental conditions.

EnvironmentInfo (Figure 4.14) object has three attributes to be used by sensor models; Humidity and SNR are numeric values that affect the sensor signal transmission range and Floor is an enumeration that effects the sensing unit.

However, low resolution sensors do not require same level of details and use a *LowResEnvironmentInfo* (Figure 4.15) object with a simpler attribute. Low resolution sensors just use a noise level enumeration, which consists of Low, Medium and High members. Sensors use this enumeration to compute their transmission capability by some constant multipliers, such as in Low noise levels sensors can receive signals

MACHINE EnvironmentInfo # Object
REFINES LowResEnvironmentInfo
SEES MrmTypeContext, Functions
VARIABLES

$Humidity$
$Floor$
SNR

INVARIANTS

inv1 : $Humidity \in \mathbb{N}$ # TYPE
inv2 : $Floor \in FloorTypes$ # TYPE
inv3 : $SNR \in \mathbb{Z}$ # TYPE
...

...
END

Figure 4.14: Event-B machine for EnvironmentInfo object

MACHINE LowResEnvironmentInfo # Object
SEES MrmTypeContext
VARIABLES

$Level$

INVARIANTS

inv1 : $Level \in NoiseLevels$ # TYPE
...

...
END

Figure 4.15: Event-B machine for LowResEnvironmentInfo object

$$\texttt{glue1} : Level = CalculateNoiseLevel(Humidity \mapsto Floor \mapsto SNR)$$

Figure 4.16: Glue invariants for EnvironmentInfo object

```
dest.Level = CalculateNoiseLevel(source.Humidity, source.Floor, source.SNR);
```

Figure 4.17: EnvironmentInfo converter routines

from 100m distance. Low resolution sensors do not change their sensing unit capability, so they do not use Floor type attribute or any other. As we use low resolution sensors away from our search zone, we accept the ignorance of affects on sensing unit.

For converter generation we need to define the glue invariant for the attributes of *LowResEnvironmentInfo* (Figure 4.16):

Level If we have information about inner computation logic of low resolution sensors we can generate a conversion function that converts the set of Humidity, Floor and SNR values into a noise level that low resolution sensor models operate normally. If not, we have to use some assumptions which may affect incorrect conversion results, however it is acceptable for low resolution sensors as described in the previous paragraph.

After definition of glue invariants, our code generator produces a code stub including the one in Figure 4.17.

EnvironmentInfo attributes have logical limits that can be verified by the monitoring statements. We define the invariants in Figure 4.18 and our code generator generates the statements in Figure 4.19.

inv4 : $Humidity \leq 100$

inv5 : $SNR < 10$

Figure 4.18: Monitor invariants for EnvironmentInfo object

```
Debug.Assert(source.Humidity <= 100,
             "EnvironmentInfo: inv4 verification error");
Debug.Assert(source.SNR < 10,
             "EnvironmentInfo: inv5 verification error");
```

Figure 4.19: EnvironmentInfo monitoring routines

4.6 Lessons Learned

We learned some lessons that come up from the problems involved in the implementation. These problems do not prevent our methodology to execute, but requires some more work on implementing the tool set.

4.6.1 Name Conflict Problem

Event-B does not have any grouping for identifiers. If same identifier name is used on both abstract and concrete machines, there is no way to use them in glue functions. It is assumed that the value is transferred as is in the refinement. And if same identifier is used with different data type on abstract and concrete machine, Rodin generates an error that prevents the use of machines in code generation.

To overcome this problem, we intentionally changed some attribute names inside the case study. For example we used Position for detailed, 3D attributes and Location for low resolution, 2D attributes.

The solution to the problem has two different ways:

- Event-B editor can be extended to allow use of machine names along with attributes in glue functions. This issue requires a deep investigation of Rodin ed-

CONTEXT FunctionSample
EXTENDS DetailedPlatformDataContext
CONSTANTS

$ScalarLength3D$
$Sqrt$
$ScalarTimes3D$
$Conv3Dto2D$
$Calculate2DVelocityWoHeading$

AXIOMS

axm1 : $ScalarLength3D \in Vector3D \rightarrow \mathbb{N}$
axm2 : $ScalarTimes3D \in (\mathbb{Z} \times Vector3D) \rightarrow Vector3D$
axm3 : $Sqrt \in \mathbb{Z} \rightarrow \mathbb{N}$
axm8 : $Conv3Dto2D \in Vector3D \rightarrow Vector2D$
axm4 : $Calculate2DVelocityWoHeading \in (Vector3D \times Vector3D) \rightarrow Vector2D$
axm5 : $\forall x, y, z \cdot ScalarLength3D(mk\_Vector3D(x \mapsto y \mapsto z)) = Sqrt(x * x + y * y + z * z)$
axm6 : $\forall x, y, z, k \cdot ScalarTimes3D(k \mapsto mk\_Vector3D(x \mapsto y \mapsto z)) = mk\_Vector3D(k * x \mapsto k * y \mapsto k * z)$
axm7 : $\forall x \cdot Sqrt(x * x) = x$
axm8 : $\forall x, y, z \cdot Conv3Dto2D(mk\_Vector3D(x \mapsto y \mapsto z)) = mk\_Vector2D(x \mapsto y)$
axm9 : $\forall v, h \cdot Calculate2DVelocityWoHeading(v \mapsto h) = Conv3Dto2D(ScalarTimes3D(ScalarLength3D(v) \mapsto h))$

Figure 4.20: Calculate2DVelocityWoHeading implementation in a context file

itor architecture and high implementation work, which goes beyond the scope of this thesis.

- Event-B Machine Generator (Section 4.2) can be extended to include the machine names as prefix or postfix to the attributes, but this hardens the human-readability of the machines and glue invariants.

4.6.2 Complex Glue Invariant Problem

As indicated in Section 4.5.1 implementing some glue invariants in Rodin editor is not always applicable. Both the editor space and function implementation complexity in set theory hardens the implementation of invariants in the editor.

For the example of Vector2D glue function in Section 4.5.1, we need to implement the function *Calculate2DVelocityWoHeading* with different inner functions and include them in a Context file that will look like the one in Figure 4.20. This implementation may look like applicable, but more complex cases are possible and not all of them are possible to implement in Rodin.

Another example for complex glue invariants is switch/case operations, which is enumerating some cases of a value to map to an output value. For the example in Section

4.5.1, implementing MapPlatformType function requires statements like:

- $Width < 5 \wedge Length < 10 \wedge Height < 4 \wedge Material = Iron \Rightarrow Type = Car$
- $Width > 5 \wedge Length > 10 \wedge Length < 20 \wedge Height < 8 \wedge Material = Iron \Rightarrow Type = Truck$
- $Width > 5 \wedge Length < 10 \wedge Height < 8 \wedge Material = Steel \Rightarrow Type = Tank$
- ...

This kind of mapping may not cover all cases of input values, and it is not easily possible for the user to find out which cases are not mapped. However, the converter should cover all possible cases and generate an output for each input. At least it should generate an error to inform the user about the problem. The Rodin editor does not inform the user about missing cases of mapping. And informing the user about missing cases in the CCG requires high implementation work.

On the other hand, set theory is much flexible than current programming languages and generating a code for all cases of Event-B statements would not be possible. To overcome this problem we can limit the use of operations in Event-B as in [47, 58] and implement the code generator for the limited cases, however this will reduce the capabilities of our users.

4.6.3 Restrictions of Current Event-B Language

Event-B language has many capabilities to define a system formally. It supports both type definition and action statements which allow a modeler to define his/her system easily. However it has some shortcomings at the present stage that limits its widespread use as a specification language. Two of them are pointed out below.

- Lack of the floating point type. Event-B does not support floating point numbers. It supports only integer types and enumerated sets as valid types of the attributes or variables. But it is desirable to have that support since virtually all simulations of physical processes require floating numbers for their operation.

- Models usually require complex arithmetic operations such as trigonometry and integral calculations to simulate the behavior of a real system. Event-B is based on set theory, so specifying the actual implementation of an arithmetic function would normally require complex set-theoretical definitions. Therefore it is desirable to have language-level support for built-in arithmetic functions.

4.6.4 Proving Glue Invariants Problem

Our methodology uses "Proving Glue Functions" step (Section 3.4.3) in code generation, however it is not possible to use proving in every glue invariant as we introduced functions. Because of the missing floating point numbers and mathematical operations, definition of all details of glue operations in Event-B is not possible in every case. We covered this deficiency by allowing use of non-implemented functions in glue invariants and left details on generated code. This drawback reduces our possibilities to use Event-B proof system and decrease our formal background.

4.6.5 A Solution Proposal

Event-B primary editor (Rodin) is developed using a very flexible architecture with extensibility in mind. There are many tools and add-ons already implemented to extend its capabilities. Although there is not a rigorous formal process defined for language extensions of Event-B, there seems to be a well-adopted procedure that involves amending the environment and the prover with necessary extensions. Following the same path, additional add-ons to Rodin can be developed to extend Event-B language in our direction. As such, given appropriate time and resources, one can incorporate both floating point support and arithmetic functions such as widely used trigonometric functions etc. into Event-B, through extensions into the editor and prover components of the Rodin environment

CHAPTER 5

DISCUSSIONS

In this chapter we present a discussion of some of the central aspects of the work that is relevant to our solution.

5.1 Discussion of Entity Resolution Mapping Related Work

Different approaches to composition of multi-resolution models [33, 34, 9] have already been proposed in the literature. Among those, a relatively recent work, [34] introduced the concept of 'MR modeling space' to separate aggregation/disaggregation (i.e. resolution conversion) logic from the mechanics of simulation execution (i.e. the simulation space). Part of their Multi Resolution Space (MRS) is the Multi-resolution Event Interface (MREI) which handles the resolution mismatches of messages between models. In fact the idea of logically separating the resolution conversion and the simulation through localizing the entity resolution conversion into MREI has similarities to our approach. The main difference of our approach lies in formalizing the notion of entity resolution conversion as a part of connector models in a DEVS setting. Furthermore, and more importantly, we address the reliability of the resolution conversion, firstly, through the formal verification of Event-B glue invariants and, secondly, through the automatic generation of resolution converter components (i.e. connector models) from declarative specifications (i.e. Event-B machines). In that respect, it is important to note that we specifically target the problem of cross resolution modeling as opposed to multi-resolution modelling. The implication of this emphasis is that our solution tackles the problem of interoperation of models that are

coupled via I/O ports exchanging objects at different resolution levels but representing the same real world entities. Those models, possibly at different resolution levels, that can replace each other in a composition, can be considered as members of a Multi Resolution Model Family (MRMF). In fact, at a given simulation time only one member of this MRMF may be operational where it would have to interoperate with other models (that may possibly be a member of another MRMF), this in turn may require a resolution conversion process due to the difference in the entity resolution levels of the coupled ports. Postulating along the same lines, the problem of replacing a simulation model with another higher or lower resolution model that is a member of the same MRMF at simulation run time (dynamically) can be addressed separately from the viewpoint of interoperation among cross resolution models, although the ramifications of the two problems are related.

Apparently, the most relevant work to our approach is that of [55] and [50] in which the authors present a number of what they call “fundamental observations" regarding the problem of cross-resolution modelling. We found those observations quite useful to determine the qualities and level of comprehensiveness of solutions in this field. We will not go into the details of each of their observations at length; instead we will compare central tenets of their solution, namely the Multi Resolution Entity (MRE) with our solution. Since the authors claimed that MREs offer a solution framework that focuses on the maintenance of consistency based on the fundamental observations mentioned above, we undertake an informal evaluation based on a qualitative comparison of the MRE solution with ours as given in the following three subsections:

5.1.1 Consistency Maintenance

MREs internalize the consistency maintenance via the management of a set of core attributes and a set of reversible mapping functions. MREs maintain “internal consistency” across multiple, concurrent levels of resolution. Within the MRE concept, each entity either maintains state information at all desired levels of resolution or produces attribute values at each simulation step. Simulations involving MREs are based on concurrently reflecting the effects of interactions at all resolution levels. Figure

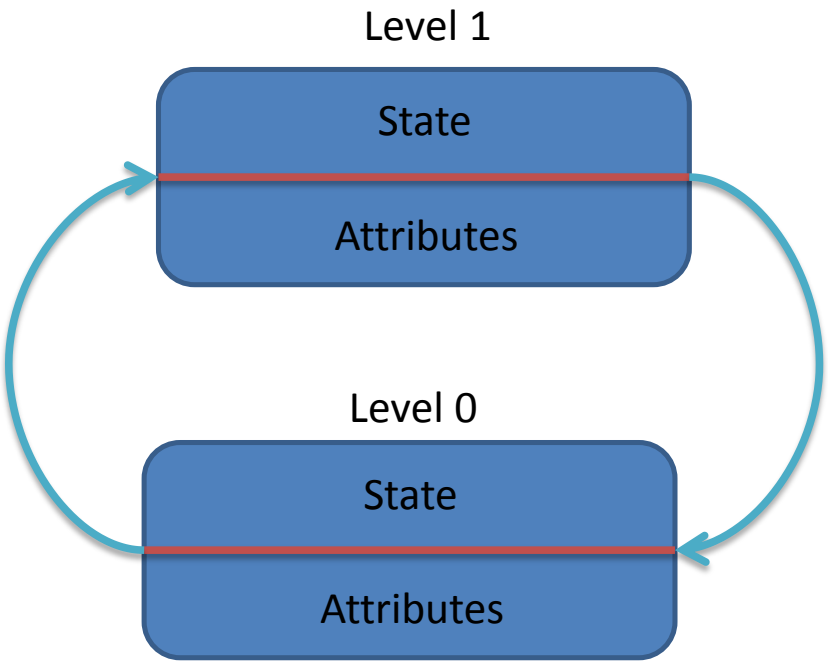

Figure 5.1: Design of an MRE with two resolution levels [55]

5.1 depicts a typical MRE for two levels; $Level_0$ for the low resolution and $Level_1$ for the high resolution. The MRE maintains the attributes at both levels at all times and the two states of the MRE are kept consistent with each other.

In order to maintain consistency among attributes of the different resolution levels, relationships between attributes must be captured. These relationships can be modeled by a directed, weighted graph where the nodes represent the attributes and the edges represent relationships. MRE proposes the notion of an Attribute Dependency Graph (ADG) [50] which depicts the various attributes and sub-entities of the MRE and portrays the relationships among them. An ADG is an encoding of the concurrent multi-resolution interactions problem, and is also an encoding of solutions thereof.

Our solution is similar in many ways and also has some additional advantages in that respect:

- First, the Event-B machines that are linked through refinement relationships to each other can be considered to collectively define a Multi Resolution Entity. From another perspective, the concept of a multi-resolution model family is

embodied by the collection of Event-B machines that are linked via refinement relationships.

- Second, the glue invariants that define the conversion logic between the attributes can be viewed as a specification of a directed graph between those attributes. In fact, the edges of such a graph are annotated with expressions that effectively specify a mapping between the related attributes.
- Third, if there are additional invariant definitions other than the glue invariants (such as range restrictions and type definitions) in the machines, those invariants are also taken into consideration by the prover during verification which provides a further consistency checking mechanism on the resolution mapping.
- Fourth, one important advantage of our approach is that since our mapping is specified using a formal language, its consistency can be verified using a prover, and its implementation can be generated automatically, addressing the concept of correct implementation of the mapping.
- Fifth, in our approach the consistency maintenance logic is internalized (and hence localized) into the notion of connector model which ensures a systematic approach to the simulation construction and an effective management of consistency issues at run-time. Consistency maintenance in the sense of [50] amounts to the preservation of glue invariants in our terms.

It is worth noting that our solution perceives simulation models as black-box components. The relation between the state and I/O of a model is crucial for the comparison between our solution and MRE. If the data flowing through the I/O ports is a direct mapping of the internal model state, the implementation of converters would be equal to the state mappings of MRE. Otherwise, we would lose some information about the state and our converter might not achieve the envisioned success rate of MRE in consistency maintenance.

5.1.2 Staging and Intrusiveness of Resolution Mapping Procedures

Although the authors of [50] do not extend their discussion towards design and implementation issues of MREs, our understanding is that MRE is more intrusive in

terms of the model internals since an MRE and the logic required to map different resolution levels specified within it seems to be coupled with the behavioral logic of the implementation of the relevant model. This implies that a change in the structure (i.e. syntax) of an entity enforces complementary modifications on the internals of the models that consume or produce that MRE. This is something that may raise questions regarding the architectural clarity and maintainability aspects of the overall scheme when it comes to the actual construction of simulations based on this concept.

Our approach separates the definition of multi-resolution entity families from the logic required to map between those multi-resolution entities. This separation is clearly expressed at a declarative level using a formal language. Moreover, our methodology ensures that the data constraints and mapping logic are verified via provers, and implementation is generated in an automated way to support correctness by the construction. The expressive power of the language used (i.e. Event-B) does impose certain restrictions on the complexity of the logic that handles the resolution mapping, however those restrictions are clearly delineated and there are routes to work around them.

5.1.3 Practicality

Since the authors [50] provide a design strategy, rather than a complete solution with guidance for implementation of MREs, there are no details given about how consistency maintenance can be achieved in a simulation and execution of ADG.

Our solution provides a methodology involving both a design strategy and detailed implementation guidance. We give details of the steps required to apply our methodology and present an example of a complete converter generation. Hence, we describe a practical process to implement a CRM solution based on formal methods and available tools.

5.2 Converters in Relation to Connectors in Component-Based Development

The use of connectors in component-based development is a well-known topic in software engineering [38, 13]. There is also considerable amount of research on developing a formal basis for connectors in the context of component composition [32], [21, 46, 15, 6].

The use of the Connector Model as a first class construct in a DEVS-based simulation construction environment is a specific case of the generalized connector concept discussed in the literature cited above. In [16] and [49], the authors propose drivers that catch incoming real-time events from hardware devices and send output commands to the hardware by user implemented driver objects. These drivers can be viewed as connectors that allow the DEVS models to interoperate with real-time systems.

In our case the connectors play a specific role of entity resolution conversion. Note that, although the adaptation of interaction protocols between components being "glued" is one of the central properties of connectors in general, in our case this is less of an issue. This is because the interaction protocol of the DEVS models is under the strict control of the simulation engine which rigorously applies the DEVS simulation protocol [65]. Through this protocol the engine drives the models via a standard control interface that is provided by each model by definition. Therefore port couplings between models act only as data flow channels. Since our connectors are defined to exist along port couplings (rather than between the engine and the models) their functionality is limited to data conversion.

It is worth noting that the use of Event-B to facilitate verification and automatic construction of connectors in the DEVS setting to overcome resolution mismatches is a novel aspect of our methodology in terms of the application of formal methods in component-based development of multi-resolution simulation software.

5.3 The Unconventional Usage of Refinement in Entity Resolution Mapping

In its conventional sense refinement, is a process to derive concrete models from existing abstract models. As such, it is used to develop more concrete models, closer

to the implementation. Many tools such as code generators developed for Event-B community normally operate on the most refined models, because refinement leads to more precise semantics and increases the level of detail, both of which are desirable properties in a system development process. In the work presented in this thesis, we take a different perspective, in that, we use refinement to define the relations between existing entities and use the defined relation itself as the source for converter code generation. Evidently, our scheme does not exclude cases where entities and refinement relations that specify resolution mapping logic are defined together at the simulation design stage. However, our solution can operate in a setting where there are entities that represent data types at different resolution levels already implemented and ready to be used in a simulation. In such a setting, we add refinement relations between those existing entities to specify how to compensate for the resolution mismatches between them in a formal language. As described in Chapter 3, refinement allows us to define fine grained mapping expressions among attribute pairs and allows the proof sub-system to be used to validate mapping specifications for the preservation of constraints and consistency. Thus, we exploit the power of refinement but employ it in an unconventional way. It is also worth noting once again that we use the proof capabilities of the Event-B tools [5] in refinement to validate our converters for constraint preservation.

5.4 Entity Resolution Mapping without DEVS and Event-B

Our approach on entity resolution mapping relies on a modeling framework that allows composition of models, DEVS, and a formal language that allows definition and verification of converter statements, Event-B. DEVS and Event-B seems to be the elementary root of our approach, however they are just well-defined bases to describe our methodology. We can use any other modeling framework and formal language that meets our basic necessities.

5.4.1 Modeling Framework

DEVS is a modeling and simulation framework that allows model composition (Section 2.3). Our methodology introduces use of connectors between simulation models to overcome the interoperability problems of data resolution mismatch. We use DEVS to describe the rules of a modeling and simulation framework and composition requirements and focus on the connector solution we provide.

Any other modeling and simulation framework that allows composition of models with use of connectors would be sufficient for demonstration of our methodology. For example, HLA (Section 2.5) is a modeling and simulation framework, developed for standardizing distributed simulations. If we assume federates (distributed nodes in HLA), as the composition units of modeling, we can use the connector approach between federates and apply our methodology.

There are several commercial products of HLA implementations and some connector solutions to overcome interoperability problems. These include the Pitch pRTI, Adapters and Extenders [53], which allow HLA simulations to communicate with other HLA implementations and simulation frameworks (i.e. DIS [1]). They also allow breaking down a single HLA simulation into sub-simulations and making them execute together.

To implement our methodology in HLA, we need to use one of these commercial products or develop our own. We can place our connectors between federates and RTI (centralized distribution service) and execute our conversion functions for each type of object.

5.4.2 Formal Language

Event-B is a formal specification language that allows systems specification and verification with formal definitions (Section 2.2). Our methodology introduces use of Event-B for specifying data types in a formal language and use language capabilities to define and verify conversion steps. We use Event-B to describe our steps of data type definition and verification process and use its actively developed open-source

editor Rodin [5].

Any other specification language that allows our data type definition and conversion step verification would be sufficient for demonstration of our methodology. For example we studied the B Method [2], which is the ancestor of Event-B, but it did not meet requirements of our methodology. It allows data type definition with B machines, but the refinement relations between machines are not suitable for our conversion definitions. Aterial B [18] is an editor for B Method, and had been commercial-only when this research was started.

We also studied rCOS [39], based on UTP [30], which is a specification language on refinement calculus for object systems. It is a mathematical characterization of object-oriented concepts and it defines a limited but functional Object Oriented language. However, its editor environment and development community were not promising to use in our methodology demonstration.

As we proposed for Event-B, both B Method and rCOS or any other specification language can be extended and/or their editor can be re-implemented to support our methodology requirements, but it will probably require a high development effort.

CHAPTER 6

HETEROGENEOUS DEVS SIMULATIONS WITH CONNECTORS

Our approach proposes using connectors between models of heterogeneous DEVS simulations to resolve the data type and time resolution mismatch.

We anticipate that the properties of Reo such as exogenous coordination (i.e., by third parties), compositional construction, arbitrary mix of synchrony and asynchrony, user-defined primitives and dynamic reconfigurability are well suited to adaptation requirements often found in complex component based simulation applications. In this chapter, we only scratch the surface of the potential realm of exploiting a powerful "glue language" such as Reo for composition of incompatible simulation models in a DEVS setting. In particular, we focus on compositional construction property of Reo and demonstrate the utility of that aspect adopting a practitioner approach.

6.1 Time Resolution Connectors

One of the problems for heterogeneous simulations is the mismatch of time resolution. This happens when the time steps used for state updates of source and destination models are not equal, and this difference in the update frequency creates a behavioral inconsistency at the destination model. For example, the source model may produce values at $\Delta_{src} = 2x$ time units and destination model may expect values at $\Delta_{dst} = x$ time units. Composition of these models is problematic due to missing input values for the destination model at odd x steps.

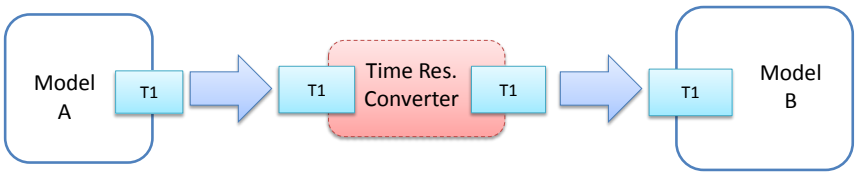

Figure 6.1: A sample time resolution DEVS connector

A connector between these two models can help solve the problem by calculating the missing values of intermediary time steps and sending them to the destination model. For an instance of a time t (e.g. an odd x step), the connector needs to estimate an output value A_t by using the past values $(A_0, A_{2x}, A_{4x}...A_{t-3x}, A_{t-x})$. Estimating sequences of series is called *extrapolation* and many different algorithms have been devised for it [12, 57]. A model composer can use one of these extrapolation algorithms to compute the missing values in his/her time resolution connector.

For each time resolution conversion requirement, a connector model needs to be developed (Figure 6.1). The connector model consists of standard DEVS functions $(\delta_{int}, \delta_{ext}, \lambda)$, in compliance with DEVS simulation execution protocol, that stores input values and dispatches extrapolated values to output ports at required time steps. Connector state(S) is an array-like data structure that stores the list of input values. External transition function (Procedure 1) adds each input value to the state S. Then it updates the model next time to the minimum time value required for values stored in S. Output function (Procedure 2) calculates the extrapolated values at current time and dispatches them to the output port. Internal transition function (Procedure 3) updates the model next time to the minimum time value as in external transition function.

input : (e, t, S) external event, current time and state
output: (S) updated state

insert (e, t) *to* S;
calculate next time nt from S;
set model next time to nt;

Procedure 1: External transition function(δ_{ext}) of time resolution converter

More complex timing inconsistencies between interacting models are also possible.

input : (t, S) current time and state

output: N/A

foreach (v, vt) *at state vector* S **do**

    calculate extrapolated value (d) *of* (v, vt, t);

    put d *on output port*;

end

Procedure 2: Output function(λ) of time resolution converter

input : (t, S) current time and state

output: N/A

calculate next time nt from S;

set model next time to nt;

Procedure 3: Internal transition function(δ_{int}) of time resolution converter

For instance, $\Delta_{src} = 3x$ and $\Delta_{dst} = 5x$ which have larger *least common multiple*, or rational coefficients may be encountered. Solution for such cases is essentially the same in terms of the mechanism, except that the extrapolation function logic would be different. For example, in the case of rational coefficients, for all steps of output an extrapolated output that depend on the rational time value is computed.

6.2 Data Conversion Connectors

If two distinct models are to be composed via their input and output ports, either they must use identical input/output data types or a conversion is needed in between. Evidently, our assumption here is that composable models are logically related with the same real world entities and phenomena but may have non-identical data representations due to different objectives or processes that lead to their creation.

To compose such incompatible models in a DEVS setting, our solution proposes to use converters between data types defining the ports of connected models (Figure 6.2). Put simply, these converters are connectors between output and input ports of models. Similar to models, converters have input and output variables, but unlike atomic models they do not participate in simulation model state calculations. In fact,

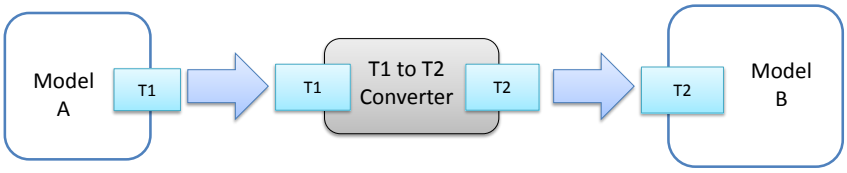

Figure 6.2: A sample data conversion DEVS connector

a convertor's behavior is not time dependent whereas an atomic model can have time-dependent behavior. This is because a data conversion operation does not consume simulation time; it is essentially an algebraic operation on input data. Formal representation of a data conversion connector was discussed in Section 3.1.

A DEVS connector model for data conversion is structurally and mechanically (i.e. in terms of execution mechanism) similar to the time resolution conversion converter discussed in Section 6.1. The most important difference is that the processing of each input value and the dispatch of the output are performed at the same time step. External transition function (Procedure 4) converts each input value and stores the converted value in the state S. Then it sets the model next time equal to current time, to be able to dispatch output on the same simulation time. Output function (Procedure 5) gets the values from S and publishes them to the output ports. Internal transition function (Procedure 6) clears the state S and prepares the model to wait for the next input.

input : (e, t, S) external event, current time and state
output: (S) updated state

calculate converted value c from e;
insert c to S;
`// immediate response`
set model next time to t;

Procedure 4: External transition function(δ_{ext}) of data converter

input : (t, S) current time and state

output: N/A

foreach *converted value c at S* **do**

    put c on output port;

end

Procedure 5: Output function(λ) of data converter

input : (t, S) current time and state

output: (S) updated state

remove all converted values from S;

`// Passivate until next input value`

set model next time to ∞;

Procedure 6: Internal transition function(δ_{int}) of data converter

6.3 Multi-Input Multi-Output Connectors

Both types of connectors we described in previous sections are simple channels that process single input port and produce values on single output port. However in some cases we may need to combine two or more input ports and may produce two or more output ports.

We build our connectors as atomic DEVS models that naturally allow multiple input and output ports. This capability allows us to build a connector that has multiple inputs and outputs as in Figure 6.3.

This type of connectors can be developed for a variety of purposes, such as: a) forming a union of two types of data values, b) decomposing a single data value into simpler types of values, c) synchronizing two input values so that an output could be produced upon arrival of both, d) any combination of all.

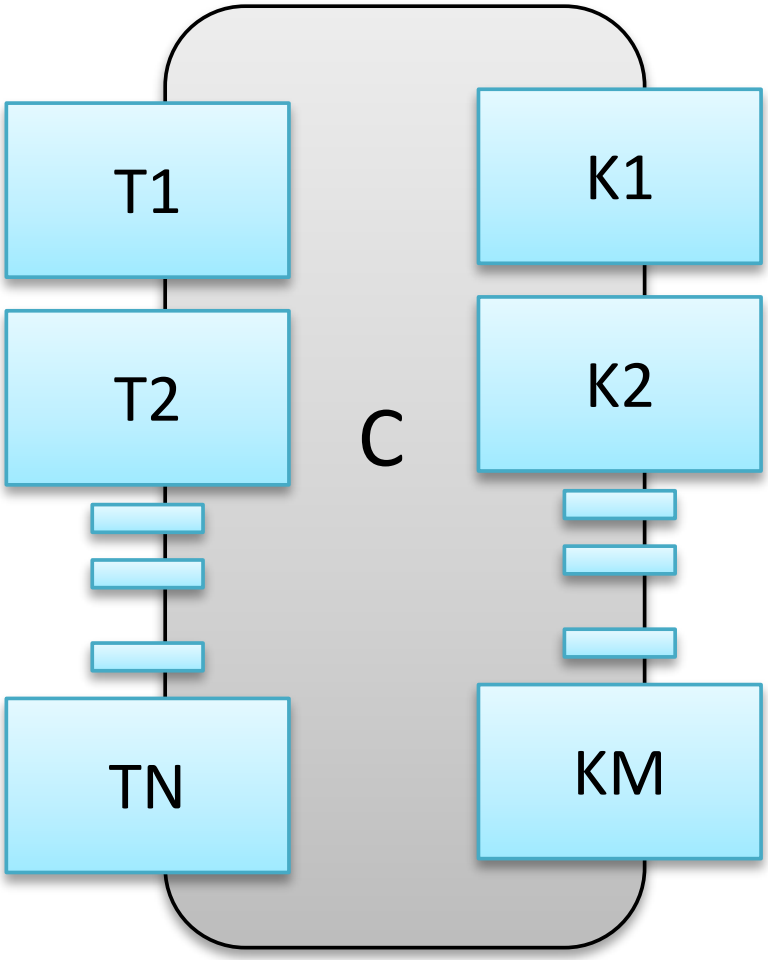

Figure 6.3: A sample N-input M-output connector

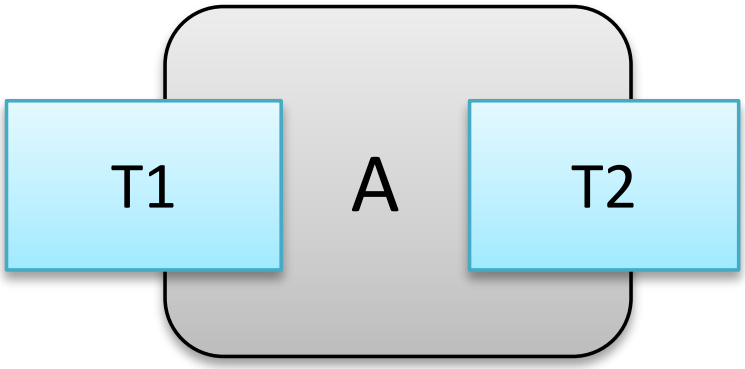

Figure 6.4: T1 to T2 connector A

6.4 Composite Connectors

As introduced above, simple connectors can be used to compensate for data or time resolution mismatches in a DEVS setting. A data converter A that converts $T1$ to $T2$ can be used in any simulation that needs to have a connection of ports with $T1$ and $T2$ data types (Figure 6.4).

For more complex cases, we benefit from "connector reuse" to build more complex (or composite) connectors by combining existing simpler connectors. If, for instance, we have another converter B of $T2$ to $T3$ connection available to us, we can build a composite connector by constructing a DEVS coupled model of A and B connectors. This will effectively end up in a connector that handles $T1$ to $T3$ conversion (Figure 6.5). Furthermore, we can combine data converters with time resolution converters and build a converter that compensate for both data and time resolution inconsistencies.

6.5 Representing Composite Connectors in Reo

Connector composition problem in Reo [8] is treated in a way appropriate for exploitation in a DEVS setting. The concepts of DEVS connectors and DEVS model

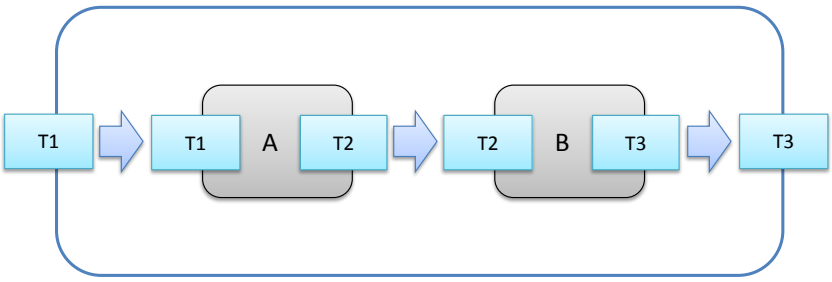

Figure 6.5: Composite connector of T1 to T3

ports are analogous to channels and sinks of a Reo circuit, respectively. Similar to Reo, any combination of DEVS connectors can be used to build different composite connectors.

Our connectors are well-defined entities with their input/output ports and their function. For example a connector A in Figure 6.4 is defined as a T1 to T2 data converter connector, where T1 and T2 are well-typed (or strongly typed) with respect to a common type system. Thus connector A can be used in any simulation that needs a T1 to T2 data conversion. Hence an arbitrary set of connectors can be composed to use in any simulation that a connector is required. In Reo, channels are also well defined and can be listed in a channel set (Figure 2.6).

Input ports and output ports are two ends of DEVS connectors, which correspond to source and sink channel ends in Reo. Connector accepts data from input port and dispenses data from output port. Connector composition is handled by port couplings. If a connector output port is coupled with another connector's input port then these two connectors are composed. In Reo, composition is handled through nodes; in DEVS we can consider port couplings as Reo nodes.

Reo allows composition of channels to build connectors. As we realize our connectors as atomic models, our solution also allows composition of DEVS connectors and building composite connectors as coupled models. A composition can take a variety of forms; examples are given in Figure 6.5 and Figure 6.6. Thus DEVS connector compositions can also be represented as a Reo circuit (Figure 2.7).

On the other hand, multi-input multi-output connectors have complex behavior, which

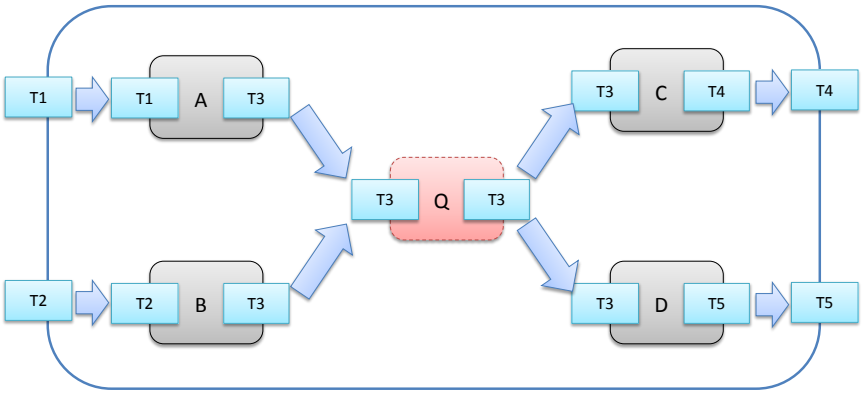

Figure 6.6: A sample connector circuit

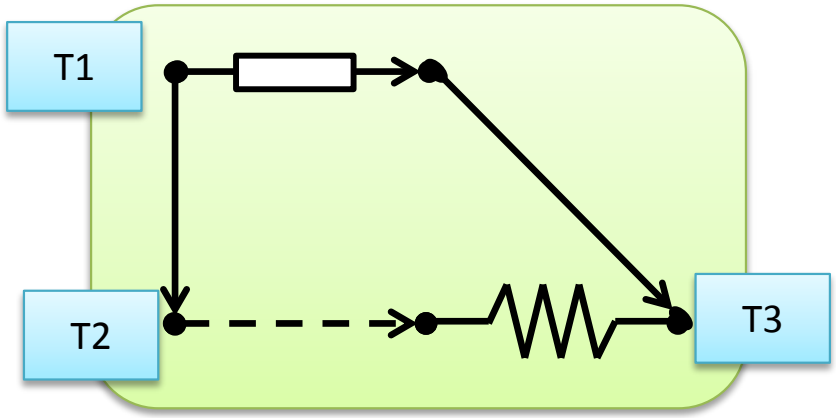

Figure 6.7: A sample MxN connector with Reo circuit

can be modeled with a Reo circuit (Figure 6.7). The interconnections of input and output ports can be designed with Reo channels and implemented by a Reo code generator (e.g. [40]) or by hand.

6.6 Case Study

To demonstrate our approach, we can think of a simple DEVS simulation application for traffic training that is built by composing two models, namely *a traffic model* and *a platform model*. The *platform model* calculates the values for various attributes of a *Car* object (Figure 6.8) at each step of the simulation and publishes this *Car* object

from its output port. The *traffic model* reads the *Car* object as its input and calculates interactions between *Car* objects (collision etc.) with the traffic statistics for post simulation analysis.

Assume, for instance, that a previously built, ready to use model producing a *truck* object (Figure 6.9) will be introduced to the traffic training simulation. The new model's output is not syntactically compatible with the *traffic model* as it stands. Looking at the content of the *truck* object, we can see that it is not logically compatible to what traffic model expects. So, if we can convert the output of the truck platform model (i.e. truck object) to a car object and fix the time resolution incompatibilities we can continue to execute our simulation without any modification to the traffic model.

- width : float
- height : float
- length : float
- licensePlate : string
- colorName : string
- posX : float
- posY : float
- posZ : float
- velX : float
- velY : float
- velZ : float

Figure 6.8: Attributes of Car object

6.6.1 Time Resolution Conversion

Truck model is updating truck object attributes at every 60 seconds, however the car model does that at every 20 seconds, and since trucks usually move slower than cars (thus truck model does not require a high update frequency). But traffic model is

- Model : string
- LeftHandDrive : boolean
- ColorRGB : RGBVector
- Position : Vector3D
- Velocity : Vector3D

Figure 6.9: Attributes of Truck object

expecting attribute updates at 20 seconds intervals to calculate collisions accurately.

In fact, this kind of "predictive" calculation of state for intermediary time values is a well-known technique in modeling and simulation, known as *dead reckoning*. Dead reckoning (DR) is a specialized extrapolation technique used in modeling and simulation domain and it is a fundamental feature of the DIS standard [1]. It was developed to compute intermediary position values to reduce the amount of communication. There are many studies about dead-reckoning for various modeling domains and solutions, e.g. [14, 27, 44].

For multiples of 60 seconds, TRC does not use a converter function and transfers the source data without modification. But for intermediate steps of 20 and 40 seconds, we need to build a dead reckoning function. As we have the velocity information of the Truck object we can estimate the missing position values at the steps of 20 and 40 seconds.

To solve this problem we can build a time resolution converter (TRC) between Truck and Traffic models. Since we need to build a data conversion connector (DCC) (to be discussed in Section 6.6.2) too, we can either insert our TRC between Truck model and DCC or between DCC and Traffic model. The decision depends on implementation complexity as well as the availability of ready-to-use assets such as dead-reckoning algorithms or the whole TRC model.

For the case of building a TRC between DCC and Traffic model we can use a conversion function such as in Procedure 7.

input : (v, vt, ct) value, value time and current time

output: (c) converted output

$dt \leftarrow ct - vt$;

$c.posX \leftarrow c.posX + (dt/60) \times v.velX$;

$c.posY \leftarrow c.posY + (dt/60) \times v.velY$;

$c.posZ \leftarrow c.posZ + (dt/60) \times v.velZ$;

Procedure 7: Conversion function of time resolution converter

6.6.2 Data Conversion

Solving the time resolution compatibility problem between Truck and Traffic models is not sufficient for building a simulation from these two models. The output data type of the Truck model is not a Car object. A connector between Truck and Traffic models can convert Truck objects to Car objects. The conversion steps are somewhat trivial because Car and Truck objects have similar attributes with different representations. Car object represents position and velocity with different attributes for each axis, however Truck object uses a Vector3D structure, which is a complex data type that consists of X, Y and Z attributes. Conversion of these attributes can be achieved by flattening the Vector3D structure.

Conversion of informative attributes is a bit more complex. Truck model does not publish size attributes of a Truck object but gives the name of the truck model, which implicitly specifies the size information. For example, Truck model produces "1975-1991 Ford E-Series 124 WB" as model name and it can be converted as 4,745 mm Length, 2,029.5 mm Height and 2,029 mm Width [62] for the Traffic model.

A similar solution can be applied to *colorName* attribute of the Car object and *Color-RGB* of the Truck object. However this problem is a bit more complicated, since not all color values (Red, Green, Blue value combinations) have a name. The converter has to map the color values to the nearest named color possible.

The *LeftHandDrive* attribute is a boolean that is true if the Truck is left-hand driven, or false if it is right-hand driven. The traffic model extracts this information from the *licensePlate* of the car object, so we can convert the true value of *LeftHandDrive*

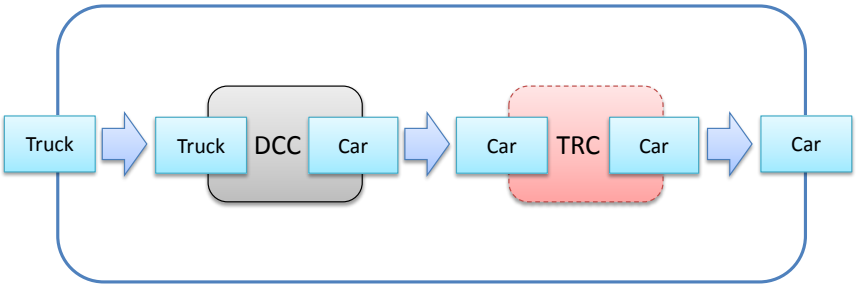

Figure 6.10: Connector designed for case study

attribute to USA and false value to UK. We know the model internals of the traffic model, that the *licensePlate* attribute is just used for drive side of the car object. If it was not, our assumption to map left-hand-drive to USA would cause improper model calculations.

After all steps given above, we finally built the connector in Figure 6.10 for our case study.

CHAPTER 7

CONCLUSION

In this thesis, we have presented our approach to implement a solution for composition of multi-resolution models. The construction of cross-resolution simulations is a multi-faceted, complex enterprise. Any attempt to devise elegant solutions that target all of the facets of such a complex problem is deemed to face enormous difficulties and likely to fall short of delivering a comprehensive remedy that addresses all of those facets. From the very start of our work our objective has been to target a closely correlated subset of those facets (in this case entity, attribute and logical dependency facets), but provide a relatively complete solution for that subset. In that respect, our approach combines the strength of formal approaches and languages, with necessary tool and framework support into a systematic methodology, to deliver a focused but in-depth solution. Clearly, through the formal approaches we sought one that facilitates precise, machine processable semantics; and through the systematic methodology and accompanying tool support we looked for a repeatable process that builds upon rigorous foundations and relieves the simulationist from adopting ad-hoc practices.

To re-iterate the merits of our approach:

(i) it involves a formal proposal to fit the concept of converters into a well-established model composition paradigm, namely, DEVS.

(ii) it provides a practical methodology that offers a well-defined sequence of steps to obtain executable converters for entity resolution mapping, given appropriate descriptions of entities and refinement relations.

7.1 DEVS Connectors

Our approach on using converters for entity resolution mapping introduced the use of connectors between DEVS models. We delved into this problem and presented our approach for composition of DEVS models that suffer from data type and time resolution incompatibilities. Our approach involves the use of connectors as first class DEVS models to convert incompatible data types and bridge over time resolution differences to enable seamless model composition. We also demonstrated the analogy between the notion of modular and hierarchical model composition in DEVS and compositional construction of connectors using channels in Reo. Our approach potentially facilitates automatic discovery and re-use of connectors (i.e. connector models as black-box components) developed for conversion requirements. A library of such connector models can be built and made available for heterogeneous modeling projects.

The idea of building arbitrarily complex connectors via composition of simpler ones is supported both by DEVS coupling semantics and the operational semantics of Reo. Complex connectors that are possible to build due to high expressive power of Reo would also be valid (or legitimate) for DEVS connector models. However this claim deserves a separate formal inquisition and our initial quest indicates that this path seems promising. At least for the case of resolving data type and time resolution incompatibilities the utility of a powerful connector language seems more obvious. Furthermore, Reo provides a powerful calculus that lends itself for dynamic composition of connectors, which is also very promising for dynamic DEVS environments [26]. In fact, the use of such dynamically configurable connectors between incompatible DEVS models could open the door for solutions to many other interoperability problems such as cross-resolution modeling [23] and heterogeneous distributed simulations [67, 48].

7.2 Future Work

7.2.1 Automatic Discovery and Re-Use of Resolution Converters

Our approach potentially facilitates the automatic discovery and re-use of resolution converters (i.e. connector models as black-box components) developed for mapping requirements that were already addressed during earlier simulation exercises. A library of such connector models can be built and made available for projects involving cross-resolution modeling. Since our simulation environment supports strongly typed port definitions both for models and connectors, the construction of a model composition graph that involves semantically equivalent but syntactically incompatible model ports, through appropriate combinations of connectors can be achieved. In fact, our research group has planned future work that can realize such graph building processes in a semi-automated way.

7.2.2 Increase Interoperability with the Help of Accessor Methods on Data Beans

Data beans itself is not sufficient for preserving consistency on refinement. A model can update fields of the data type without paying attention to the semantic meanings. For example increasing the velocity of a platform object may require to change the heading attribute because of the aerodynamics of the platform. A low resolution model would not take care of this situation and effect the high resolution models' calculations. Transfer and use of accessor methods along with the data types through the model ports can solve this problem. If the receiving model does not change a variable directly and use the accessor methods, the consistency will be increased.

Our methodology used Event-B to describe the data types as machines and used the refinement relations to describe the conversion routines. We propose the use of events of Event-B as accessor methods and include them in refinement relations to overcome this problem as a future work.

7.2.3 Connectors to Enhance Distributed DEVS Approach

Distributed simulations generally consist of heterogeneous models that might have several inconsistencies. Our approach on using connectors to overcome some of the interoperability problems in heterogeneous models can be used in a distributed simulation environment. Connectors can be used on input/output ports of the distributed nodes to resolve the data type and time resolution mismatches.

Research work has been planned in our group to implement an implementation of distributed DEVS with the help of connectors. Our implementation will implement one of the Distributed DEVS approaches [61, 67, 43, 67, 48] on SiMA [41] with the help of our connector approach and based on WCF [17] service oriented software development architecture.

REFERENCES

[1] *1278.2-1995 - IEEE Standard for Distributed Interactive Simulation - Communication Services and Profiles*.

[2] J. Abrial. *The B-Book: Assigning programs to meanings*. Cambridge Univ Pr, 1996.

[3] J. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge Univ Pr, 2010.

[4] J. Abrial and S. Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to Event-B. *Fundamenta Informaticae*, 77(1):1–28, 2007.

[5] J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT*, 12(6):447–466, 2010.

[6] R. Allen and D. Garlan. Formalizing architectural connection. In *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, pages 71–80, Sorrento, Italy, 1994. IEEE Computer Society Press.

[7] M. N. Alpdemir. SiMA: a discrete event system specification-based modelling and simulation framework to support model composability. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 9(2):147–160, 2012.

[8] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[9] L. Baohong. A formal description specification for multi-resolution modeling based on DEVS formalism and its applications. *Journal of Defense Modeling and Simulation*, 4(3):229, 2007.

[10] F. J. Barros, B. P. Zeigler, and P. A. Fishwick. Multimodels and dynamic structure models: An integration of DSDE/DEVS and OOPM. In *Proceedings of the 30th Conference on Winter Simulation*, WSC '98, pages 413–420, Washington, D.C., USA, 1998. IEEE Computer Society Press.

[11] D. Bozağaç, G. Karaduman, A. Kara, and M. N. Alpdemir. Sim-petek: A parallel simulation execution framework for grid environments. *The Journal*

of Defense Modeling and Simulation: Applications, Methodology, Technology, 9(4):303–319, 2012.

[12] C. Brezinski. A general extrapolation algorithm. *Numerische Mathematik*, 35(2):175–187, 1980.

[13] T. Bureš. *Generating Connectors for Homogeneous and Heterogeneous Deployment*. PhD thesis, Faculty of Mathematics and Physics, Charles University, 2006.

[14] W. Cai, F. Lee, and L. Chen. An auto-adaptive dead reckoning algorithm for distributed interactive simulation. In *Proceedings of the thirteenth workshop on Parallel and distributed simulation*, pages 82–89. IEEE Computer Society, 1999.

[15] X. Chen, J. He, Z. Liu, and N. Zhan. A model of component-based programming. In F. Arbab and M. Sirjani, editors, *International Symposium on Fundamentals of Software Engineering*, volume 4767 of *Lecture Notes in Computer Science*, pages 191–206. Springer Berlin Heidelberg, 2007.

[16] S. M. Cho and T. G. Kim. Real-time DEVS simulation: Concurrent, time-selective execution of combined RT-DEVS model and interactive environment. In *Proceeding of 1998 Summer Simulation Conference, Reno, Nevada*, 1998.

[17] P. Cibraro, K. Claeys, F. Cozzolino, and J. Grabner. *Professional WCF 4: Windows Communication Foundation with. NET 4*. John Wiley & Sons, 2010.

[18] ClearSy System Engineering. *Atelier B 4 - User Manual*, 2013.

[19] Committee on Modeling and Simulation for Defense Transformation, National Research Council. *Defense Modeling, Simulation, and Analysis: Meeting the Challenge*. The National Academies Press, 2006.

[20] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly. The department of defense high level architecture. In *WSC '97: Proceedings of the 29th conference on Winter simulation*, pages 142–149, Atlanta, Georgia, United States, 1997. IEEE Computer Society.

[21] J. Davies, D. Faitelson, and J. Welch. Domain-specific semantics and data refinement of object models. *Electronic Notes in Theoretical Computer Science*, 195:151–170, 2008.

[22] P. Davis and R. Hillestad. Families of models that cross levels of resolution: Issues for design, calibration and management. In *Proceedings of the 25th conference on Winter simulation*, pages 1003–1012. ACM, 1993.

[23] P. K. Davis and J. H. Bigelow. *Experiments in multiresolution modeling (MRM)*. RAND Corporation, 1998.

[24] P. K. Davis, J. H. Bigelow, and J. McEver. Exploratory analysis and a case history of multiresolution, multiperspective modeling. *RAND Reprints RP*, 925, 2000.

[25] P. K. Davis and A. Tolk. Observations on new developments in composability and multi-resolution modeling. In *Proceedings of the 39th Conference on Winter Simulation: 40 Years! The Best is Yet to Come*, WSC '07, pages 859–870, Washington D.C., 2007. IEEE Press.

[26] F. Deniz, M. N. Alpdemir, A. Kara, and H. Oğuztüzün. Supporting dynamic simulations with Simulation Modeling Architecture (SiMA): a Discrete Event System Specification-based modeling and simulation framework. *Simulation*, 88(6):707–730, 2012.

[27] R. M. Fujimoto. Parallel and distributed simulation. In *Simulation Conference Proceedings, 1999 Winter*, volume 1, pages 122–131. IEEE, 1999.

[28] R. M. Fujimoto and R. M. Weatherly. Time Management in the DoD High Level Architecture. In *In Proceedings of the 1996 Workshop on Parallel and Distributed Simulation, 60-67. Institute of Electrical and Electronics Engineers, Piscataway*, pages 60–67. IEEE Computer Society, 1996.

[29] S. Hallerstede. On the purpose of Event-B proof obligations. In E. Börger, M. Butler, J. Bowen, and P. Boca, editors, *Abstract State Machines, B and Z*, volume 5238 of *Lecture Notes in Computer Science*, pages 125–138. Springer Berlin Heidelberg, 2008.

[30] J. He and C. Hoare. *Unified Theories of programming*. Prentice Hall International, 1998.

[31] A. Hejlsberg, S. Wiltamuth, and P. Golde. *The C# Programming Language*. Addison-Wesley Professional, 2006.

[32] C. Hoare and J. He. *Unifying theories of programming*, volume 14. Prentice Hall, 1998.

[33] S.-Y. Hong and T. Kim. A resolution converter for multi-resolution modeling/simulation on HLA/RTI. In K. Koyamada, S. Tamura, and O. Ono, editors, *Systems Modeling and Simulation*, pages 289–293. Springer Japan, 2007.

[34] S.-Y. Hong and T. G. Kim. Specification of multi-resolution modeling space for multi-resolution system simulation. *Simulation*, 89(1):28–40, 2013.

[35] IEEE. *IEEE standard for modeling and simulation (M&S) high-level architecture (HLA) Federate Interface Specification*. IEEE 1516.1-2000, IEEE-SA Standards Board, 2000.

[36] IEEE. *IEEE standard for modeling and simulation (M&S) high-level architecture (HLA) Framework and Rules*. IEEE 1516-2000, IEEE-SA Standards Board, 2000.

[37] IEEE. *IEEE standard for modeling and simulation (M&S) high-level architecture (HLA) Object Model Template (OMT) Specification*. IEEE 1516.2-2000, IEEE-SA Standards Board, 2000.

[38] H. Jifeng, X. Li, and Z. Liu. Component-based software engineering. In D. Hung and M. Wirsing, editors, *Theoretical Aspects of Computing – ICTAC 2005*, volume 3722 of *Lecture Notes in Computer Science*, pages 70–95. Springer Berlin Heidelberg, 2005.

[39] H. Jifeng, X. Li, and Z. Liu. rCOS: a refinement calculus of object systems. *Theoretical Computer Science*, 365(1-2):109–142, 2006.

[40] S.-S. T. Jongmans, F. Santini, M. Sargolzaei, F. Arbab, and H. Afsarmanesh. Automatic code generation for the orchestration of web services with reo. In *Service-Oriented and Cloud Computing*, pages 1–16. Springer, 2012.

[41] A. Kara, F. Deniz, D. Bozağaç, and M. N. Alpdemir. Simulation Modeling Architecture (SiMA), a DEVS Based Modeling and Simulation Framework. In *Proceedings of the 2009 Summer Computer Simulation Conference*, SCSC '09, pages 315–321, Istanbul, Turkey, 2009. Society for Modeling & Simulation International.

[42] S. Kasputis and H. Ng. Composable simulations. In *Simulation Conference Proceedings, 2000. Winter*, volume 2, 2000.

[43] K.-H. Kim and W.-S. Kang. Corba-based, multi-threaded distributed simulation of hierarchical devs models: transforming model structure into a non-hierarchical one. In *Computational Science and Its Applications–ICCSA 2004*, pages 167–176. Springer, 2004.

[44] W. D. McCarty, S. Sheasby, P. Amburn, M. R. Stytz, and C. Switzer. A virtual cockpit for a distributed interactive simulation. *Computer Graphics and Applications, IEEE*, 14(1):49–54, 1994.

[45] J. McEver, P. K. Davis, and J. Bigelow. Implementing multiresolution models and families of models: from entity-level simulation to desktop stochastic models and" repro" models. *Proceedings of Enabling Technology for Simulation Science IV, Orlando FL*, 2000.

[46] S. Meng and F. Arbab. Connectors as designs. *Electronic Notes in Theoretical Computer Science*, 255:119–135, 2009.

[47] D. Méry and N. K. Singh. Automatic code generation from Event-B models. In *Proceedings of the Second Symposium on Information and Communication Technology*, SoICT '11, pages 179–188, Hanoi, Vietnam, 2011. ACM.

[48] S. Mittal, B. P. Zeigler, and J. Martin. Implementation of formal standard for interoperability in m&s/systems of systems integration with devs/soa. *International Command and Control C2 Journal, Special Issue: Modeling and Simulation in Support of Network-Centric Approaches and Capabilities*, 3(1), 2009.

[49] M. Moallemi and G. Wainer. Designing an interface for real-time and embedded DEVS. In *Proceedings of the 2010 Spring Simulation Multiconference*, SpringSim '10, pages 137:1–137:8, Orlando, Florida, 2010. Society for Computer Simulation International.

[50] A. Natrajan, P. Reynolds, and S. Srinivasan. MRE: a flexible approach to multiresolution modeling. In *Parallel and Distributed Simulation, 1997., Proceedings., 11th Workshop on*, pages 156–163, Lockenhaus, 1997. IEEE.

[51] NRC. *Modeling and Simulation*, volume 9, pages 2000–3030. National Academy Press, Washington, D.C., 1997.

[52] E. H. Page and J. M. Opper. Observations on the complexity of composable simulation. In *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*, pages 553–560. ACM New York, NY, USA, 1999.

[53] Pitch Products. `http://www.pitch.se/products/products-overview`, 2013.

[54] D. R. Powell. Control of entity interactions in a hierarchical variable resolution simulation. In *Conference: Fall simulation interperability workshop, Orlando, FL (United States), 8 Sep 1997*, 1997.

[55] P. Reynolds Jr, A. Natrajan, and S. Srinivasan. Consistency maintenance in multiresolution simulation. *ACM Transactions on Modeling and Computer Simulation*, 7(3):392, 1997.

[56] H. S. Sarjoughian. Model composability. In *Proceedings of the 38th conference on Winter simulation*, page 158. Winter Simulation Conference, 2006.

[57] A. Sidi. *Practical extrapolation methods: Theory and applications*. Number 10. Cambridge University Press, 2003.

[58] N. Singh. *Reliability and Safety of Critical Device Software Systems*. PhD thesis, Université Henri Poincaré - Nancy, 2011.

[59] C. Szyperski, D. Gruntz, and S. Murer. *Component software: beyond object-oriented programming*. Addison-Wesley, 2002.

[60] The Eclipse Foundation. `http://www.eclipse.org`, 2013.

[61] G. A. Wainer, R. Madhoun, and K. Al-Zoubi. Distributed simulation of devs and cell-devs models in cd++ using web-services. *Simulation Modelling Practice and Theory*, 16(9):1266–1292, 2008.

[62] Ford E-Series. `https://en.wikipedia.org/wiki/Ford_E-Series`, 2013.

[63] XML. `http://www.w3.org/XML`, 2013.

[64] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008.

[65] B. Zeigler, T. G. Kim, and H. Praehofer. *Theory of Modeling and Simulation*. Academic Press, January 2000.

[66] B. P. Zeigler. *Theory of Modeling and Simulation*. John Wiley, 1976.

[67] M. Zhang, B. P. Zeigler, and P. Hammonds. Devs/rmi-an auto-adaptive and reconfigurable distributed simulation environment for engineering studies. In *Proceedings of the 2006 DEVS Integrative M&S Symposium (DEVS'06), Huntsville, AL*, 2006.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Kara, Ahmet
Nationality: Turkish (TC)
Date and Place of Birth: 1981, Eskişehir
Marital Status: Married
Email: ahmet.kara@tubitak.gov.tr

EDUCATION

Degree	Institution	Year of Graduation
M.S.	Bilkent University	2006
B.S.	Bilkent University	2003

PROFESSIONAL EXPERIENCE

Year	Place	Enrollment
2003-Present.	TÜBİTAK BİLGEM İLTAREN	Chief Researcher

PUBLICATIONS

1. F. Deniz, M. Alpdemir, A. Kara, and H. Oğuztüzün. Supporting dynamic simulations with simulation modeling architecture (sima): a discrete event system specification-based modeling and simulation framework. Simulation, 88(6):707–730, 2012.

2. D. Bozağaç, G. Karaduman, A. Kara, and M. Alpdemir. Sim-petek: A parallel simulation execution framework for grid environments. The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, 9(4):303–319, 2012.

3. A. Kara, F. Deniz, C. D. Bozağaç, and M. N. Alpdemir. Simulation Modeling Architecture (SiMA), A DEVS Based Modeling and Simulation Framework. In Proceedings of Summer Computer Simulation Conference (SCSC'09), pages 315–321. SCS, 2009.

4. D. Bozağaç, G. Karaduman, A. Kara, M. N. Alpdemir. Sim-PETEK : A Parallel Simulation Execution Framework for Grid Environments. In Proceedings of Summer Computer Simulation Conference (SCSC'09), pages 275 - 282. SCS, 2009.

5. F. Deniz, A. Kara, M. N. Alpdemir, H. Oğuztüzün. Variable Structure and Dynamism Extensions to SiMA, A DEVS Based Modeling and Simulation Framework. In Proceedings of Summer Computer Simulation Conference (SCSC'09), pages 117 - 124. SCS, 2009.

6. A. Kara, D. Bozağaç, M. N. Alpdemir. Simülasyon Modelleme Altyapısı (SiMA) : DEVS Tabanlı Hiyerarşik ve Modüler Bir Modelleme ve Koşum Altyapısı. In Proceedings of İkinci Ulusal Savunma Uygulamaları Modelleme ve Simülasyon Konferansı (USMOS 2007), pages 271 - 281. 2007