NTNU

Innovation and Creativity

Multi-Formalism Modelling of a Submarine Combat System Test Facility: an Application of DEVS

Kjell-Inge Skogstad

Master of Science in Computer Science

Submission date:	June 2006
Supervisor:	Peter Hughes, IDI
Co-supervisor:	Robert Macdonald, FFI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

FFI is developing a test facility for submarine combat systems, represented by a combination of real-world and simulated units, which is to be used for concept development and later acquisition of military technology. Recent developments in the theory of modelling and simulation seek to provide a rigorous basis for this purpose and particularly the DEVS-based theory of modelling and simulation (Zeigler et al. 2000) has been developed with respect to a wide context of parallel and distributed simulation, continuous and discrete paradigms and multiple modelling formalisms.

The proposal has three parts, which build on one another:
(i) apply the general concepts of DEVS to produce a generic architectural description of a representative subset of the submarine combat system at the level of a network of coupled components establishing: types of components and formalisms to be used, influencer and influencee sets and the nature of the output/input couplings including real-time constraints.
(ii) use the above generic description to identify any special requirements and operating parameters needed for a practical example of the system, taking account of the existence of legacy components and available simulation technologies.
(iii) explore how far the DEVS-based theory is useful for analysing such combat systems and in helping to establish their credibility.

Assignment given: 20. January 2006
Supervisor: Peter Hughes, IDI

Abstract

This thesis aims at applying and exploring the DEVS-based theory for the purpose of gaining experiences and recommendations with regards to the usefulness of DEVS in the analysis of a submarine combat system and in establishing simulation credibility.

In this regard, first a literature study of the DEVS-based literature is performed, before a case study is carried out, targeting a subset of the submarine combat system test bed under construction at Forsvarets forskningsinstitutt (FFI).

In doing so, an architectural description of the subset based on DEVS is created and special requirements are discussed with regards to legacy components and technologies. Finally, the usefulness of DEVS is discussed based on experiences made during the first two tasks. Note that implementation and aspects with regards to an executable framework for the simulator is not covered.

The findings of this study indicates that DEVS with its formal nature can be a valuable tool both with regards to analysis as well as in establishing credibility. Especially with regards to couplings and composability DEVS might prove helpful. The formal specifications and definitions can ensure that consistency is achieved. A formal experimental frame also ensures an unambiguous foundation for creating the simulation.

With this in mind, an issue was discovered with regards to how far DEVS should go in covering aspects which can be argued to be part of the simulator within the abstract model. A solution involving the use of one or more lumped models is suggested, but need further study.

Finally, the need for a proper tools and a graph notation is emphasised if DEVS is to be practical in complex simulations.

Keywords: *DEVS*, *Methodology*, *Multi-Formalism*, *Modelling*, *Simulation*, *Submarine Combat System.*

Preface

This thesis has been written by Kjell-Inge Skogstad as part of the *Master's Thesis, Technology Programme* at the *Norwegian University of Science and Technology (NTNU), Department of Computer and Information Science (IDI)*, during the spring semester 2006.

The thesis is also a result of a collaboration with Forsvarets forskningsinstitutt (FFI) [1] and the TEKULA project carried out there, aiming at simulating a submarine combat system for the purpose of concept development and later military acquisition.

The intention of this work is to study the DEVS methodology through application and exploration of its usefulness with regards to analysis and establishing the credibility of complex simulations.

In this regard I would like to thank my teaching supervisor, Professor Peter Hughes, as well as my supervisor at FFI, Robert Macdonald, for their support and constructive criticism throughout the period. In addition I would also like to thank Magne Mandt and Tormod Linnerud at FFI for their valuable input and feedback on submarine combat systems.

Trondheim, June 2006

Kjell-Inge Skogstad

[1] Norwegian Defence Research Establishment

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1
Introduction

This chapter provides an introduction to the thesis and the motivation behind it. Further, the problem definition and context as well as goals and scope are listed. At the end of the chapter, an outline for the report is provided.

1.1 Motivation

Experimentation and exploration of different kinds of systems can often be a difficult and costly task. If the system to be experimented with is not accessible, it even becomes impossible. Simulation can in this context provide an alternative by mimicking the behaviour of the system of interest.

Due to its great potential, computer simulation has received much attention in later years. For many years however, the ability to simulate complex systems has been limited due to the processing power of computers. Now, with the advance of distributed computing, this limitation seems to be relaxed. Linking computers together, each machine computing only a subset of the global simulation, allows for larger and more complex simulations to be performed.

With the increasing complexity follows increasing attention to the analysis and credibility of the simulations. A simulator which does not sufficiently represent what it is suppose to represent, has little or no value. Attempts are also made towards including real hardware and human interaction in the simulations, increasing the complexity further.

This argues for the need of efficient and well-grounded methods for analysing the systems to be simulated and for establishing the credibility needed. Several studies of the topic have been carried out in recent years, but much is still to be done. This thesis aims to further add to this research and will hopefully provide further answers towards a recommended methodology for modelling and simulation.

1.2 Problem Definition and Goals

The assignment text is given below.

FFI is developing a test facility for submarine combat systems, represented by a combination of real-world and simulated units, which is to be used for concept development and later acquisition of military technology.

Recent developments in the theory of modelling and simulation seek to provide a rigorous basis for this purpose and particularly the DEVS-based theory of modelling and simulation (Zeigler et al. 2000) has been developed with respect to a wide *context of parallel and distributed simulation, continuous and discrete paradigms and multiple modelling formalisms.*

The proposal has three parts, which build on one another:

(i) apply the general concepts of DEVS to produce a generic architectural description of a representative subset of the submarine combat system at the level of a network of coupled components establishing: types of components and formalisms to be used, influencer and influencee sets and the nature of the output/input couplings including real-time constraints.

(ii) use the above generic description to identify any special requirements and operating parameters needed for a practical example of the system, taking account of the existence of legacy components and available simulation technologies.

(iii) explore how far the DEVS-based theory is useful for analysing such combat systems and in helping to establish their credibility.

As can be seen here, the text identifies three distinct parts. The first involves creating a description of a representative subset of the specified system using the concepts from the DEVS-based theory. Secondly, special requirements of this system, for example limitations in value ranges or incompatibility between simulators and real hardware, as well as operating parameters shall be identified. This second task should reflect upon the existence of legacy components and technologies.

The final task is related to using the DEVS-based theory in analysing the system and establishing the credibility of the simulations. Here, experiences based on the previous tasks as well as related topics described in literature, will form a basis for evaluating and exploring the contribution of the theory for this purpose.

With this is mind, the main goal of the thesis is to apply and explore the usefulness of the DEVS-based theory for the purpose described above. Based on the assignment text, sub goals can be listed as follows:

- Identify relevant DEVS concepts
- Create an architectural description using the DEVS-based theory
- Identify operating parameters and special requirements
- Explore usefulness with regards to analysis and credibility
- Identify areas of further study

Note that [38] will be used as a starting point for this study.

1.3 Scope

The thesis is limited to studying modelling and simulation, with respect to the DEVS-based theory. Other theories and standards might exits, but will not be covered.

Further, this thesis will focus on using the DEVS-based theory in the analysis of the specified system, with special interest in the basic formalisms. Alternative extensions and reductions of these formalisms will not be covered unless special need for these are demonstrated in the TEKULA project. Although the DEVS-based theory also covers a framework for simulator implementation, this is seen as outside the scope of this thesis.

With regards to the TEKULA project, only a subset of the system to be simulated will be analysed here. As this thesis will focus on the usefulness of the DEVS-based theory, analysing a complete system may not be necessary. This is however based on the assumption that the subset chosen is representative, with respect to types of components and the couplings between them. Note that functionality need not be taken into account.

Finally, this thesis will not cover the final implementation of the analysed system. However, issues with regards to simulators and simulation credibility will be discussed with regards to task 2 and 3.

1.4 Thesis Context

This thesis is carried out in collaboration with the TEKULA project at the Forsvarets forskningsinstitutt (FFI) which aims at building a test bed for a submarine combat system. The test bed will consist of both simulated and real world components, and will also facilitate a synthetic environment. Further, the system will allow human interaction with some of the components.

The system is intended to be used with regards to concept development and experimentation with new functionality for the purpose of providing recommendations to later acquisition of military technology.

To allow for the experimentation capabilities sought for, the system needs to be modular and allow for new, not yet developed, components to be added with minimal impact on the remaining system. Further, it is also decided that distributed middleware shall be applied in the simulator. Both the High Level Architecture (HLA) and the Common Object Request Broker Architecture (CORBA) has been suggested for this purpose.

Complexity issues with such an initiative argue for the use of well-founded methods and tools to be employed through all of the phases of the project. In this context the DEVS-based theory provides a complete theory on modelling and simulation. In addition, later studies have also shown how DEVS can be applied together with HLA. Further exploration of the benefits and drawbacks of using the DEVS-based theory is therefore of great interest for the project.

1.5 Report Outline

This report is divided into 5 parts. First, an introduction to the project and the domain is presented in Part I - Introduction. Secondly, Part II - State-of-the-art, presents a review of related topics found in the literature. Further, Part III - Own contribution, presents the work and findings of this thesis, and Part IV presents the conclusion and thoughts of further work. The final part, Part V, contains relevant appendixes.

A short description of the different chapters is provided next for ease of reading.

Chapter 1 – Introduction
This chapter presents an introduction to the thesis, including the problem definitions, goals, scope and context.

Chapter 2 – Background
The background chapter presents an introduction to modelling and simulation, known challenges and basic concepts used in this thesis.

Chapter 3 – Research Method
The chapter presents the research method used in this thesis.

Chapter 4 – Modelling and Simulation Framework
Here, a framework for modelling and simulation is presented, establishing the concepts needed for the further study of the DEVS-based theory.

Chapter 5 – Formalisms
This chapter presents and discusses relevant formalisms which can be used in describing a system.

Chapter 6 – Validation and Verification
This chapter presents an review of topics related to verification, validation and simulation credibility found in literature.

Chapter 7 – Submarine Combat System
A general overview of the submarine combat system is given in this chapter and a subset of this system is chosen for further study. Special aspects with regards to the simulation of this system is also described.

Chapter 8 – Model Description
Here, an architectural description of the chosen subset of the submarine combat system based on DEVS-based theory is presented.

Chapter 9 – Requirements for Model Simulation
Legacy components and technology are here introduced in a step towards actual simulators. Operating parameters and special requirements found in this analysis are also discussed.

Chapter 10 – Analysis and Simulation Credibility
Finally, a discussion with recommendations, with regard to using DEVS in the analysis of such a system is given also taking into account the simulation credibility.

Chapter 11 – Conclusion
This chapter will sum up the findings of this work and present a final conclusion with regards to using the DEVS-based theory.

Chapter 12 – Further Work
Finally, thoughts of possible further work and associated research directions will be outlined in this chapter.

Chapter 2 Background

This chapter presents the background for this thesis. First, modelling and simulation will be discussed in general, before areas of usage and known challenges are presented. At the end of this chapter basic concepts are listed for ease of reading.

2.1 Modelling and Simulation

Simulation can be said to be the mimicking of the behaviour of a system by a different, often simplified, system. Through simulators, virtual representations of the systems that need to be studied can be created, allowing experimentation and exploration that perhaps was previously impossible. For example, it may be far easier to experiment with a simulated submarine than a real one.

In this regard, a system can be almost anything from a single object or component to a complex system consisting of many smaller systems. Examples here are, computer systems, traffic systems, weather systems and so on.

The creation of simulators however demands a proper analysis of the system in question, and a model that accurately produces a similar behaviour needs to be created. In this task modelling methodologies play an important part, as they define how the system and model are described.

Recently, much attention has been made towards distributed simulations. The reason for this is that it allows for more complex simulations to be achieved. With distributed computation the calculations of a simulation can be spread to multiple computers, implying that the computational limitations are relaxed, as well as the components of the simulator can be geographically dispersed.

Hardware or human in the loop is also a topic of recent research. Introducing real hardware components in the simulations can for example increase the credibility of the simulations. Real hardware produces real data to the system, which can be seen as error-free with regards to simulation.

Introducing humans in the simulations also allow for a great spectrum of new possibilities. Synthetic environment is often mentioned in this context. A synthetic environment is a simulated reality. For example, is it possible to simulate a car in a simulated world, and let a real human control the vehicle.

In this regard, the DEVS-based theory provides a complete theory on the domain, including among other things; a conceptual framework, a modelling methodology and a simulation framework. Note that the latter will not be studied here.

2.2 Areas of Usage

Computer simulations can be and are used in a number of areas. Some of these will be presented next. Note that this is not a complete list of areas of usage, but some specific areas are described for the purpose of showing some of the potential of computer simulation.

Experimentation

Experimentation with systems and new concepts is one possible area of usage. Experimenting with the real system may not be possible or the cost of doing so could be too high. For example, studying a torpedo's movement and detonation against a boat side might be extremely limited in the real world. However, if credible simulations of the objects were to be created, the study might still be possible. Another example is studying pollution in the atmosphere. A simulation may here allow for more experimentation than otherwise possible.

The goal of this form of experimentation if often to gain new knowledge about a system or to prove that a concept is valid. This is often achieved by experimenting with different parameters and combinations of components or functionality.

An example of such a project can be found in [9] where developing tactics for sea warfare are discussed with regards to simulation.

Simulation-Based Acquisition

Simulation-based acquisition (SBA) is close to experimentation. Also here, parameters, functionality and structure between components are experimented with. However, here the goal is different. The goal of these simulations are to prove concepts and show functionality for the purpose of later acquisition of systems or components.

Often in SBA, subsets of the system to be analysed are not yet available or perhaps do not even exist. For example if new torpedoes are to be bought, simulations of the various alternatives might shed light on their capabilities and provide important information to the buying process.

More on SBA can be found in for example [29].

Training

Training is another area of usage. With simulated systems it is possible to create a simulated world, a synthetic environment, where for example pilots may practice manoeuvring virtual airplanes.

This usage has close resemblance to some products in the computer game industry. It is however currently unknown how well these products interact with greater simulations and military requirements.

Demonstrator

Simulators can also be used as demonstrators. By for example providing a visualisation of the simulation in a 3D-world, the concepts or situations studies may be communicated to others in a fashion that is more convincing than that of showing some results on paper.

Note that the tools used in this scenario are often also used in experimentation, in simulation-based acquisition and in training.

2.3 Known Challenges

Many challenges still remain within the modelling and simulation domain. In the literature, topics related to hardware support for simulations, standards, development time, architectural questions, representation of human factors and virtual reality are often mentioned [19] [24].

Focusing at analysis and credibility, several challenges can also be listed. Below, some of these are discussed briefly.

2.3.1 Methodology

The analysis of a given system should be in compliance with a given methodology. In the context of the modelling task, a methodology includes both the way at which the system is described as well as the process showing how this should be done.

Several challenges in modelling methodology are described in [24]. Among other things, it is stated that a methodology should support:

- Representation of business applications
- Representation of human factors
- Representation of geographical dispersed networks
- Rapid model development
- Flexible model maintenance

Standards are also stressed in this regard [24]. The High Level Architecture (HLA) standard is often mentioned, standardizing the interface in distribute simulations [40]. In connection to HLA, also a development process, FEDEP, is defined, providing a recommended process for the development of the simulator. It can however be argued that HLA focuses on interoperability at a low level and does not sufficiently specify the higher level dynamics involved [26]. It is also stressed in [5], a study covering composability of DoD Models and Simulations, that it is time for the Department of Defense (DoD), the creators of HLA, to revisit the standards. Especially the need for separating models from general software is mentioned.

UML is also often mentioned. However, UML is partly built on informal concepts [39] and does only provide with a range of modelling notations to be used in analysing the system.

For this reason, the choice of methodology is still unclear.

2.3.2 Complexity

Complexity is also a key challenge with respect to analysis and credibility [3]. The more complex the system is, the more difficult the analysis and the establishment of the credibility will be.

Controlling complexity can be achieved through controlling the level of detail and the scope of the simulation [3]. However, these cannot be controlled freely. The level of detail is closely related to accuracy and often high accuracy in the system implies a high level of detail. On the other hand, it has also been shown that too high level of detail reduce the confidence in the system, as the chance of error increases [3].

Similarly, the scope is often determined by the objectives of the simulation, arguing for thorough specification of what the intended use of the simulator is. Including everything may not be needed to accomplish the task specified.

Modularity is a tool which can help in this regard [21]. A modular approach allows for a divide and conquer strategy to be followed, further allowing parts of the total system to be studied and specified independently of the rest. This may greatly influence the difficulties in dealing with complexity and should therefore be a key aspect of the tasks related to both analysis and the establishment of the credibility.

[20] presents these five principles which should be kept in mind throughout the modelling and simulation process. The three first of these, which speak for themselves, all deal with complexity.

- Model simple, think complicated
- Be parsimonious, start small and add
- Divide and conquer, avoid mega-models

2.3.3 Credibility

A final challenge often mentioned, is related to the simulation credibility. Credibility and confidence is a necessity when dealing with simulations. If there is little confidence in the results provided by the simulator, or the results are not credible, then the simulation has little value to the user.

For this reason, much attention must be given to establishing the correctness of the simulation. However, issues arise with regards to how this should be accomplished. For example, has the real world system been studied thoroughly and a representative model of it been created? Does the simulator accurately reflect this model? Is it possible to prove that the simulator provides the correct output given any possible scenario?

Testing and formal proofs are often mentioned in this regard. By testing one can show that the simulator is correct given a particular scenario. However, is it possible to test all possible scenarios? Even within a limited scope this might be difficult.

Formal proofs on the other hand requires that the behaviour studied is specified in a formal manner. Can one guarantee that the specification of the behaviour is without error? Further, as the simulator grows and becomes more complex, will the formal proves become unfeasible?

One can argue that there is no correct way of establishing the credibility needed and hence also this topic needs to be studied further.

2.4 Basic Concepts

For ease of reading, some of the basic concepts used throughout this thesis are explained in Table 2.1.

Table 2.1: Basic Concepts

Concept	**Description**
Modelling	An activity where a model of a given system is created
Simulation	A simulation is an activity where models are executed for a given purpose [19]
Source system	The real system to be simulated
Experimental frame	The conditions under which the system shall be studied
Model	An abstract representation of the source system
Lumped Model	Usually, a simplification of a model. Here, also used about a model which reflects some of the simulator aspects
Simulator	An entity capable of executing the behaviour described in a model
Formalism	Formal specification of a system
DEVS	The Discrete Event System Specification (DEVS) is a particular formalism. However DEVS will also here refer to the methodology described in the DEVS-based literature
Verification	Refers to whether a simulator correctly executes a model or not.
Validation	Refers to how well a model represents the source system given the experimental frame
Morphism	A morphism is a mapping between two structures
Homomorphism	A morphism is a one way mapping between two structures
Isomorphism	A morphism is a two way, one-to-one and onto, mapping between two structures
Trajectory	A trajectory is a function identifying a value in a given set by time, also called time function or signal
Segment	A segment is a restriction of a trajectory within a given interval
Time Base	A time base is set of elements with a given ordering. Examples are discrete and continuous time base
Composability	Refers to the capability to couple components together
Distributed Middleware	Software allowing message passing between a collection of computers. High Level Architecture (HLA) and Common Object Request Broker Architecture (CORBA) are examples of such software

Chapter 3
Research Method

This thesis will be based on a combined literature and case study, see Figure 3.1.

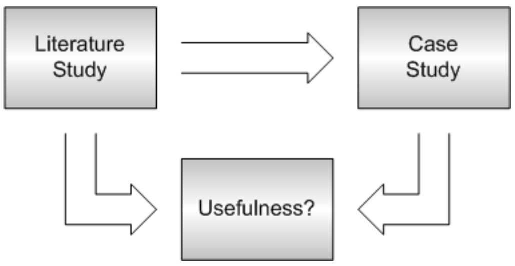


Figure 3.1: Research Method

First, a study of recent relevant literature will be conducted, with focus on the DEVS methodology and verification and validation. The book of Zeigler [38] will here form a natural starting point of this study, as well as relevant articles from authorities and oft cited authors will be used as sources of additional knowledge of the domain as well as providing a more complete foundation for the study. Note however that some literature on simulation that could be of relevance may not be available due to security classification. The most relevant results of the literature study are presented in part II, Chapter 4, Chapter 5 and Chapter 6.

The literature study will further form the basis for the case study of the submarine combat system to be simulated at FFI and the usefulness of the DEVS-based theory in this regards. An attempt will be conducted to use DEVS in creating an architectural description of a subset of this system. Further, special simulation requirements with regards to legacy components and technologies will be explored. Results of this second study is given in Chapter 7, Chapter 8 and Chapter 9.

Based on the experience gained from both studies, the usefulness of the DEVS-based theory will finally be explored with regards to analysis and simulation credibility. Findings of this exploration are presented in Chapter 10.

Part II

State-of-the-art

Chapter 4
Modelling and Simulation Framework

A unified framework is needed to provide a common ground on which discussion and understanding can be achieved. A list of challenges that such a framework for the modelling and simulation domain will need to cover, is cited in [41] and further listed below.

- Discontinuous nature of discrete events
- Continuous nature of most performance measures
- Importance of probabilistic formulation
- Need for hierarchical analysis
- Presence of dynamics
- Feasibility of the computational burden

Previous approaches towards a unified framework in modelling and simulation have not been able to cover all of these challenges. The DEVS-based theory however was created explicitly for this purpose and in [38] Zeigler presents a general framework describing the basic entities and relations between these.

This framework is based upon systems theory to adopt the formal correctness and the sound semantics needed. A short overview of systems theory will therefore be provided, before the framework is introduced.

In [27], a refinement of this framework is presented. This refined view will also be shortly presented before a final discussion about the framework and its applicability will be given at the end of this chapter.

4.1 Systems Theory

Systems theory describes how dynamic systems can be defined through sound mathematical formalisms [39]. In doing so, the theory differs between two orthogonal aspects; *levels of system specification* and *systems specification formalisms.*

Levels of system specification refer to the different levels at which you can describe a system given by how much knowledge you have about it. Table 4.1 presents an overview of the these levels.

For further information about the different levels, please refer to [38]. Note however that structured versions of level 0 - 3 can also be constructed. Structured level 4 is equal to level 5.

Further, system specification formalisms are types of modelling styles or structures. The DEVS formalism, described in Chapter 5 is an example of such a formalism.

Table 4.1: Hierarchy of System Specifications

Level	Specification	Description
0	I/O Frame	Describes time base and input/output value sets)
1	I/O Relation Observation	Describes all possible input/output segment pairs
2	I/O Function Observation	Transform the relation to a set of functions
3	I/O System	Adds the systems internal states, global state transition and output function
4	Iterative Specification	Replaces the global state transition function with iterative state transitions
5	Structured System Specification	All sets are represented as multivariable sets
6	Non-Modular Coupled Multi-Component System	Introduce coupled systems where the components influence each other directly through their state transition functions
7	Modular Coupled Network of Systems	Introduces modularity by restricting influence to input and output interfaces

Usually in this context a system is visualised as a box with input and output, see Figure 4.1.

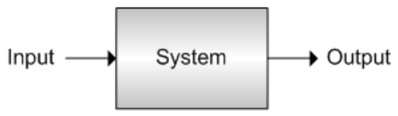


Figure 4.1: Input/Output System

While the input and output of the system defines the external *behaviour* of the system, the inner workings of the system, the *structure*, is often described through functions or states and state-to-output functions. Note that it is this inner structure that is known at the higher levels of specification.

Further, a system can be decomposed into several sub-systems as shown in Figure 4.2.

As can be seen here, the system is decomposed into three sub-systems. An important property of systems theory is that the system is *closed under coupling* [38]. This implies that the composed system, the *resultant*, can be represented in the terms of the same system theory as the sub-systems. For example, the resultant of coupling two DEVS systems can itself be expressed as a DEVS system. This guarantee that well-defined structure and behaviour is kept and that the coupled system itself might be part of a larger system.

Modularity is achieved by using systems where all interaction with the environment is done through predefined input and output ports [21].

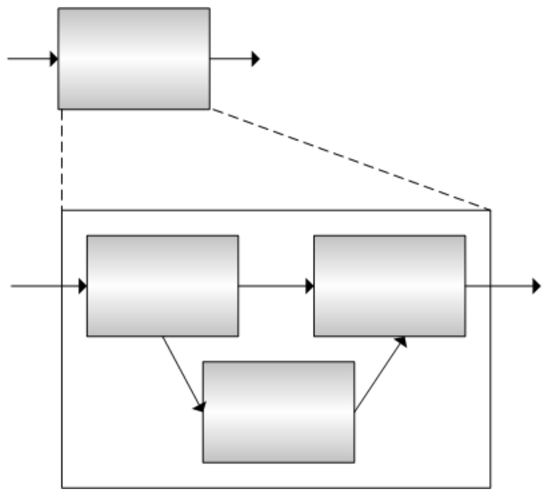


Figure 4.2: Hierarchical System Decomposition

4.2 Original Framework

As shown, system theory provides a sound and rigorous foundation for building a framework, allowing formal specified entities and relations to be adopted. Figure 4.3 presents an overview of the main concepts in the modelling and simulation framework as presented in [38].

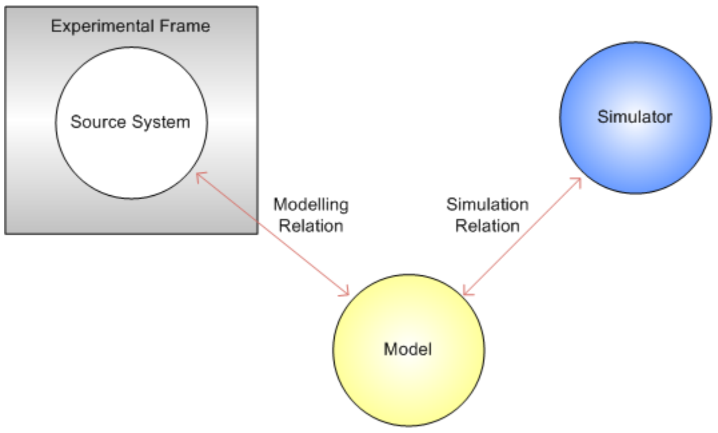


Figure 4.3: Basic Entities and Relations

As can be seen, the basic entities of this framework are *the source system, the experimental frame, the model* and *the simulator*. Further, the most important relations are *the modelling relation* and *the simulation relation.*

4.2.1 Entities

In the following, all of the entities mentioned above will be described in more detail. Note that all entities can be visualised as systems and formally described through formalisms.

Source System

The source system is the system that is to be modelled and later simulated. Often little is known about the internal structure of this system and the system can only be studied through its observable data. This data can be represented as input/output pairs of time segments [35], resulting in specification of the system usually at level 1 or less.

In dealing with man-made systems however, for example a computer system, more of the structure may be known in advance, allowing higher levels of specification to be achieved. As the structure of the system is known here, choosing a similar structure in the simulator may result in increased credibility. It is however important to remember that the goal here is to produce a simulator where the behaviour is equal to the system of interest and not necessarily a copy of it.

Experimental Frame

The experimental frame is defined to refer to the conditions under which the system is observed and experimented with [35] [22]. More precisely, the experimental frame should include at least the following [35]:

- **Input Stimuli**
  Specification of the legal input trajectories.
- **Control**
  Specification of the conditions under which the system will be initialized, continued under simulation and terminated.
- **Metrics**
  Specification of the data summarization functions and measures for providing quantitative or qualitative analysis of the behaviour. Note that in a composed system, the metrics include all output from the component.
- **Analysis**
  Specification of the means for analysis of the collected data.

As systems seldom can be run without some initiating control system connected to it, it can be argued that the experimental frame can be realized as such a system [34]. In this interpretation of the experimental frame the experimental frame can be envisaged as shown in Figure 4.4.

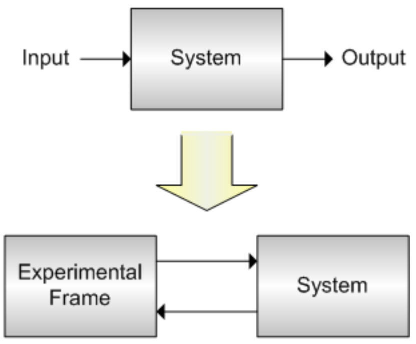


Figure 4.4: Input/Output System and the Experimental Frame

Note the duality of the model and the experimental frame system in this figure. Both are systems within the problem studied.

Finally, as the experimental frame specifies the conditions under which the simulator shall work, the experimental frame must reflect upon the simulation objectives. Given different simulation scenarios it is also possible that a system to be associated with multiple experimental frames depending on what is being studied.

Note that sub-systems also may have their own experimental frame as these are systems in their own right.

Model

The model is an abstract representation of the source system, seen in the context of the experimental frame, on which the final simulator shall be built. In contrast to the source system, a model in modelling and simulation is usually specified at a higher level of specification. The reason for this is of course that here the instructions for generating the output, the structure, will need to be made explicit.

Note however, that this is often accomplished based on assumptions about the structure of the source system.

Simulator

A simulator is an agent capable of generating the behaviour specified by a model. That is, it is either an implementation of the model or a simulator engine capable of executing the model's behaviour. Figure 4.5 shows the two options.

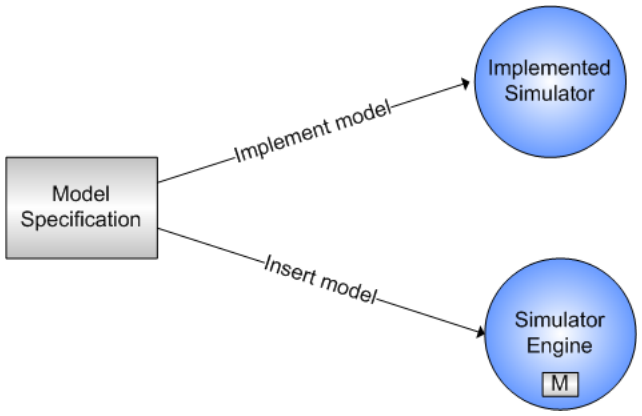


Figure 4.5: From Model to Simulator

Both approaches towards creating simulators are possible. However, as mentioned in [21] using an abstract simulator engine capable of simulating a range of models, has many benefits. For example, it can be argued to be much faster to modify or create new models given this solution, than hard-coding all simulators from scratch.

Similarly to the source system, this entity is a real world entity. However, since this entity is man-made, the structure of the simulator is known. For this reason it might be possible to specify also this component at a higher level of specification.

4.2.2 Relations

The two primary relations in this framework are the modelling and simulation relations.

The modelling relation is the relation between the source system, the experimental frame and the model, see Figure 4.3. This relation deals with how well the model represents the source system, where the experimental frame defines the conditions under which this comparison should be evaluated.

The separation of model and simulator is further an important feature of this framework. The relation between these entities is called the simulation relation, see Figure 4.3. This relation deals with whether the simulator simulates the model correctly or not. [38] indicates that for this to be true, the model and simulator have to at least be in agreement at level 2 in the specification hierarchy.

Both these relations are closely related to validation, verification and simulation credibility and will therefore be addressed further in Chapter 6.

In addition to these relations, there can be argued to be a relation between two collaborating systems, for example, the experimental frame and the model. This relation is often referred to as the applicability or accommodation relation [38]. That is, the frame must be applicable to the model and the model needs to accommodate the frame.

4.3 Refined Framework

Although the framework described above is well accepted in the modelling and simulation community, [27] presents a refinement which can be worth mentioning. Figure 4.6, adopted from [27], presents this alternative which is motivated by the need to also formally capture the context under which a system is studied.

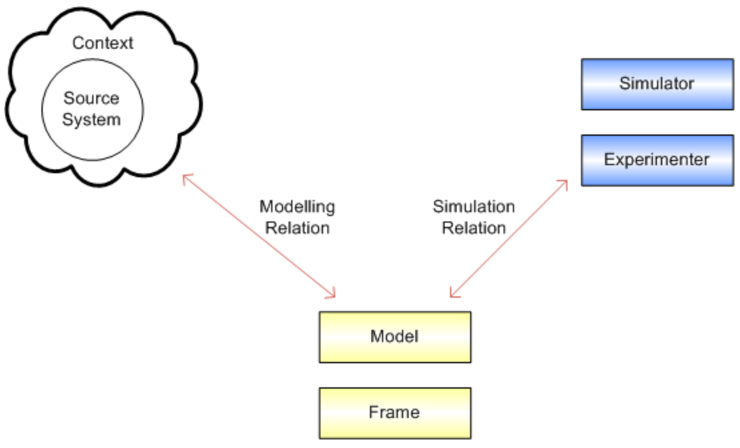


Figure 4.6: Alternative Framework

As can be seen here, this framework resembles the previous one. However, the duality between the experimental frame and the model is made explicit. [27] argues that the simulation of the system alone is seldom useful without an additional system which initiates and interacts with it. For this reason, it can be argued that this additional system should play a part in the development process along with the study of the system of interest, resulting in a formal abstract representation of the context in addition to the abstract representation of the system.

For this reason the Frame and Experimenter entities are defined with similar roles to that of the Model and the Simulator, but representing the context of the study.

4.4 Discussion

The basic entities and relations of the modelling and simulation framework depicted in [38] are presented in this chapter to provide a basis for the further study of DEVS.

The framework separates between the source system, the experimental frame, the model and the simulator. The separation of source system and model is perhaps trivial, but the separation of model and simulator provides a separation of concerns which ultimately might make the simulation more credible. For example, models can be created without any concern of how they later shall be implemented and abstract simulator engines can be created once, capable of simulating a range of model classes. Further, the framework is built upon a system theoretic foundation allowing modularity and ease of reuse.

The experimental frame has a special position in this framework, defining what shall be simulated and how it shall be analysed. For this reason, as well as the need to create a system which interacts with the simulator, the refined framework tries to formalize this entity.

It is however not given that one always wants to create an experimenter as suggested in the refined framework. The system of interest, might be a subset of a larger system to be simulated and is not meant to be executed alone. However, it can be argued that an experimenter could still be useful with regards to validating this component.

Finally, as mentioned earlier all entities can be specified by formalisms at a given level providing with an unambiguous description which can be studied with abstract tools. An overview of the usual levels of specification related to each entity is given in Table 4.2.

Table 4.2: Entities and Levels of Specification

Entity	**Level**
Source System and Context	0 - 1
Model and Frame	5 - 7
Simulator and Experimenter	7

Development Process

There is currently no formal development process defined specially for DEVS. However, similarly to UML, the DEVS-based theory can be applied in a range of situations as a modelling technique as well as with respect to validation and verification.

It can however be argued that the framework lends itself towards a process including the following steps:

1. **Identify Objectives**
   Identify simulation objectives.
2. **Analyse Source System and Objectives**
   Create specifications of the source system and context at level 0-1 based on observations of the real system and with regards to the objectives.
3. **Create Model and Frame**
   Create the model and frame specifications at level 5-7.
4. **Validate Model and Frame**
   Validate the model and frame with respect to their level 0-1 specifications.
5. **Create Simulator and Experimenter**
   Develop simulator and experimenter as well as create their specifications based on observations of their execution.
6. **Verify Simulator and Experimenter**
   Verify the simulator and experimenter with respect to the model and frame.
7. **Execute simulations**
   Execute simulations as planned.

Some studies have also suggested uses of DEVS with regards to the FEDEP development process, see [39] and [23]. Although adopting FEDEP in the TEKULA project would perhaps be reasonable, studying this option would be a too large undertaking in this thesis as it would require following the whole project life cycle. Studying issues with regards to a recommended development process is therefore proposed as possible further work.

Chapter 5 Formalisms

Formalisms can be used to describe the behaviour and structure of systems [35]. This chapter will present some of the formalisms found in the literature which are the most relevant to the work done in this thesis.

A formalism can usually only describe a subset of all possible systems. Often, this separation is done with regards to the system's behaviour in relation with time. In this context, three paradigms for handling behaviour has been depicted. These are associated with *discrete event systems, discrete time systems* and *continuous time systems* [38].

Figure 5.1, adopted from [41], shows how the formalism associated with these systems relate to one another.

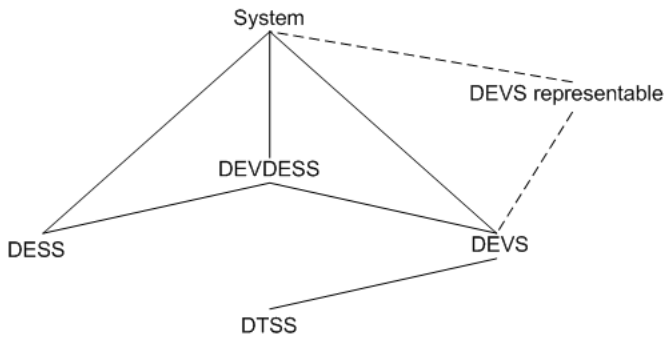


Figure 5.1: System Classes and Formalisms

In the figure, DESS is used to specify continuous time systems, DTSS is used to specify discrete time systems and DEVS is used to specify discrete event systems. Further, DEVDESS can be used to specify systems which have both continuous and discrete time behaviour.

The theory also differs between atomic and coupled formalisms. Atomic formalisms define the behaviour of the system by itself, while coupled formalisms couples together two or more formalisms.

In the following, the formalisms mentioned above, as well as other relevant formalisms, will be described in more detail. Note that this thesis focuses on modular descriptions. For this reason, only the coupled multi-component formalisms are introduced. For more information about non-modular systems see [38]. Note also that much of the information presented in this chapter is based upon Zeigler's work in [38].

5.1 Basic Formalisms

As previously mentioned the three basic types of systems are: discrete event systems, discrete time systems and continuous time systems. In the following, these systems and their associated formalisms, both atomic and coupled, will be described in more detail.

5.1.1 Discrete Event System Specification

Discrete event systems are systems where the time is driven forward by events. For example, when new input are presented on the systems input ports, these represents an event in time. The system now processes this event and perhaps creates some output after a given time which again will indicate an event.

As can be seen here, the time is defined by the events that occur. All events are associated with a specific time and as they are processed, time elapses in discrete steps. Note however that the events do not follow a specific frequency, but can occur after a random time interval.

The discrete event system's nature is often identified by its piecewise constant input and output trajectories [41]. Figure 5.2 shows how the input and output can change over time.

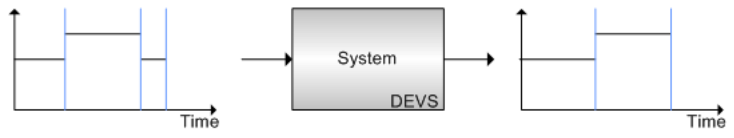


Figure 5.2: DEVS Input and Output

As can be seen here, the ports' values change in discrete steps without a fixed frequency. All instantaneous changes here indicate an event in time.

The formalism for creating an atomic model of discrete event systems is the Discrete Event Specified System (DEVS) formalism. A basic DEVS specification consists of the following:

$DEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$

X is the set of input events
Y is the set of output events
S is the set of sequential states
$\delta_{ext} : Q \times X \rightarrow S$ *is the external state transition function*
$\delta_{int} : S \rightarrow S$ *is the internal state transition function*
$\lambda : S \rightarrow Y$ *is the output function*
$ta : S \rightarrow \Re_0^+ \cup \infty$ *is the time advance function*
($Q = (s, e)s| \in S, 0 = e = ta(s)$ *is the set of total states)*

As shown, the behaviour is defined through sets of legal input and output events. Further, the dynamics of the system is specified through the set of states and two functions for moving between these. The external state transition function deals with external events. That is, this function is applied when a new input event occurs. The internal state transition function is applied when a state is naturally completed, given by the time advance function, and no external events has happened [1].

The output function further specifies the output given at each state and the time advance function specifies the duration of the event processing, that is, the maximum duration of the state. Note that output is only computed after an internal state transition.

A system including a number of DEVS models can be created by using the Discrete Event Specified Network Formalism (DEVN). This structure is given by:

$$DEVN = (X, Y, D, \{M_d\}, EIC, EOC, IC, Select)$$

X is the set of input events
Y is the set of output events
D is the set of component references
$\{M_d\}$ is the set of DEVS models, where $d \in D$
EIC is the set of external input couplings
EOC is the set of external output couplings
IC is the set of internal couplings
$Select : 2^D \rightarrow D$, where $Select(E) \in E$, the tie-breaking function

Here, EIC indicates the couplings between the external inputs to the internal models, EOC indicates the couplings from the internal models to the external output and IC the couplings between the internal models, see Figure 5.3.

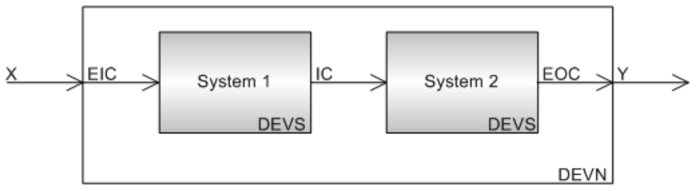


Figure 5.3: Example, DEVN Formalism

Finally, the Select function is used to select the imminent component, that is, the component that is scheduled to be activated next [2].

Note that except for the tie-breaking function, no behaviour is specified in this formalism.

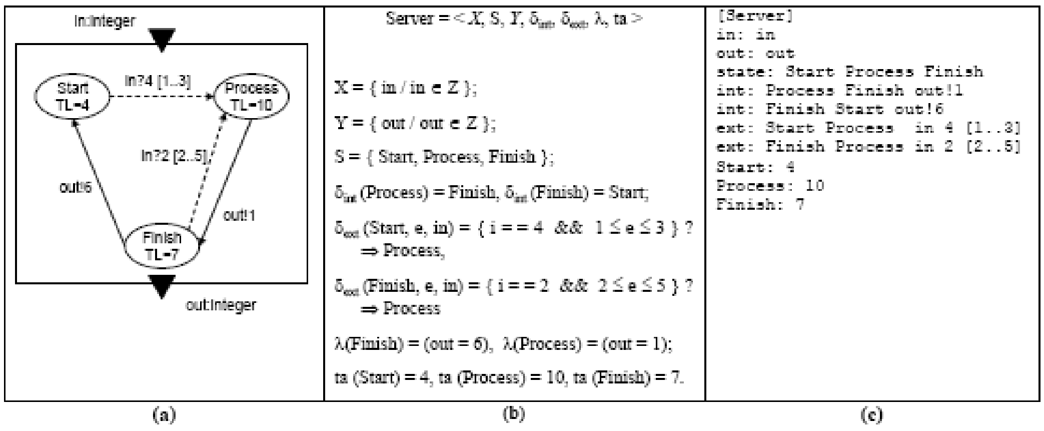
Alternative Notation

The close relation between DEVS and finite state machines indicates that state diagrams might be used as an alternative notation for the specification of these systems [19]. Graph based notations can be argued to be superior in communicating requirements and specifications and is usually easier to understand than sets of mathematical functions.

Figure 5.4 presents an example of such an notation described in [15]. Part a) presents the finite state machine, b) the DEVS notation and c) presents a notation used specifically in their project.

[1]If an internal and an external event occurs simultaneously, it is specified that the internal state transition function is to be applied before the external.

[2]The component associated with the next event



Server = < X, S, Y, δ_{int}, δ_{ext}, λ, ta >

X = { in / in ∊ Z };

Y = { out / out ∊ Z };

S = { Start, Process, Finish };

δ_{int} (Process) = Finish, δ_{int} (Finish) = Start;

δ_{ext} (Start, e, in) = { i = = 4 && $1 \leq e \leq 3$ } ? ⇒ Process,

δ_{ext} (Finish, e, in) = { i = = 2 && $2 \leq e \leq 5$ } ? ⇒ Process

λ(Finish) = (out = 6), λ(Process) = (out = 1);

ta (Start) = 4, ta (Process) = 10, ta (Finish) = 7.

```
[Server]
in: in
out: out
state: Start Process Finish
int: Process Finish out!1
int: Finish Start out!6
ext: Start Process  in 4 [1..3]
ext: Finish Process in 2 [2..5]
Start: 4
Process: 10
Finish: 7
```

Figure 5.4: Graph Notation for Specifying a DEVS model

As can be seen here the structure of the system is completely specified by the finite state machine. Note that the external state transition function is shown with dashed arrows, while the internal state transition function has whole arrows.

A similar notation can also be found as a proposal for standardisation in [28].

5.1.2 Discrete Time System Specification

Discrete time systems are similar to discrete event systems in that they deal with a discretized time base. The difference is however that these systems operate on a stepwise mode with a fixed frequency. That is, input, state and output is dealt with at fixed time intervals. Figure 5.5 shows how the input and output changes over time.

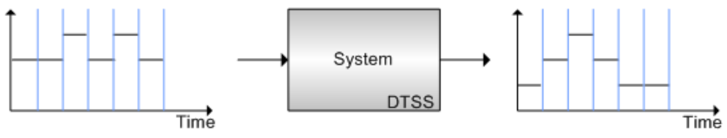


Figure 5.5: DTSS Input and Output

As can be seen here, even though no changes in the input value occur, input will be evaluated at the fixed time intervals.

A discrete time system is specified by the Discrete Time Specified System (DTSS) formalism, which structure is given by:

$$DTSS = (X, Y, Q, \delta, \lambda, c)$$

X is the set of inputs
Y is the set of outputs
Q is the set of states
$\delta : Q \times X \rightarrow Q$ is the state transition function
$\lambda : Q \rightarrow Y$ or $\lambda : Q \times X \rightarrow Y$ is the output function[3]

[3]Moore or Mealy type, see [38]

c is a constant employed for the specification of the time base

X and Y here indicate the sets of input and output events as in DEVS. Q defines the set of all possible states and δ is the only state transition function.

Also DTSS systems can be composed to form larger systems. Such a system is specified by the Discrete Time Specified Network Formalism (DTSN). This structure is given by:

$$DTSN = (X, Y, D, \{M_d\}, EIC, EOC, IC, c)$$

X is the set of inputs
Y is the set of outputs
D is the set of component references
$\{M_d\}$ is the set of DTSS models, where $d \in D$
EIC is the set of external input couplings
EOC is the set of external output couplings
IC is the set of internal couplings
c is a constant time advance in the time base

Here EIC, EOC and IC all represent the same as in the DEVS formalism. The c is the same time constant as used in the atomic formalism, but note that all sub-systems must share the same constant.

The literature also deals with special discrete time systems, for example input-free and memory-less systems. This thesis will however focus on the basic formalisms as these are believed to be capable of describing most systems. For more on these special systems and related specifications, see [38]. Also notice that the discrete time system can be seen as a special case of discrete event system.

5.1.3 Differential Equation System Specification

The behaviour of continuous time systems is often specified by differential equations. That is, functions are used that specify the rate of change of a value, for example:

$$dvalue/dt = 4 * t + 7$$

Figure 5.6 shows an example of how the input and output of a continuous time system can change over time.

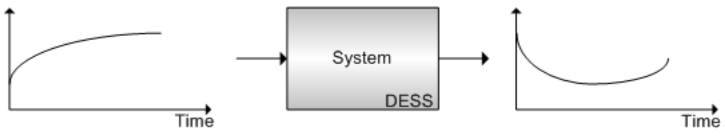


Figure 5.6: DESS Input and Output

As can be seen here, the graphs are continuously evolving. Also piece wise continuous behaviour is allowed. This behaviour can be found in many systems, but is often associated with natural systems. Examples are wave behaviour or in systems involving motion.

In the DEVS-based literature these systems are specified by the Differential Equation Specified System (DESS) formalism. The DESS structure is given by:

$DESS = (X, Y, Q, f, \lambda)$

X *is the set of inputs*
Y *is the set of outputs*
Q *is the set of states*
$f : Q \times X \rightarrow Q$ *is the rate of change function*
$\lambda : Q \rightarrow Y$ *or* $\lambda : Q + X \rightarrow Y$ *is the output function*[4]

Similarly, a network of DESS specified systems can be described by the Differential Equation Specified Network Formalism (DESN), which structure is:

$DESN = (X, Y, D, \{M_d\}, EIC, EOC, IC)$

X *is the set of inputs*
Y *is the set of outputs*
D *is the set of component references*
$\{M_d\}$ *is the set of DESS models, where* $d \in D$
EIC *is the set of external input couplings*
EOC *is the set of external output couplings*
IC *is the set of internal couplings*

Treating continuous change in computers is however not without difficulties. The reason for this is that computers only deal with discrete values and time. Numerical integration is therefore applied to simulate the behaviour of a continuous time system. An example of such a method is Euler's method of integration where the next value of q is defined as $q(t + h) = q(t) + h * dq(t)/dt$ where h is the time step. Note that as $h \rightarrow 0$ also the $error \rightarrow 0$.

Using numerical integrations implies that the DESS cannot be simulated without error [38]. The reason for this is naturally that the time step has to be larger than 0. For this reason, it can be argued that DESS may be a valid way of specifying the system, but when creating the simulator based on this formalism, the error that eventually will be introduced is hidden to the modeller. That is, it might play an important role with regards to the simulation relation.

5.2 Multi-Formalism Modelling

All previously described formalisms have in common that they specify a specific subset of systems. However, in many situations it may be useful to specify systems which behaviour can be characterized as both discrete and continuous. Two approaches for multi-formalisms modelling have been depicted in [38] and [36].

The first approach involves combining formalisms into a super formalism. The Combined Discrete Event and Differential Equation Specified System (DEVDESS) formalism mentioned above is a result of this approach.

The second approach involves embedding formalisms into DEVS [5]. Due to the computational limitations with regards to representation of continuous data flows, attempts are being made towards simulating this behaviour using discrete formalisms.

[4]Moore or Mealy type, see [38]

[5]The term 'embedded' is here meant to refer to that the formalism can be simulated by DEVS components

In addition to these approaches, a new formalism has been introduced in recent time. This alternative is called Generalized DEVS (GDEVS) and will be introduced along with the other alternatives in the following.

5.2.1 Combined Discrete Event and Differential Equation Specified System

As previously mentioned the DEVDESS formalism tries to represent both continuous and discrete time systems [38]. Figure 5.7 shows how such a system is envisaged with both continuous and discrete input and output ports.

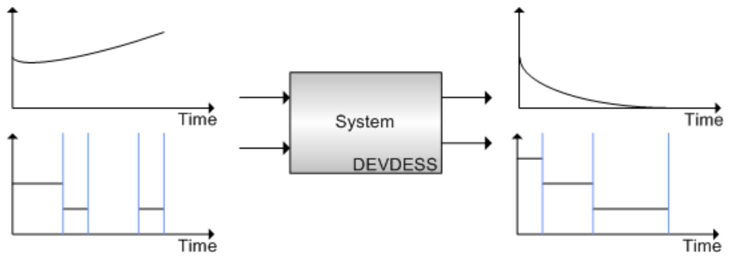


Figure 5.7: Example, DEVDESS System

The details of how this solution is envisaged will not be discussed here. However, it can be mentioned that the interaction between the discrete and continuous part is the key of this formalism. The structure of the DEVDESS formalisms is given by:

$$DEVDESS = (X^{discr}, X^{cont}, Y^{discr}, Y^{cont}, S, \delta_{ext}, C_{int}, \delta_{int}, \lambda^{discr}, f, \lambda^{cont})$$

X^{discr} *is the set of input events from DEVS*
X^{cont} *is the structured set of continuous value inputs*
Y^{discr} *is the set of output events from DEVS*
Y^{cont} *is the structured set of continuous value outputs*
$S : s^{discr} \times s^{cont}$ *is the set of states*
$\delta_{ext} : Q \times X^{cont} \times X^{discr} \rightarrow S$ *is the external state transition function, where Q is the set of total states*
$\delta_{int} : Q \times X^{cont} \rightarrow S$ *is the internal state transition function*
$\lambda^{discr} : Q \times X^{cont} \rightarrow Y^{discr}$ *is the event output function*
$\lambda^{cont} : Q \times X^{cont} \rightarrow Y^{cont}$ *is the continuous output function*
$f : Q \times X^{cont} \rightarrow S^{cont}$ *is the rate of change function*
$C_{int} : Q \times X^{cont} \rightarrow Bool$ *is the state event condition function*

As can be seen here, the formalism adopts many of the elements from both DEVS and DESS. However, this formalisms differs by having two output functions, one discrete and one continuous. Also a state event condition function is used to identify when a state change is ready due to changes in the continuous part. For example, when some continuous state variable has exceeded a given threshold.

Like with the basic formalisms, also here it is possible to build larger systems of DEVDESS systems. The formalism used to specify these systems is called Multi-Formalism Network (MFN), which structure is given by:

$$MFN = (X^{discr}, X^{cont}, Y^{discr}, Y^{cont}, D, \{M_d\}, EIC, EOC, IC, Select)$$

X^{discr} *is the set of input events from DEVS*
X^{cont} *is the structured set of continuous value inputs*
Y^{discr} *is the set of output events from DEVS*
Y^{cont} *is the structured set of continuous value outputs*
D is the set of component references
$\{M_d\}$ *is the set of models, either DEVS, DEVN, DTSS, DTSN, DESS, DESN, DEVDESS or MFN, where* $d \in D$
EIC is the set of external input couplings
EOC is the set of external output couplings
IC is the set of internal couplings
Select is the tie-breaking function

The structure of MFN is similar to that of the other coupled formalisms. Note however that this formalisms allows for almost any type of formalism to be inserted, truly providing a tool for proper multi-formalism modelling [6].

In addition, the DEVDESS formalism also has some unique capabilities not available in the other formalisms [36]. These include state change detection, derivative function change, integrator state reset and event detector threshold change, all of which increase the ability to express and represent systems.

Unfortunately, similarly to the DESS formalism, DEVDESS also incorporates the issue of how to simulate continuous signals on discrete computers. That is, also here numerical integration will be needed to compute the continuous output, resulting in a hidden discrete simulated behaviour.

Couplings

As MFN is capable of coupling formalisms based on different time paradigms, a natural question in this regard is whether any limitations exist.

It has however been shown in [38] that almost all couplings between DEVS, DTSS and DESS components are allowed [7]. Figure 5.8 adopted from [38] presents the relevant couplings.

As can be seen here, all couplings except DESS to DEVS are as one would expect. DEVS to and from DTSS has no limitations. Further, DTSS can sample the output from a DESS and DESS allows any input as long as events are treated as piecewise constant segments.

The DESS to DEVS case however has to be limited to using continuous systems where the continuous output from the DESS component is piecewise constant. The reason for this is that a continuous changing input to the DEVS component would result in an infinite number of events. Note however that if the DEVS component is expressed as a DEVDESS this would not be an issue.

5.2.2 Embedding Formalisms into DEVS

The second approach of multi-formalism modelling, involves the transformation of continuous time systems into discrete systems. That is, simulating the continuous

[6] [38] shows that both DEVS, DTSS and DESS can be interpreted as special cases of DEVDESS.
[7] Note that interfaces might be required to transform the input.

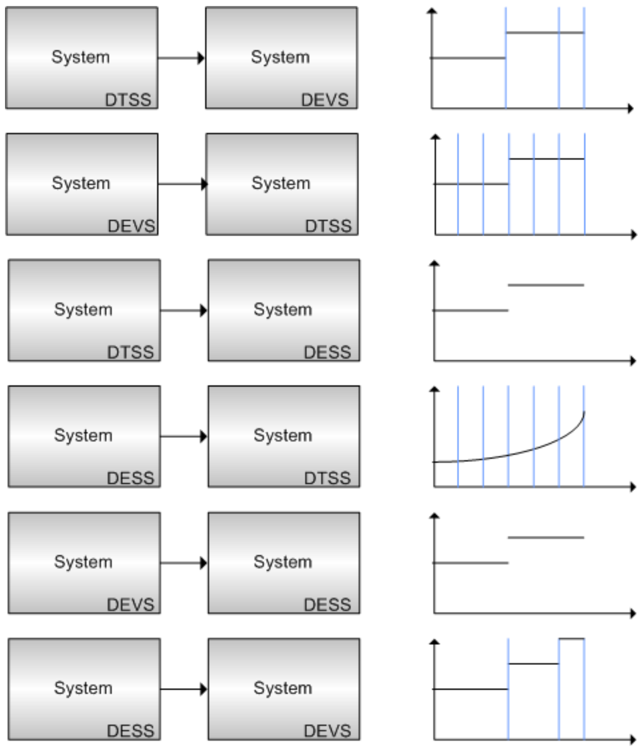


Figure 5.8: Multi-Formalism Couplings

behaviour by either discrete time or discrete event behaviour.

[38] and [36] have proven that all the above mentioned formalisms can be embedded into DEVS at a given accuracy, providing a uniform way of simulating systems. As mentioned before, the trivial approach for doing this is by sampling or numerical integration, but the concept of Quantized Systems (QS) has also been developed for this purpose.

As opposed to partitioning the time into time intervals like in DTSS, the aim of QS is to partition the input and output into quantums, resulting in a DEVS-like behaviour. A quantum is here a measure of how big a change must be to be considered significant [14], see Figure 5.9 adopted from [14].

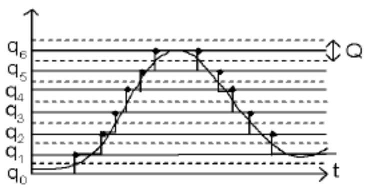


Figure 5.9: Signal Quantization

As can be seen here, the value range is divided into a set of quantums and events are produced whenever the signal exceeds the quantum threshold.

QS can be realized by the help of a component called the Quantizer [14] which transforms the continuous input signal to a discrete output by detecting when a significant event occur. Further adopting this approach results in a component called the Quantized Integrator which is the quantized counterpart of the discretized (numerical) integrator [8].

The accuracy of QS has also been shown to be equal to that of the discretized approach [4]. However, QS benefits from the reduction of message passing as only updates are performed when a quantum threshold is reached, a significant aspect with respect to distributed computing.

In later years an extension of QS is also found called Quantized State Systems (QSS) which include the use of hysteresis to increase the accuracy of this approach. QSS2 has also been developed, further improving this approach. For more information about these techniques see [4] and [14].

The result of this approach is closely related to the DEVS Bus concept found in [38]. The DEVS Bus allows any component with a DEVS interface to be coupled to the simulator. Embedding formalisms into DEVS provides with such DEVS interfaces for all the formalisms mentioned above. Note however that this is usually achieved by using DEVS wrappers.

5.2.3 Generalized DEVS

The three basic types of systems described above can be seen as three paradigms for describing a system. Generalized Discrete Event Specification (GDEVS) is a new initiative, and hence a fourth paradigm in this regard [6] [7].

GDEVS is a generalization of the standard DEVS formalism [6]. This formalism allows the input and output to be specified as piecewise polynomial segments based on the coefficients as opposed to continuous and piecewise constant segments as with the basic discrete formalisms. For example, if the output of a given component can be represented as a linear function f(x) = (a * X) + b, the message (a, b) can be sent to the receiving component at time t_0 which then is able to calculate all future values until a new message arrives. Figure 5.10, adopted from [7], shows how this is envisaged.

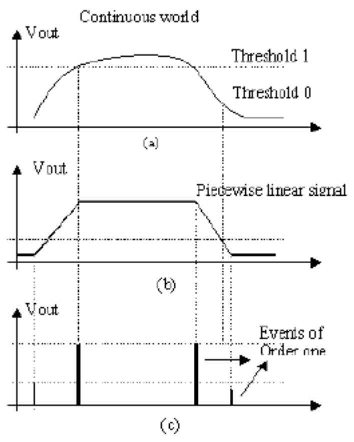


Figure 5.10: GDEVS Approximation

[8]The Quantized Integrator can be specified by DEVS

The figure shows how a continuous signal in a) is represented as a piecewise linear function in b). Also shown in c), is that only four events are needed to represent this behaviour.

The concept of a coefficient event is here introduced as an instantaneous change in at least one of the coefficients [6] [9]. Such an event will result in a new message being sent.

This example uses a piecewise linear approximation which can be characterized as a GDEVS of order one. Note that any order might be used allowing the modeller to specify the accuracy of the simulator. Also note that the standard DEVS formalism is equal to GDEVS of order zero [6].

The structure of GDEVS is in the form:

$$GDEVS_n = (XM, YM, S, \delta_{ext}, \delta_{int}, \lambda, ta, Coeff)$$

$XM = A^{n+1}$, *where A is a subset of integers or real numbers*
$YM = A^{n+1}$
$S = Q \times (A^{n+1})$
$\delta_{ext} : Q \times XM \rightarrow S$ *is the external state transition function*
$\delta_{int} : S \rightarrow S$ *is the internal state transition function*
$\lambda : S \rightarrow YM$ *is the output function*
$ta : S \rightarrow \Re_0^+ \cup \infty$ *is the time advance function*
Coeff is the coefficient function

As can be seen here, instead of the standard input and output sets, XM and YM refers inputs and outputs consisting of multiple coefficients. A Coeff function is also provided to convert between coefficients and trajectories.

Also, similarly to DEVN a GDEVN can be realized where the structure of the GDEVN is equal to that of DEVN. A GDEVN can also consist of components of different order. An interface might then be needed to translate between the different expressions.

An important property of GDEVS not mentioned above is the need to have knowledge about future behaviour. As the output from a GDEVS also determines future values until a new output event is made, this knowledge is required if using GDEVS shall be useful. However, in many cases this requirement can be argued to be met, as one often knows the output of the system over time, given that the input does not change. In systems where the input, and hence the output, often changes, GDEVS might not fulfil its promises of reduced message traffic.

5.3 Relevant DEVS Extensions

Other relevant formalisms are related to distributed and parallel simulation as well as real-time simulation. The system to be specified in this thesis has elements associated with both of these types of simulations and for this reason, two of the most relevant extensions of DEVS will be described below.

[9]Note that this definition might be relaxed to allow for a certain error if needed.

5.3.1 Parallel DEVS

Parallelism is often related to distributed computing. Although the goal of distributed computing often is to have components geographically dispersed, it can also be to exploit the possible parallelism available.

Standard DEVS is based upon serialisation to avoid event collisions, and a Select method is used to choose the imminent component. Further, the semantics of DEVS states that when an internal and an external event occurs simultaneously, the internal state transition function is applied first.

An extension of DEVS, Parallel DEVS (PDEVS), has therefore been developed to allow for parallel execution of discrete event systems [2]. In particular, PDEVS was developed to handle collision handling, parallelism, closure under coupling and hierarchical consistency.

The structure of an atomic PDEVS is given by:

$$PDEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$$

X *is the set of input events*
T *is the set of output events*
S *is the set of sequential states*
$\delta_{ext} : Q \times X^b \rightarrow S$ *is the external state transition function, where* X^b *is a set of bags over elements in* X
$\delta_{int} : S \rightarrow S$ *is the internal state transition function*
$\delta_{con} : S \times X^b \rightarrow S$ *is the confluent transition function*
$\lambda : S \rightarrow Y$ *is the output function*
$ta : S \rightarrow \Re_0^+ \cup \infty$ *is the time advance function*
$Q = (s, e)s| \in S, 0 < e < ta(s)$ *is the set of total states*

As shown here a new function δ_{con} has been created to deal with the collision between internal and external events. When such a collision occurs, this function is used to determine the next state instead of the external and internal state transition functions. Adding this function also greatly increase the modellers' control of the simulation and can be independently specified for each component.

The coupled version of the PDEVS structure is called PDEVN and similar to DEVN as described above with the exception that the Select function is removed [2]. All models inserted into the PDEVN structure must be PDEVS.

PDEVS was introduced in [2] as a revision of DEVS and is also often called DEVS in the later literature. It can also be argued that PDEVS removes a limitation of DEVS, the implicit serialization, and allows more freedom to the modeller. For this reason, as well as both DEVS and PDEVS specify the same subset of systems, it can further be argued that PDEVS should be used instead of DEVS when specifying DEVS systems [10].

5.3.2 Real-Time DEVS

Human in the loop or hardware in the loop simulations can often be classified as real-time systems. Real-time systems differs from standard systems in the fact that

[10]Note, that it is presently unknown how well PDEVS is supported in DEVDESS and MFN.

in addition to accuracy, timeliness becomes important [11] [1]. That is, the output or results of calculations are only valid if given at the correct time or within a given time window or deadline.

In literature, two approaches for real-time simulation are mentioned. These are either adopting a new formalism, specially created for the purpose, or by controlling this issue as part of the simulator, for example through a special protocol (supervisory control).

The Real-Time DEVS (RTDEVS) formalisms, an extension of DEVS, is an example of the first alternative [38] [10], where an abstract simulator. The simulation time is here synchronized with the computer clock through mapping of an activity to each state. This activity is an executable function whose completion is required before the state can be changed.

The structure of an atomic RTDEVS is given by:

$$RTAM = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta, ti, \psi, A)$$

X *is the set of input events*
Y *is the set of output events*
S *is the set of sequential states*
$\delta_{ext} : Q \times X \rightarrow S$ *is the external state transition function*
$\delta_{int} : S \rightarrow S$ *is the internal state transition function*
$\lambda : S \rightarrow Y$ *is the output function*
$ta : S \rightarrow \Re_0^+ \cup \infty$ *is the time advance function*
$ti : S \rightarrow \Re_{0,\infty}^+ \cup \Re_{0,\infty}^+$ *is the interval time advance function*
$\psi : s \rightarrow A$ *is the activity mapping function*
A *is the set of activities with constraints*

Note that in addition to the activities and the mapping function, an interval time advance function, which specifies the minimum and maximum limits of when the activity is supposed to be finished, has been introduced. This is due to the non-deterministic nature of the activity execution. Execution will therefore be tested against these boundaries in run-time and errors produced if the boundaries are exceeded.

Further also here coupled systems can be made. The coupled version of RTAM, RTCM, is similar to DEVN with the exception of the Select function which is not present. Parallelism is here implicit in this formalism.

The RTDEVS formalisms was created for the purpose of bridging the gap between analysis and implementation [10], however there might also be many situations where it may be useful to have the simulator run in virtual time as well. This can for example be when validating components or when no real world component is present in the loop.

In this situation having the real-time constraints as a part of the model might be unfavourable, as there will be a need for having two separate models. One real-time model for real-time simulation and one regular model for other simulations [11].

This, as well as simulation time can be said to be a property of the simulator and not the model, argues for the second alternative suggested above. That is, using only a special protocol to achieve real-time simulation. One such approach is discussed in [33] where real-time control is implemented in the DEVS-Scheme environment.

[11]Note however that RTDEVS can be seen as a wrapper around a standard DEVS specification.

In such an approach the model would be almost independent of the chosen simulation time [12] and would be reusable with regards to simulations requiring different timing. Further, this approach would also suite the separation of concerns benefit aimed for by separating model and simulator better.

Notice however that as one argues that the real-time control should be part of the simulator and not the model, this implies that RTDEVS might be used in the specification of the simulator. Naturally, this assumes that the real-time protocol followed in the simulator corresponds to the semantics of the RTDEVS formalism.

Examples of using RTDEVS and supervisory control can be found in [10] and [33] respectively.

5.4 Discussion

This chapter has presented some of the formalisms described in the literature and shown how these are applied to specify the behaviour of different systems at both an atomic and coupled level.

It is also noted that approaches have been performed towards multi-formalism modelling. For any large-scale and complex system to be modelled, it is likely that the system will contain both discrete and continuous time behaviour arguing for the use of formalisms that allow both types.

Table 5.1 presents an overview of the formalisms examined and what kind of systems they are capable of describing.

Table 5.1: Comparison of Formalisms

System Time Base	**DEVS**	**DTSS**	**DESS**	**DEV-DESS**	**G-DEVS**	**QS**
Discrete Event	YES	NO	NO	YES	YES	YES
Discrete	YES	YES	NO	YES	YES	YES
Continuous	NO	NO	YES	YES	YES*	YES*

As can be seen here, DEVDESS and its coupled version the MFN together with the basic formalisms, are the only formalisms allowing potentially error-free specification of all types of systems [13]. The only limitation here exists with regards to couplings from a DESS to a DEVS component where the input needs to be piecewise constant. However if the DEVDESS formalism is used instead of DEVS, this issue can be relaxed.

Further, it has been shown that error-free simulation of continuous time systems on discrete computers is not possible. Realizing these systems through use of GDEVS or through embedding formalisms into DEVS will however allow for simulation, but only at a given accuracy.

Due to the error introduced in the transformation to discrete systems it might be useful to create a lumped model which is only based on discrete formalisms in addition to the more accurate base model, see Figure 5.11. This lumped model will both allow for studying the relation between the base model and the lumped model and at the same time make the difference explicit.

[12]Simulation time would only be limited by the computational complexity of the model, as well as architecture of the simulator.

[13]Note that there might exist continuous systems which cannot be described through differential equations.

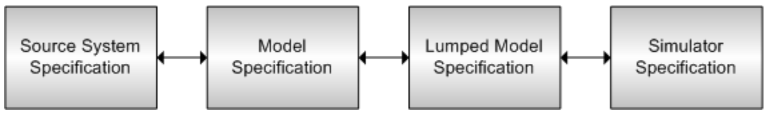


Figure 5.11: Lumped Model

In this context, it can be argued that DEVDESS and the other basic formalisms be used to specify the base model, while the lumped model is specified by either QS or GDEVS. The choice of which formalism to use, QS, GDEVS or other formalisms with similar properties, is however unclear and should be studied further. This is however outside the scope of this thesis and will not be discussed any further. Some early experiences on continuous and hybrid simulations can however be found in [4].

Note here that the goals of such a lumped model is to get a representation of the source system which perhaps is a simplification, but is possible to simulate within the given constraints of the simulator.

5.4.1 Parallel and Real-Time Simulation

With regards to parallel and real-time simulation, both PDEVS and RTDEVS has been mentioned. PDEVS could be argued to be used instead of DEVS in many situations. However since its support with regards to DEVDESS and MFN is unknown, it can be argued that this first be introduced in the lumped model which is based upon DEVS or GDEVS alone.

An assumption is here made that a parallel version of GDEVS can be found. Since GDEVS only differs from DEVS in the messages sent, this should however not be an issue.

With the case of real-time simulation it was argued that RTDEVS only should be used in specifying the simulator. The lumped model as defined above may provide a entity prior to the simulator, which may be suitable for this formalism.

Also here, it can be argued that a GDEVS version of RTDEVS should be possible.

5.4.2 Hierarchy of System Specification

The formalisms presented in this chapter can all be associated with level 5 and up in the hierarchy of system specification. A natural question would then be how to express the lower level specifications and in particular the source system specification which usually is specified at level 0-1.

If the system is already specified at a higher level, this task becomes trivial as one can always infer a lower level specification. However, this is not always the case.

A specification of level 0 can be argued to be obtained through using only the input and output slots in the formalisms presented in this chapter. Rigid formalisms for level 1 and 2 are however unknown. [38] uses the concept *system behaviour database* for this purpose. Producing a list of behavioural trajectories, this can be argued to be a specification at level 1. A similar approach can be envisaged for level 2, producing a list of functions for each input trajectory. One should however notice that with this representation, manually comparing specifications can become difficult, if not infeasible.

Finally, it can be argued that one should always use the structured formalisms with defined ports and variables. This may both help present a more logical view of the system as well as it can be argued that one always know, or is able to identify, the variable that one is observing or studying.

5.4.3 Experimental Frame Formalism

As mentioned in Chapter 4, special formalisms also exist for the experimental frame. It can however be argued that if the experimental frame should be realized as a system it should be specified by the same formalisms as the model providing a unified description of the global system. Further, this will also allow the same abstract simulator to be used on both frame and model.

For this reason, it is chosen not to focus on these alternative formalisms in this thesis. For more information on these structures see [39] [27].

5.4.4 Alternative GDEVS Usage

The above presentation of GDEVS propose that higher order GDEVS specifications should only be used on continuous behaviour. However, if the future discrete output of a system can be predicted, it can be argued that this behaviour can be represented as continuous for example through interpolation.

Figure 5.12 shows how this is imagined.

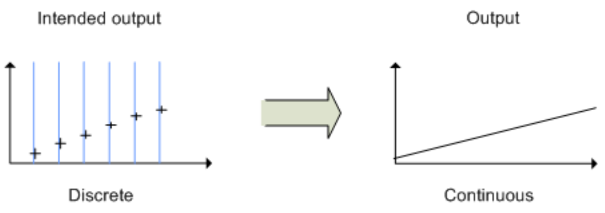


Figure 5.12: Alternative GDEVS Usage

As can be seen here, the discrete output, to the left, can be represented as a continuous output, to the right. For example, this could be the simulation of a discrete clock. In stead of having the clock component output a new value at a fixed frequency, it can output a linear function describing the future behaviour, reducing number of messages needed to 1.

This approach can be envisaged in two ways. Either the function is directly specified in the formalism as described above or one adopts an interpolation algorithm to calculate new polynomial functions for example based upon the last X discrete events.

In the end this can be seen as an approach towards reducing the number of messages sent based on predicting future values. If no correct predictions are made, no reduction in messages is achieved either.

The approach is however not without issues. For example, as with all interpolation, there is no guarantee that the interpolated functions are correct in terms of time points not part of the original output. Issues with regards to validation and in how this data is interpreted in the receiving component may arise.

At present time it is unknown how well this approach acts out in practice. Although it has a potential for further reducing the message traffic, its possible drawbacks should be studied further. The technique should also be studied in light of interpolation processing time and chance of prediction.

Chapter 6
Validation and Verification

Being able to trust the results provided by a simulation is an absolute necessity. In the DEVS-based literature credibility is closely related to the concepts of validation and verification [39] [38]. Well validated models and well verified simulators provide a basis for achieving this trust [40].

In the transformations between source system, model and finally simulator there will most likely be some degree of error. For example, the model will often only be a simplification of the source system. It is therefore important that these relations are studied thoroughly. Figure 6.1 shows how validation and verification is associated with the relations found in the modelling and simulation framework.

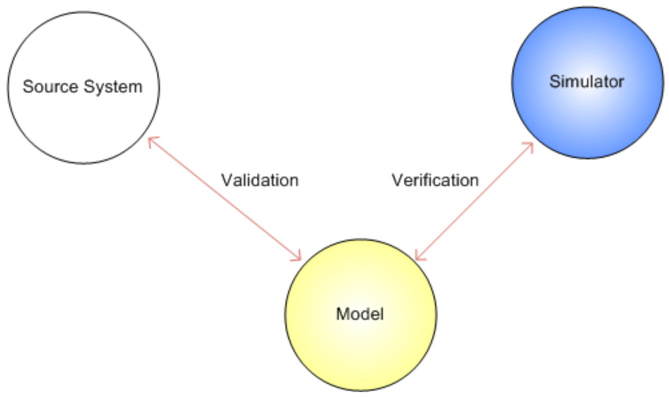


Figure 6.1: Validation and Verification

The need for formal methods for establishing these relations is one of the main motivations for the DEVS-based literature [34]. In the DEVS-based theory these relations are dealt with as morphisms between the system specifications. For this reason a short presentation of morphisms will be given first, before validation and verification are defined.

6.1 Morphisms

A *morphism* is a mapping between two abstract objects or structures [31]. That is, a morphism tries to capture the similarity between two structures, for example two system specifications.

Both the modelling relation and the simulation relation in the modelling and simulation framework can be seen as morphisms between specifications [38].

Further, two types of morphisms are important: *isomorphism* and *homomorphism*, where isomophism is achieved when there is a one-to-one and onto mapping between the elements of both structures. A general morphism is often called homomorphism, and only requires a one way mapping.

In the context of DEVS, homomorphism is often used in regards to the specification of structure. Although the behaviour of the compared systems need to be equal, isomorphic, their inner states can differ as long as there is a mapping between them. Note that the states need not be exactly the same [38]. Figure 6.2 shows an example of two homomorphic systems, where the states of system 2 can be mapped to the states of system 1 in a one way fashion. This can for example be a mapping of a model onto a simulator specification.

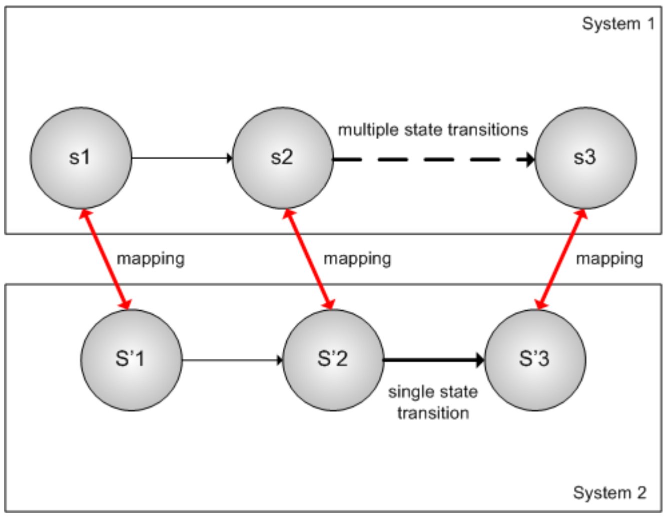


Figure 6.2: Homomorphism

With this in mind, morphisms can be found at each level in the hierarchy of system specifications. Although the details of these morphisms are outside the scope of this thesis, it can be mentioned that since typically the source system is specified at level 1 or less, achieving isomorphism at this level between the source system and the simulator is a natural goal in simulation. At higher levels, homomorphism might be sufficient.

More on the hierarchy of system morphisms can be found in [38]

Approximate Morphisms

As the reader might be aware of, the relation between systems need not necessarily be error free. For example can a system be too complex or there might be a trade-off between accuracy and efficiency involved. For this reason the concept of *approximate morphism* is introduced, where an approximate morphism is a morphism where not all requirements are met [38]. This is typically found between the source system and the model and also between the model and the lumped model.

[38] identifies a component as critical in this regard if it is source of error growth and there exists a path from the component to the experimental frame. Error growth here refers to whether the error of the component grows or attenuates over time. Naturally, if the error constantly grows there will be an issue. Further, the error is only interesting if it influences the output. It can however be noted that error can be controlled if the component memory is reset at given intervals.

If the component can be said to be critical, naturally modifications will need to be studied further until a satisfactory representation of this component is achieved.

6.2 Validation

Validation is usually associated with the modelling relation in the modelling and simulation framework. That is, validation deals with whether the model is an acceptable representation of the source system or not [15].

With respect to validation the experimental frame plays an important role. That is, the model is only evaluated accordingly to the conditions specified in the experimental frame. In other words, it validated with regards to intended use [17], and it is only valid if the applicability relation between the frame and the model holds [34].

Usually one deals with replicative validity and predictive validity, here called behavioural validation, as well as structural validation. More on these topics will be discussed in the following.

6.2.1 Behavioural Validation

Behavioural validation refers to validating the behaviour of a given model. In simulation the behaviour is what we are actually trying to mimic, and for this reason, validating the behaviour becomes of the greatest importance.

Replicative validity determines whether the model accurately produces the same results as observed by the source system given a set of inputs. This validation is associated with morphism at level 1 [38].

However, most likely it is only possible to observe and register a subset of all possible scenarios and hence you do not have any guarantee that the model will produce correct results in scenarios not yet observed. This introduces *predictive validity* which tries to say something about how well the model behaves on input segments not used in creating the model. Proving this form of validity is however difficult as you can only prove error and never complete correctness. Testing and comparing with new unseen input segments can however increase the belief in the predictive validity of the model. Often this validation is associated with morphism at level 2 [38].

6.2.2 Structural Validation

Structural validation refers to validation of the structure of a model. Although simulation only deals with mimicking the behaviour of a the system of interest, the structure, if known, may also be of interest.

The reason for this is that the chosen structure need not be the only structure consistent with observable data. That is, there might be many systems which produce the correct output for the given input subset. Finding the correct structure also capable of producing correct output on unseen scenarios may be valuable.

This is highly related to predictive validity, and if structural validity is proven, the predictive validity should be strengthened.

As this validation refers to structure and not only behaviour, the morphism here studied is often done at level 3 or higher.

6.3 Verification

Verification deals with how well a simulator executes models [38]. That is, verification deals with the simulation relation in the modelling and simulation framework. Clearly, understanding the precision and limitations of the simulator is valuable with regards to the simulation credibility.

By using formal proofs and specifications it may be possible to establish that the simulator and the model are morphic. Often this is associated with morphisms at level 3 or higher. Note in this regard that the specification of the simulator must be based upon observations of the simulator, due to time discrepancies in real-time simulations, as well as the knowledge of the inner structure.

6.4 Discussion

As presented here, the DEVS-based literature deals with credibility through validation and verification, both of which can be analysed through the system morphism concept given that proper specification exist of all entities.

A few issues are needed to sort out in this regard. First, establishing morphisms depend on the correct answer being known, also known as the oracle problem [15]. Further, there is a question about what is the correct set of input values with regards to intended use. This knowledge can be argued to only be known, if possible, through thorough study of the source system.

Second, the issue of human error becomes important. That is, there need to be a human in the loop which specifies the different entities. Also, the proofs are most likely calculated by humans or humans are needed to interpret the results. This might influence the result greatly, but is also hard to omit. Perhaps, through making more of the process computerized in well-designed tools this issue will be relaxed.

Included in this second issue, is also the problem that similarly as it is possible to model a system wrongly, it is possible to specify and model the experimental frame wrongly. For example, if the experimental frame is set to include a continuous component and an experimenter is used in the validation task, the simplification of the discretization should at least be made explicit so that its influence can be studied.

With these issues in mind, formal analysis also provides with some great benefits. Through formal proofs of the capabilities of the simulator, credibility can be said to be strengthened. Also, the modular property of DEVS, delayed binding [21], allows for component wise validation reducing complexity and increasing credibility.

Testing and Formal Proofs

Both testing and formal proofs have been suggested in establishing the different relations, but given the formal aspect of DEVS it is clear that the theory lends itself towards formal methods. [39] also states that using morphisms for this purpose has been very helpful and [34] states that it is likely that one is in a much better position to argue for correctness using a formal modelling and simulation foundation.

However, it is not a clear case. Some of the aspects of testing and formal proofs are compared in Table 6.1.

Table 6.1: Testing versus Formal Proofs

	Testing	**Formal Proofs**
+	directly comparable to real	no execution, absolute proofs
-	limited scenarios, can only find defects, requires execution	complexity issues, human error, abstract not same as real

Although formal proofs may be affected by human error and complexity issues, it has a great benefit by not being dependent on simulator execution. Comparing specifications can be done entirely independent of the simulator given that the specifications of all entities are available. Further, using abstract logic, real proofs can be made. Note however that the simulator must be executed to specify the simulator specification.

Testing on the other hand does require execution and hence will also be limited to studying only a subset of all possible scenarios if this set is large. Further testing can only identify defects, never prove correctness.

Testing does however also has the benefit that the simulator becomes directly comparable with the source system, which, for many, is far more credible than any abstract analysis.

For this reason it is not known which approach is the most appropriate for this purpose. However, it can be argued that a validation and verification process including both strategies should perhaps be conducted to ensure the trust needed.

Examples of techniques related to both testing and formal proofs can be found in [15], [30] and [39].

Part III

Own Contribution

Chapter 7
Submarine Combat System

This chapter presents an overview of the submarine combat system to provide the reader with some background of the domain. Based on the introduction presented, a subset of the system will be chosen for further study and special simulation aspects and requirements, associated with this subset, will be listed at the end of this chapter.

Note that all information about the submarine combat system is provided through the contacts at FFI. Some details about the system is also based upon a general submarine description and not the specific submarine studied at FFI due to security classification.

7.1 Overall Description

This section presents an overview of how the submarine is usually decomposed and what systems are present. As stated earlier, the system to be analysed in more detail is a subset of the submarine as envisaged it in 2009. The final configuration of this system is therefore at present time not fully decided.

7.1.1 The Submarine

A submarine consists of many systems and can be argued to be a very complex entity. Figure 7.1 shows an overview of the submarine, decomposed into some of the systems usually present.

As can be seen here, the submarine is usually divided into two parts, the Submarine Combat System and the Platform. The Platform here refers to physical systems and components, including the Steering Console and the hull, engine, rudders and so on.

The Submarine Combat System is more abstract and refers to all systems used in controlling the submarine. This includes the following sub-systems: the Navigation System, the Weapons System, the Communication System, the Sensor System and the Combat Management System.

It is here chosen to focus on the Navigation System and a more detailed analysis of this system will be presented in the following. This sub-system is chosen due to its potential for including all types of DEVS-components. Also, the final simulation of this system will potentially include both real hardware and human in the loop. This is also a central system with respect to creating a synthetic environment for

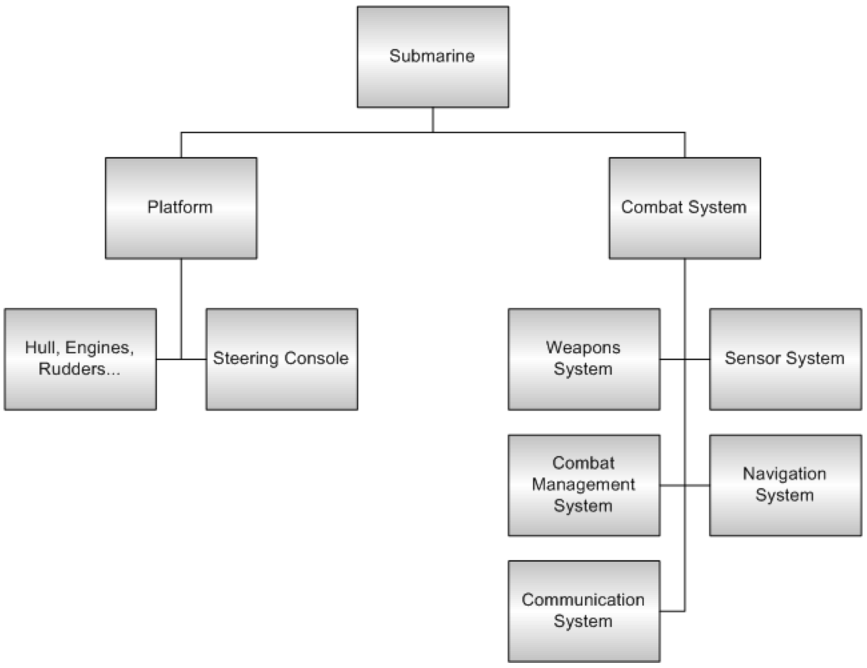


Figure 7.1: The Combat System

the hybrid simulator as this is the system where most of the contact with the environment occurs.

Due to the close relationship between the Navigation System and the Steering Console, a limited version of the Steering Console will also be part of this analysis. This is done to ease the understanding of how the Navigation System works as it gives a more holistic picture of the system.

Figure 7.2 shows a very rough picture of the data flow envisaged in relation to steering the submarine.

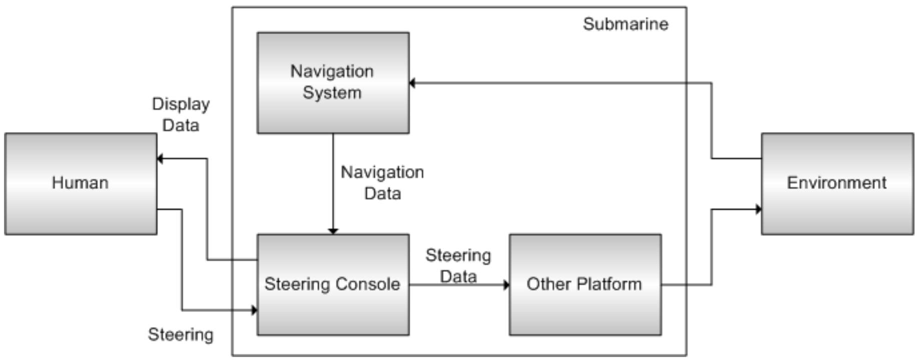


Figure 7.2: Rough Dataflow of Steering

As can be seen here, both the Navigation System and the Steering Console are involved in this process. Humans interact with the Steering Console to change rudder positions and engine rotational speed. This data is sent to the engine and rudders, here called Other Platform, where it is executed. Naturally, these will interact with the sea, here called Environment, and the Navigation System will measure and calculate position, orientation, speed and so on which is finally displayed in the Steering Console.

As stated earlier, humans play an important part of this interplay and should therefore be included. So does the environment, but modelling and simulation of the environment will not be covered in this thesis due to limited resources.

That is, the subsystems; Human, Navigation System and a limited Steering Console will be part of the following analysis.

7.1.2 The Navigation System

The Navigation System is the system whose task it is to find and present information about position, speed and orientation. The system shall also present the surroundings of the submarine in form of a map and/or a navigation sonar.

Two issues are especially important to notice in this regard. The first is that, unlike ships and cars, the submarine moves and navigates in a 3D environment. This increases the complexity greatly and implies that more resources are need to cover these tasks.

In this regard, a geographic coordinate system will be applied to uniquely specify a position. All positions in the following analysis will be given in a vector [x, y, z] referring to latitude, longitude and depth respectively. Latitude and longitude will be given in degrees while depth is given in meters.

In addition, velocity, acceleration and rotation will be described with a similar vector [x, y ,z] indicating velocity or acceleration in each axis and rotation around each axis. These vectors use a coordinate system relative to the submarine where x is along ship, y is to the right and z is down.

Secondly, many sources of information are unavailable while the submarine is submerged. For example, data cannot be received from external sources and no visual input is available. This aspect can be further strengthened by the need for discreetness in many combat situations. The issue is however dealt with in a submarine by including additional functionality for calculating and approximating the unavailable information based on the information currently available and the history. The chosen subset is here strongly influenced by this functionality.

Figure 7.3 shows an overview of the different components that are envisaged part of the Navigation System as of 2009.

As can be seen here, several components make up the system. Each of these will be described in more detail in the following chapter.

7.1.3 Steering Console

The Steering Console allows for the crew to interact with and steer the submarine. Naturally, this is closely related to the functionality provided by the Navigation System as seen in Figure 7.2.

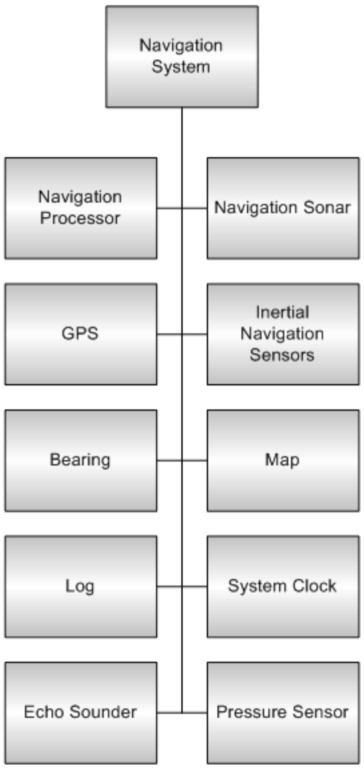


Figure 7.3: The Navigation System

The console includes visualisations of position, orientation, speed, engine rotational speed, rudder positions and so on. Also, devices for changing heading, for example a joystick, and speed are present here.

Focus in this analysis will however be on the Navigation System, but to allow for a more holistic picture of the system, a limited version of the Steering Console will be included. With this in mind, only the visualisation part and the manual steering of the submarine will be included in the limited Steering Console. Components related to order giving and automatic steering will not be covered.

7.2 Simulation Aspects

With respect to simulation of the chosen subset several special simulation aspects can be outlined and should be supported by the methodology used. A short description of these will be given below, before they are discussed in more detail in Chapter 10.

- **Representative Specifications**
  The chosen subset includes both discrete and continuous components. Although this is mostly a discrete computerized system, all system types will need to be described in an acceptable fashion.
- **Legacy Components and Technologies**
  From the above description it is clear that both humans and real hardware may interact with the simulation in some simulation scenarios. The methodology should therefore be able to handle and study aspects of these components as

well as other legacy components and technologies used.

- **Complexity and Maintenance**
  Although the subset to be studied is only a small part of the system to be simulated at FFI, strategies for dealing with complexity will need to be supported. In addition, being able to modify and maintain models is needed to support the aim for studying different submarine combat system concepts. This is also closely related to the goal of rapid model development and is essential if different paths are to be studied with regards to concept development.
- **Composability**
  With regards to the system to be built, the final configuration of the system is not defined. For this reason, the methodology must allow for and support modularity and composability. That is, it must be possible to describe the system as a set of modules and also to compose larger systems based on these. Note, that also legacy components must be supported in this task.
- **Validation and Verification**
  The credibility of such a system is of the greatest importance. As acquisition decisions will be made based on the simulation results, the methodology must support proper validation of the models and verification of the simulators used.

Chapter 8
Model Description

This chapter presents an architectural description of the chosen subset of the submarine combat system. This description can also be seen as a case study of the system to be built in the TEKULA project at FFI as described in Chapter 1. The description will further be a starting point for the discussion of the DEVS-based theory and its usability with regards to the analysis of systems to be simulated and in establishing the credibility of these simulations.

First, assumptions and notation will be shortly described before the experimental frame and the model itself are presented. Chapter 9 will extend the discussion made here, taking into consideration legacy components and technologies.

8.1 Assumptions and Notation

The analysis will be based upon the DEVS-based theory and the main concepts used here. First, the experimental frame of the system will be discussed before an abstract model of the system is given.

In this study a full specification of the different components will not be created. However, an architectural description will be presented, identifying components, influencees and type of specification needed. This is similar to a specification on level 7 for the global system and structured level 0 for each component.

For this task a notation similar to the one used in [38] will be used. This notation is also close to that of a data flow diagram. Figure 8.1 presents the notation.

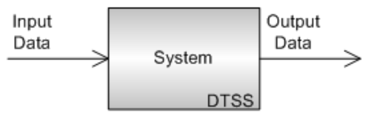


Figure 8.1: Example System

As can be seen here, a system will be given by a grey box with the system type, that is the type of specification that should be used to describe this system, in the lower right corner. To and from this box there will be arrows indicating data flows. Note that names on data flows may be abbreviated to reduce clutter where it is needed.

Finally, textual descriptions, value ranges and real-time constraints for each component will be provided in a table, see Table 8.1.

The value ranges used here will reflect normal operating parameters. This is not necessarily the same as what the real components can handle. In some cases it might also be interesting if the simulated component behaves like the real one even

Table 8.1: Navigation Sonar Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**

outside the normal operating parameters. In the final specification of the system this issue will need to be considered.

It can be argued that real-time simulation is only constrained by discrete time and continuous signals. Real-time constraints will therefore include discrete time frequency and whether the signal is continuous.

Further, since all details about the simulation objectives are not known at present time, this description will not include measures related to objectives, control signals or other signals not part of the source system.

8.2 Experimental Frame

As stated in Chapter 4, the source system is the real world system and the experimental frame shall describe the conditions under which the system shall be experimented with as well as how.

With regards to the systems to be simulated in the TEKULA project, much of this information is still to be determined. That is, what metrics are to be used and how these are to be analysed is presently unknown. However, it is known that the subset described here is part of a greater simulation, which provides with some form of knowledge of what input and output are needed, further providing with a basis for an experimental frame.

With regards to simulation objectives, it is here stressed that these should be determined before a final configuration of the experimental frame and the system is specified.

Similarly, the source system is not available for study. However, much is know about the components making up this system from general descriptions and specifications. For this reason, the description given here will be a simplified specification of the model and not the source system as it is felt that a source system specification should be based upon observations of the source system.

Figure 8.2 presents the experimental frame, shown as a system, based on the knowledge presently available.

As can be seen here, a range of data flows are envisaged. These data flows can be mentioned to consist both of discrete signals and continuous signals. For this reason it is determined that the modelled system should be specified through the MFN formalism. This will be further elaborated in the system model.

Further, it can also be mentioned that the data flows here described covers both data to and from the rest of the submarine system as well as to and from the environment.

8.3 Model

Figure 8.3 presents an overall abstract model of the chosen subset.

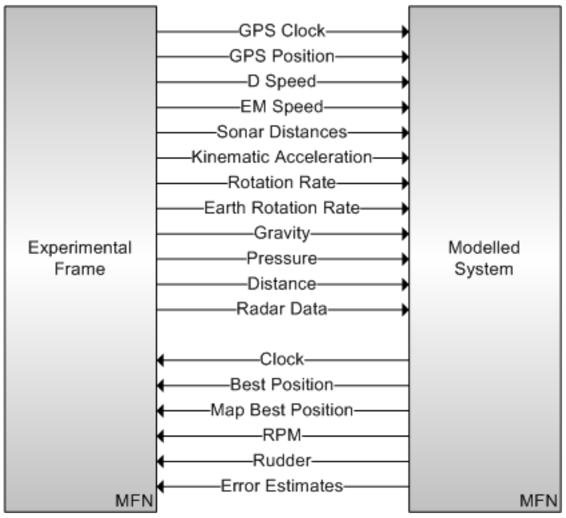


Figure 8.2: Experimental Frame

As can be seen in the figure, most of the components handle input from the environment, experimental frame, and forwards some information to the Navigation Processor [1]. This information is then processed and approximations of position, orientation and speed is sent to the Steering Console as well as out of this system.

Further, the human's role in the system is made explicit and modelled as a separate system. The human steers the submarine through input to the Steering Console and manual bearings to the Bearing component.

With respect to type of behaviour, it can be seen that most of the components in the Navigation System can be described as being discrete. Exceptions are the Echo Sounder and the Navigation Sonar which have both discrete and continuous behaviour. Further, also the human component shows partially continuous behaviour. Note that most input from the environment is modelled as continuous input.

Next, all components of the Navigation System, the Steering Console and the Human component will be described in more detail also adding value ranges and real-time constraints. Some components are decomposed into lesser models. Here, all sub-components will recursively be described using the same method.

In Appendix A examples of fully specified system components are provided for some of the relevant components.

8.3.1 GPS

As shown in Figure 8.3 the GPS component accepts input, clock and position, from the environment. This is the data received from the GPS satellites. The clock received can be used to update the system clock if necessary.

This component can also receive clock from the system clock component and position from the Navigation Processor. Starting up the GPS involves a small time window where erroneous values are given as output. Keeping the GPS component alive at all times with internal system values reduce this error.

[1]Note that the type of input to the system is defined to the left in the figure.

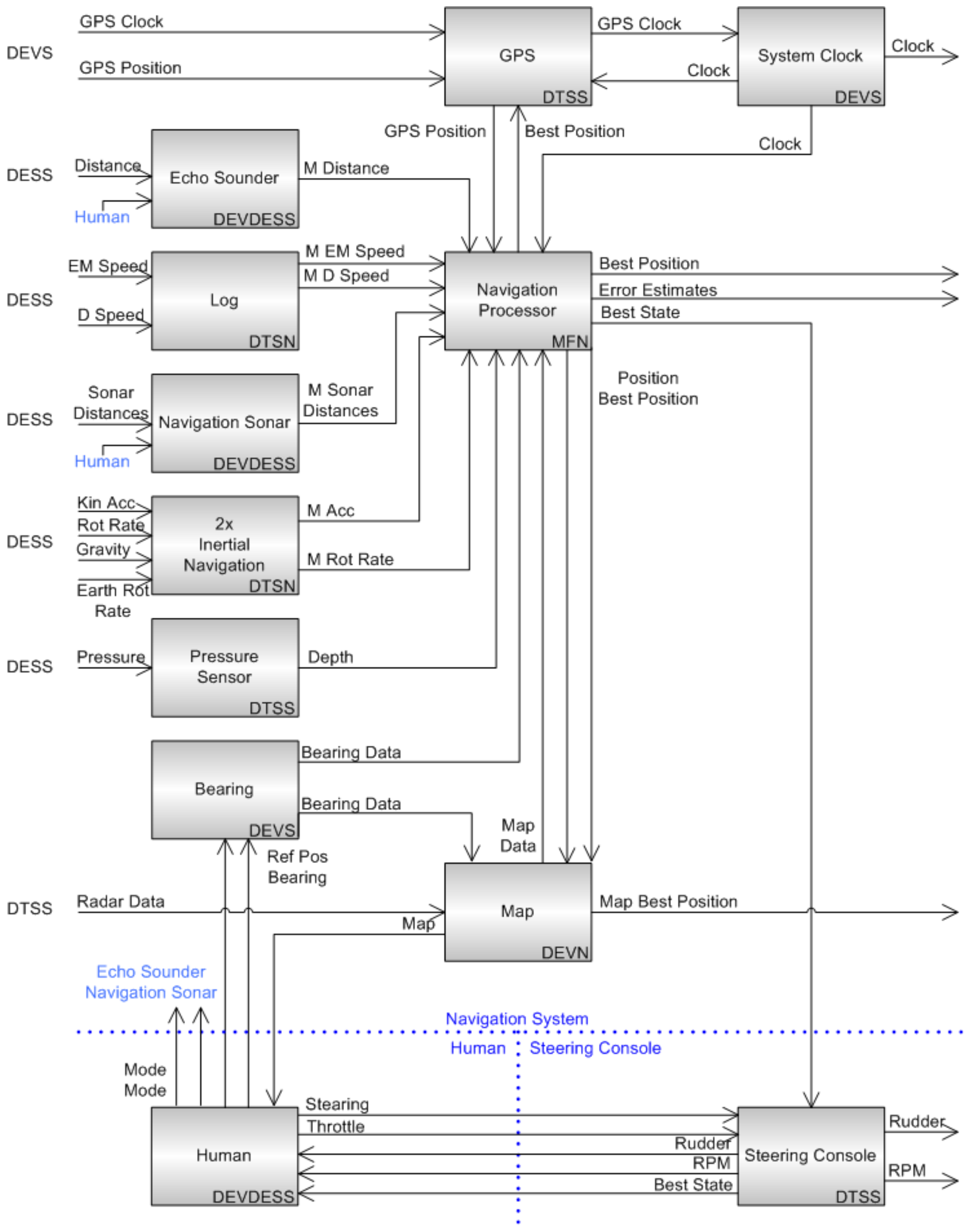


Figure 8.3: Navigation System Model

Table 8.2 presents the interface of this component.

Table 8.2: GPS Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	GPS Clock	>0	
Input	GPS Position	[x,y,z] where x and y is given in degrees and z in m	
Input	Clock	>0	10 Hz
Input	Best Position	[x,y,z] where x and y is given in degrees and z in m	10 Hz
Output	GPS Clock	>0	10 Hz
Output	GPS Position	[x,y,z] where x and y is given in degrees and z in m	10 Hz

It might seem like this component is event driven due to the fact that it receives data at random times. However, as all sources of data are external and no internal state transitions will occur, DTSS should be used to specify this system. Note that the output is also generated at a specific frequency.

An example of a fully specified GPS, given the current level of detail, can be found in Appendix A, Example A.1.5.

8.3.2 System Clock

The system clock defines a common clock used in the system. Although it can be updated from the GPS component, it should be an extremely precise clock, for example an atomic clock. Table 8.3 defines the component interface.

Table 8.3: System Clock Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	GPS Clock	>0	
Output	Clock	>0	100 Hz

With respect to the formalism that should be used to specify this component, it can be mentioned that the update in clock is an instantaneous change. This leaves the possible formalisms to either DEVS or DEVDESS. Since the time is updated at all time steps it may be tempting to use the DEVDESS formalism, but due to the discrete output of this component it is here chosen to use DEVS so that the model is as accurate as possible.

An example of a fully specified System Clock, given the current level of detail, can be found in Appendix A, Example A.1.1.

8.3.3 Echo Sounder

The echo sounder measures the distance to the bottom directly below the submarine. This component has three modes. First, it can be used to perform single pings identifying the distance once. Secondly, it can be run with a given low frequency or thirdly, it is off. Table 8.4 presents the interface of this component.

Table 8.4: Echo Sounder Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	Distance	[0, 1000] m	cont
Input	Mode	[ON, SINGLEPING, OFF]	
Output	Measured Distance	[0, 1000] m	

The component is event driven due to the different modes it handles. However, to be able receive continuous input this component have to be described using the DEVDESS formalism.

An example of a fully specified Echo Sounder, given the current level of detail, can be found in Appendix A, Example A.1.3.

8.3.4 Navigation Sonar

Similarly to the Depth Finder, the Navigation Sonar also measures the distance to the bottom. The difference is however that the sonar presents a 3D image of the surroundings. That is, distance in all directions. Also this component works in three modes similarly to the Echo Sounder.

Table 8.5 presents this component's interface.

Table 8.5: Navigation Sonar Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	Sonar Distances	Not decided	cont
Input	Mode	[ON, SINGLEPING, OFF]	
Output	Measured Sonar Distances	Not decided	

Also this component should be specified using the DEVDESS formalism.

8.3.5 Log

The Log component consists of two independent loggers. The Doppler log logs speed relative to the bottom, while the EM log logs speed relative to the sea. The two loggers also differ in that the EM log only measures speed along ship while the Doppler log measures speed in all directions (3D).

Figure 8.4 presents an overview of the Log component.

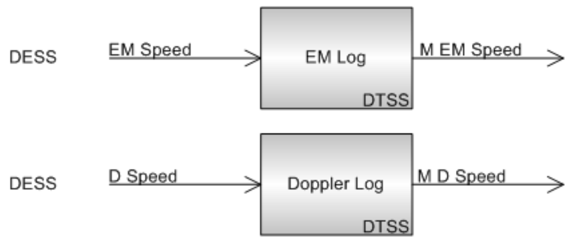


Figure 8.4: Log Model

Doppler Log

The Doppler log logs the submarine speed relative to the bottom. Table 8.6 presents an overview of the component interface.

Table 8.6: Doppler Log Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	Doppler Speed	[x,y,z] where x, y and z are [-25, 25] m/s	cont
Output	Measured Doppler Speed	[x,y,z] where x, y and z are [-25, 25] m/s	[1, 10] Hz

This component should be specified by the DTSS formalism. This is due to its discrete time nature.

EM Log

The EM Log measures the submarine speed relative to the surrounding sea in the forward-backward direction. Its interface is described in Table 8.7.

Table 8.7: EM Log Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	EM Speed	[-25, 25] m/s	cont
Output	Measured EM Speed	[-25, 25] m/s	[1, 10] Hz

Also this component should be described using the DTSS formalism.

8.3.6 Inertial Navigation

The Inertial Navigation component is responsible for logging information relative to the inertial forces. An overview of the Inertial Navigation component is given in Figure 8.5.

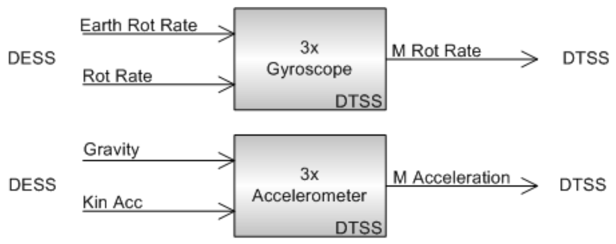


Figure 8.5: Inertial Navigation Model

Note that even though only one Inertial Navigation component is shown in Figure 8.3, the submarine actually contains two Inertial Navigation components. Also each Inertial Navigation component contains three accelerometers and three gyroscopes. Since these are equal only one of each is shown in the figure.

Note that their combined output creates a vector [x,y,z] describing acceleration and orientation relative to all three axes.

An example of a fully specified Inertial Navigation with sub-systems, given the current level of detail, can be found in Appendix A, Example A.1.4.

Gyroscope

The gyroscope is a component that measures the rotation rate of the submarine around one axis. For this reason three gyroscopes are present in the system, each one measuring one axis.

The interface of each Gyroscope is shown in Table 8.8.

Table 8.8: Gyroscope Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	Earth Rotation Rate	[-15.04, 15.04] deg/h	cont
Input	Rotation Rate (relative to the earth)	[-50, 50] deg/s	cont
Output	Measured Rotation Rate (relative to the inertial space)	[-50, 50] deg/s	Not decided

Although the input of this component is continuous, it is here chosen to specify these components with DTSS. It can be argued that both DEVDESS and DTSS is valid, but as the DESS to DTSS coupling is fully allowed there is no reason for making the component more complex than needed.

Accelerometer

The Accelerometer measures the acceleration of the submarine in one direction. Similarly to the Gyroscope component, also three accelerometers are needed to measure the acceleration in all three directions.

The interface of each component is shown in Table 8.9.

Table 8.9: Accelerometer Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	Gravity	[-1, 1] g	cont
Input	Kinematic Acceleration	[-0.5, 0.5] g	cont
Output	Measured Acceleration	[-1.5, 1.5] g	Not decided

Based upon the same grounds as with the Gyroscope, also these components should be specified by the DTSS formalism. Note that the Measured Acceleration is here Kinematic Acceleration - Gravity. Note that 1 g is approximately 9.81 m/s/s.

8.3.7 Pressure Sensor

The pressure sensor measures the pressure of the environment. As the pressure increase with the depth of the submarine, it is possible to calculate a depth based on this information.

Table 8.10 presents this components interface.

Table 8.10: Pressure Sensor Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	Pressure	>0 bar	cont
Output	Depth	>0 m	[1, 10] Hz

This component samples the pressure at given intervals. For this reason, it is chosen to specify the component with the DTSS formalism.

An example of a fully specified Pressure Sensor, given the current level of detail, can be found in Appendix A, Example A.1.2.

8.3.8 Navigation Processor

As previously mentioned the navigation processor is the main component in this system with the task of calculating the position, speed and orientation of the submarine.

This component is further decomposed, and a decomposed view is shown in Figure 8.6.

The navigation processor here consists of the inertial navigation algorithms, a Kalman filter and a terrain correlation component.

The inertial navigation algorithms dead reckons on the position, speed and orientation of the submarine based on the gyroscope and accelerometer measurements. Naturally, these values are approximations and will in time differ from the actual values. For this reason the Kalman filter is used with input from measurements

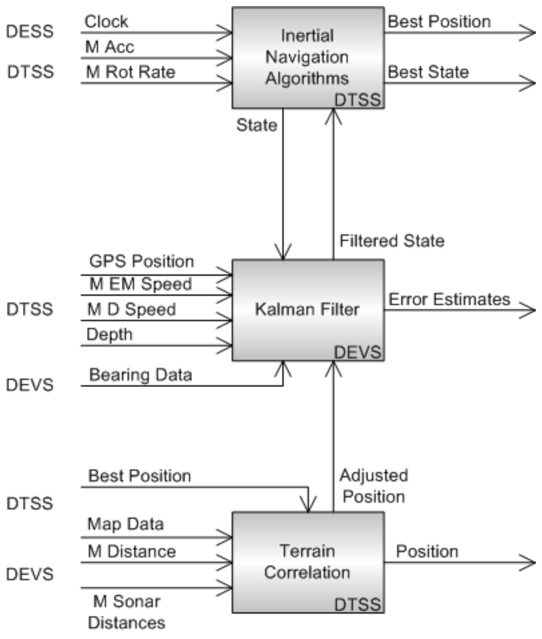


Figure 8.6: Navigation Processor Model

from other sensors, like depth and position, to reset or update these calculations. Terrain correlation can also be used to further improve this process.

The concept of *state* has been introduced to limit arrows in the diagram. State is a superset including both position, speed and orientation. See definitions of the respective definitions for value ranges.

Inertial Navigation Algorithms

The Inertial Navigation Algorithms component is the component which computes the best position, speed and orientation known in the system. The calculations are based on the previous known values of these parameters and measured acceleration and rotation rate of the ship.

Table 8.11 presents an overview of the interface of this component.

The component uses discrete time and updates at a regular frequency and should therefore be specified by the DTSS specification.

Kalman Filter

The Kalman filter is used to refine the calculations in the inertial navigation. This component receives data from a number of sources which can be used to get more precise approximations of position, speed and orientation. When necessary, for example if the error exceeds a given threshold, this component updates the inertial navigation algorithms with the filtered state.

The complete interface of this component is shown in Table 8.12.

This component is event driven, activated by receiving a position update, and should therefore be specified by the DEVS formalism. When the error of the state exceeds

Table 8.11: Inertial Navigation Algorithms Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	Measured Acceleration	[x,y,z] where x, y and z are [-1.5, 1.5] g and >0	
Input	Measured Rotation Rate	[x,y,z] where x, y and z are [-50, 50] deg/s and >0	
Input	Clock	>0	100 Hz
Input	Filtered State	As defined	
Output	State	As defined	100 Hz
Output	Best Position	[x,y,z] where x and y is given in degrees and z in m	100 Hz
Output	Best State	As defined	100 Hz

Table 8.12: Kalman Filter Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	State	As defined	
Input	GPS Position	[x,y,z] where x and y is given in degrees and z in m	
Input	Measured EM Speed	[-25, 25] m/s	
Input	Measured D Speed	[x,y,z] where x, y and z are [-25, 25] m/s	
Input	Depth	[0, 1000] m	
Input	Bearing Data	As defined	
Input	Adjusted Position	[x,y,z] where x and y is given in degrees and z in m	
Output	Filtered State	As defined	
Output	Error Estimates	Not decided	

a given threshold, Filtered State is sent to the Inertial Navigation Algorithms.

Note that error computations are done at a given frequency and for simplicity the Bearing Data is inserted directly to the Kalman Filter. Note also that there should be a cross-bearing process involved in this process.

Terrain Correlation

Also this component refines the position of the submarine. This is done through matching the ocean depth with map data. The interface is given in Table 8.13.

Table 8.13: Terrain Correlation Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	Best Position	[x,y,z] where x and y is given in degrees and z in m	
Input	Map Data	Not decided	
Input	Measured Sonar Distances	Not decided	
Input	Measured Distance	[0, 1000] m	
Output	Position	[x,y,z] where x and y is given in degrees and z in m	Not decided
Output	Adjusted Position	[x,y,z] where x and y is given in degrees and z in m	

Further, also this component is discrete. Position is sampled and used to retrieve Map Data from the Map component. Depth and map data is compared to possibly adjust the position. This behaviour is set to be run at a given low frequency and for this reason it should be specified by the DTSS formalism.

8.3.9 Bearing

The Bearing component is the component where the submarine crew manually can input bearings, in the form of a reference position and the bearing to this position relative to the submarine's heading.

The interface is described in Table 8.14.

Table 8.14: Bearing Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	Reference Position	[x,y,z] where x and y is given in degrees and z in m	
Input	Bearing	[0,360] deg	
Output	Bearing Data	As defined	

This component is event driven and should therefore be described using the DEVS formalism. Bearing Data is the union of Bearing and Reference Position and is forwarded to the Map component for visualisation and to the Navigation Processor for calculating cross-bearings.

8.3.10 Map

The Map component presents dynamic radar data and the submarine in a map environment. The submarine's position is based on the best approximation available. Also this component allows for forwarding map data to other components, as well as bearings can be inputted here.

An overview of the Map component is found in Figure 8.7.

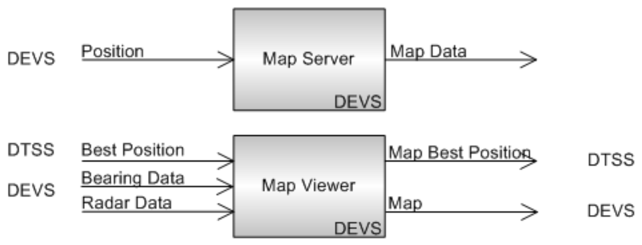


Figure 8.7: Map Model

Note that it is here assumed that both components have access to a common map database not shown here.

Map Server

The Map Server provides with the functionality for sending map data to other components based on the given position. The interface is described in Table 8.15.

Table 8.15: Map Server Data

I/O	Name	Value Range	Real-Time Constraints
Input	Position	[x,y,z] where x and y is given in degrees and z in m	
Output	Map Data	Not decided	

As this component is event driven, it should be specified by the DEVS formalism;

Map Viewer

The Map Viewer displays the submarine and bearing information in a map environment. In addition, this component also dead reckons on the position of the submarine.

The component's interface is described in Table 8.16.

Table 8.16: Map Viewer Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	Radar Data	Not decided	
Input	Best Position	[x,y,z] where x and y is given in degrees and z in m	
Input	Bearing Data	As decided	
Output	Map Best Position	[x,y,z] where x and y is given in degrees and z in m	Not decided
Output	Map	Not decided	

Also this component can be said to be event based. Updates in the display only happens when new data is available and the component should therefore be specified through the DEVS formalism.

Note that Map Best Position is calculated and updated at a given frequency.

8.3.11 Human

The Human component describes the behaviour made by a human in the loop. As a human plays an important part of the interplay in the system, it is here modelled as a separate component present in the system. This view is also suitable if the human's behaviour is to be simulated in stead of using a human in the loop.

With respect to using humans in the loop, the manner in which the input and output are transported differs. If the human is to be simulated, this data needs to be processed like all other signals in the simulation. However, if humans are in the loop, the signals will most likely take the form of visualisations and input created by human controllable devices, for example a joystick.

The interface towards the system is however described in Table 8.17.

Table 8.17: Human Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	Map	Not decided	
Input	Rudder	[w,x,y,z] where w, x, y and z are [-35, 35] deg	
Input	RPM	Not decided	
Output	Reference Position	[x,y,z] where x and y is given in degrees and z in m	
Output	Bearing	[0, 360] deg	
Output	Steering	[y,z] where y and z are [-1, 1]	
Output	Throttle		

Summing up, this behaviour includes both continuous and discrete event behaviour. If specified further, this component should be described using the DEVDESS specification. It is however unlikely that all possible actions made by this component will be describable. This implies that if this component is to be simulated, only a subset of the behaviour, linked to a specific scenario, should be specified.

8.3.12 Steering Console

The steering console allows for the crew to steer and control the submarine. Here, the engines rotational speed (RPM) is set and the positions of the rudders are controlled. This component also visualises rudder positions, engine rotational speed, speed, position and orientation (3D).

The steering console's interface is described in Table 8.18.

Table 8.18: Steering Console Data

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	Best State	As defined	
Input	Steering	[y, z] where y and z are [-1, 1]	
Input	Throttle	Not decided	
Output	Rudder	[w,x,y,z] where w, x, y and z are [-35, 35] deg	Not decided
Output	RPM	Not decided	Not decided

The Steering input is given by a vector [y,z] where y any z are normalized from -1 till 1 [2]. With this format, z is meant to refer to rotation around the z axis (heading), while the y is referring to rotation around the y axis (pitch). Based on this information the Steering Console calculates the rudder positions of the four rudders.

It can be argued that this component samples the continuous data provided from the Human component and should therefore be discrete. For this reason this component should also be specified by DTSS. An alternative is however to use DEVDESS if DEVS-like behaviour is preferred in the output.

Note that this is a limited version of the Steering Console and does not include automatic steering or order giving.

8.4 Comments

As can be seen in the above system description, the formats of some of the data are still to be determined. However, this will not have any great influence on the further study. Note however that in order to facilitate proper validation, input/output segment sets are needed to be studied, not only value range sets.

Further, this model description is based upon the assumption that the component providing output and the component receiving it has the same specification with regards to allowed data trajectories. For example that the Depth Finder produce a

[2]This done due to the fact that input devices usually use normalization. The input can however easily be changed to more appropriate numbers if needed.

depth between 0 and 1000 m which is the same as the Kalman Filter allows. In a final model of this system this is not necessarily given and a systematically matching will be needed to be performed.

An important aspect of this model is however that the continuous data flows and components described cannot be realized in a practical simulator. As mentioned in Chapter 5, this often involves using numerical integration resulting in discrete behaviour. It may for this reason be useful to create a lumped model of the system, taking this into consideration in addition to eventual other issues that need to be covered. This will however be the topic of Chapter 9.

Chapter 9 Requirements for Model Simulation

Creating an abstract model of the system to be simulated is only an intermediate step towards creating a practical simulator of the system. This chapter will focus on the process that needs to take place in creating the model with special focus on legacy components and technology from both a theoretical and practical view.

The term 'legacy' is here meant to refer to components and technology that are available to, but not created as part of the current project. This includes for example simulators created by other projects, existing middleware and other components of interest.

Currently, no legacy components are identified as part of the system to be simulated at FFI. However, if one expands the definition slightly, real world components can be argued to be included in this set. These are also components which need to be dealt with for a practical example of the system to be made.

For this reason, it is here chosen to study special requirements with regards to the real world component identified in the specified subset. With respect to legacy technologies, distributed computing middleware has been identified and will be discussed in the following.

The model described in Chapter 7 is based upon an ideal view of the system. This is however not necessarily realistic. Already by having continuous time systems in the model, issues start to occur since these cannot be simulated without loss of accuracy. In addition, when legacy components and technologies are included, the picture becomes even more complex.

This argues that a lumped model, as mentioned in Chapter 5, taking into consideration more of the simulation aspects, should be created.

This chapter will present examples of such lumped models taking into consideration the components and technologies indicated above. An overall goal of this work is identify special requirements with regards to these components and technologies, as well as to produce a model which might be an approximation of the model in Chapter 7, but which is closer to the model to be implemented as a simulator.

In doing so, an iterative approach will be employed. First, the case of using distributed computing middleware will be studied, before real world components are introduced. At the end of this chapter, final thoughts about creating the actual simulator will be discussed in addition to other special requirements and operating parameters. Special requirements with regards to other legacy components will alos be discussed in general.

9.1 Distributed Computing Middleware

As mentioned earlier, both the High Level Architecture (HLA) and the Common Object Request Broker Architecture(CORBA) have been proposed as middleware in the TEKULA project for the purpose of enabling message passing between computers in a distributed environment. In the DEVS-based literature, many examples combining the DEVS framework with both HLA and CORBA have been proposed in among other [37], [18], [18] and [13]. Also a framework based on GDEVS and HLA is presented in [32].

In this section, special requirements with regards to applying distributed middleware in this context will be discussed. Note that although having different services and properties, HLA and CORBA both share the message passing property which is the most important here.

9.1.1 Discrete Message Passing

As mentioned earlier, continuous time systems cannot be simulated without loss of accuracy due to the discrete nature of computer systems. In distributed computing this characteristic is strengthened. As all messages need to be sent and received among many computers the efficiency will most likely decrease, reducing the number of possible messages to send if timing constraints are present.

As demonstrated in Chapter 5, several approaches for embedding continuous time formalisms into DEVS are possible. Both discretizing and quantizing the continuous output are valid candidates.

Given the discrete nature of the system as described in Chapter 8 it can be argued that the choice of approach will not influence the resulting accuracy or number of messages sent much. However, if one considers the whole submarine it is highly probable that more continuous behaviour will be present. The input from the environment is an example of this.

For this reason, the choice is not insignificant. As mentioned in Chapter 5 however, it is still unclear whether GDEVS or QS should be preferred. Further, the architecture of the simulator and model might have great influence as well.

It is however chosen to use GDEVS in the lumped model presented in this section. The rationale for this is that GDEVS promises the greatest reduction in number of messages needed to simulate the continuous behaviour. It should also be noted that the relationship between these formalisms should be studied further.

9.1.2 Parallelism

Distributed processing allows for exploitation of parallelism which can greatly improve the simulation efficiency. The standard DEVS formalism is however based on serializing. So, to exploit this feature the model will need to be modified to a formalism allowing this behaviour.

The Parallel DEVS formalism described in Chapter 5 is one such formalism. This is an extension of DEVS and since DTSS, DESS and DEVDESS can be embedded in

DEVS, it is possible to allow parallelism between all of models in a multi-formalism system.

With respect to the previous discussion of discrete messages and GDEVS, it may however be interesting to create a parallel version of GDEVS for this purpose. Note that, GDEVS only differs from DEVS in the format of the messages sent. A parallel version of the GDEVS should therefore be created.

Further, parallelism presents requirements with regard to synchronization. Synchronization is needed to ensure that the causality between events in the system is correct. An overview of relevant parallel protocols for this purpose can be found in [38].

9.1.3 Operating Parameters

Adopting the use of distributed middleware should not affect the operating parameters of the system. However, it can be mentioned that the need for discretization is made explicit here and strategies for reducing message traffic might be needed. Note that this is more connected to the form of the parameter than the range of values.

However, facilitating interoperability is often a goal in using distributed middleware and this might place some requirements on the operating parameters, the experimental frame, of the system.

An example of this is the adoption of the Real-time Platform-level Reference FOM (RPR-FOM) standard with regards to HLA [12]. This standard defines the communication between the different components of the simulator through specifying objects, attributes, interactions and parameters allowed.

Particularly, the RPR-FOM defines an object called Submarine Vessel with a defined set of attributes including position, velocity and orientation. Although the details of this object will not be discussed here, adopting this standardised FOM will allow communication with other HLA federates in the federation, for example VR-Forces and the 3D Stealth Viewer from MÄK Technologies [25].

Adopting a standard like RPR-FOM requires that the input and output of the system is linked to the concepts defined in this standard. A facade, translating between the different message types might be needed [1], but the issue in itself also implies that the experimental frame of the system needs to be adapted to allow for the data needed by the standard [2].

Further, by using distributed middleware additional control signals, for example synchronising signals, may have to be included in the experimental frame. This is however dependent on the middleware in question.

[1]For example, translating a GDEVS defined function to a RPR-FOM defined interaction parameter value

[2]Note that a mapping between couplings and states in DEVS and respectively interactions and objects in HLA has been suggested in [23].

9.1.4 Lumped Model

Based on the requirements described above, a lumped model of the analysed system is presented in Figure 9.1:

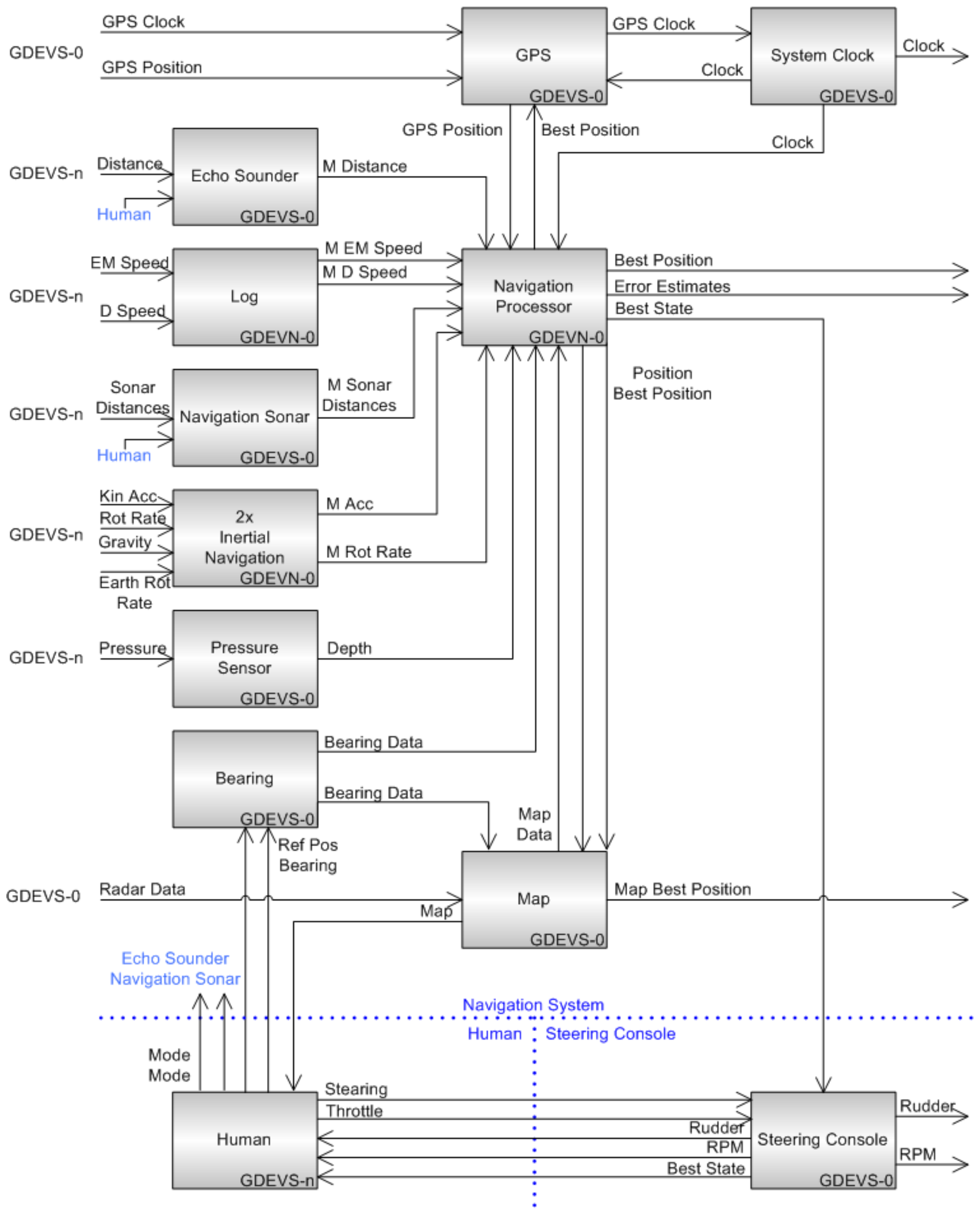


Figure 9.1: Lumped Model

As can be seen here, all models are here specified by the GDEVS formalism, and its coupled version, which allows a unified way of specifying the components [3]. Note that order of GDEVS system is shown as n in GDEVS-n.

[3]QS could also have been used here, resulting in a all DEVS model. PDEVS could then have been exploited to achieve parallelism.

Further, the lumped model is similar to the model given in Chapter 7 and for simplicity no additional input or output ports are added. This indicates that using a distributed middleware is largely independent of the model. This is of course as it should, since the behaviour of the model is independent of how the messages are transmitted.

Notice however that the mapping between models and computers has not been conducted. Such a mapping could be visualised in the model as coupled models, however it would have no affect on the behaviour of the system. Also note that in literature a flattening approach is suggested for this purpose, reducing the hierarchy and at the same time messages needed [8].

An example of a fully specified GDEVS components, given the current level of detail, can also be found in Appendix A, Example A.2.2 and A.2.1.

Organization

The distributed simulator can be organized in many ways. For example, it may be possible to use one middleware technology in one part of the structure and a different technology in another part. Another example is to use a facade between the combat system simulator and the external world where HLA is used in this communication and CORBA is used internally in the combat system simulator.

Although this structure can be formed in any way, using multiple layers may provide additional overhead to the system which may in turn affect simulation efficiency. In any organization of the simulator, the real-time constraints specified will need to be met, reducing the options.

With regards to the final organization it can therefore be argued that both model complexity, computational power and real-time constraints need to be taken into consideration before a decision is made.

9.2 Real World Components

The system to be built is envisaged as a hybrid system, where both simulators and real components interact. Humans, the system clock, the steering console, the map component and the GPS have for this purpose been identified as potential real world components.

Different types of simulations might introduce different requirements and constraints upon the simulation system. Four distinct types/scenarios can be envisaged. These are:

- Only computer simulated components
- Human in the loop
- Hardware in the loop
- Human and hardware in the loop

Below, special requirements related to these scenarios will be discussed.

9.2.1 Representation, Validation and Verification

Real world components are by definition source system components. When using such components in the simulator, it can be argued that they in a way become their own simulators. Figure 9.2 shows how this is envisaged.

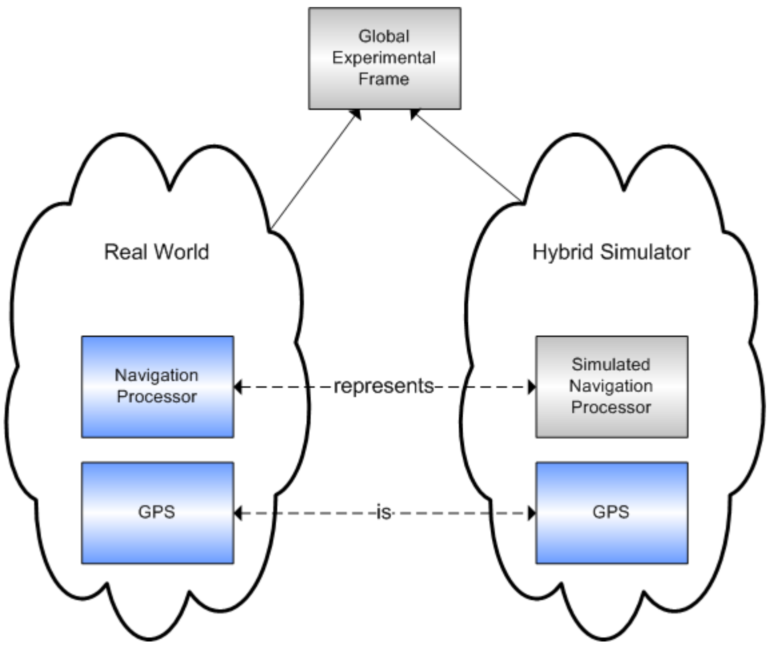


Figure 9.2: Hybrid System

As can be seen here, both sides are meant to represent the same system seen in context with the experimental frame. The hybrid system however includes both real and simulated components which have slightly different relations to the real world. For example, it can be argued that the 'is' relation is stronger than the 'represents' relation. That is, the simulated component does not necessarily need to be 100 % equal to its real world counterpart. A real world component however is.

A natural question is: is there need for modelling these components? As one shall not create a simulator, it can be argued that specifying the inner structure is not needed. However, the behaviour could be argued to be useful with regards to applicability and checking consistency in regards to other components in the system, implying a specification at level 0-1.

There will however be a requirement for an interface component with regards to using real components in a simulated system. This is due to the need to transform the real input and output to be applicable within the simulator. For this reason, it can argued that the composed system, including real world component and interface might need to be verified at a global level.

Finally, it can also be argued that all components interacting with real humans may need additional validation simply due to man's unique ability to make mistakes.

9.2.2 Simulation Time

Standard simulation often deals with virtual time. That is, the time in the simulation is completely independent of wall clock time and can be both faster than and slower than wall clock time, as well as unequal time steps can be used. Adopting real world components in the simulator might however not allow such a loosely defined time.

Table 9.1 presents an overview of how the different time regimes are supported in the different scenarios. 'VT' is here meant to indicate *virtual time*, while 'SRT' is *soft real-time*, here defined as allowing only small time discrepancies with regards to wall clock time, and 'HRT' is *hard real-time*, where the simulation time needs to be synchronized precisely to wall clock time. Note that this usually is realized by using deadlines which all must be met.

Table 9.1: Scenarios versus Time Regimes

Scenario	**VT**	**SRT**	**HRT**
Computer Simulated	YES	YES	YES
Human in the Loop	NO	YES	YES
Hardware in the Loop	?	?	YES
Human and Hardware in the Loop	NO	?	YES

As can be seen here, when all components in the simulator are computer simulated, simulation time can adapted almost freely. That is, the simulation time can be both faster and slower than wall clock time. Note that an upper bound is present related to computational power. The simulation cannot go faster than what is computational feasible given the hardware and the architecture. Note also that the rate of the time can differ throughout the simulation.

Humans and real hardware on the other hand are not as flexible. Introducing humans in the loop implies that that the system as seen from the humans view must follow wall clock or near wall clock time. Small inaccuracies may however not matter much as humans generally adapt quickly. The inaccuracy can however not become so large that the person or persons loose the sense of realism.

Having hardware in the loop, including both hardware in the loop and hardware and human in the loop, is often the most constraining scenario. How real hardware will react to too slow or too fast input, as well as how the rest of the simulation handles if there these components are slower/faster than simulation time, differ. In most cases it is likely that the hardware will malfunction or the simulation will become corrupt. In this case, the simulation time needs to be equal to the wall clock time.

Note however that not all hardware components require hard real-time simulation. Many systems' behaviour can be seen as independent of time allowing also virtual or soft real-time simulation time. This if for example typical in client-server architectures and often are such systems specified by a DEVS formalisms.

Table 9.2 shows how the four hardware components suggested to be included in the final simulator of the chosen subset deal with time.

As can be seen here, it can be argued that both the GPS and the System Clock only handles hard real-time simulation. The rationale for this is that both components outputs signals at fixed time intervals. If the rest of simulation followed a different clock, these signals would become out of synch.

Further, the Map component shows sign of the client-server architecture indicating

Table 9.2: Real World Component and Time Regimes

Component	**VT**	**SRT**	**HRT**
GPS	NO	NO	YES
System Clock	NO	NO	YES
Map	?	?	YES
Steering Console	YES	YES	YES

no constraints. However, the Map Best Position is outputted at a fixed frequency. For this reason this component is hard real-time dependent if this Map Best Position is used in the system.

Finally, the Steering Console in its limited version here only process input and sends it directly out, which can be argued to be time independent behaviour. For this reason it allows all types of simulation times. Note however that there might be requirements on the receiving part of this output not covered here.

Further, it is clear that having a real world steering console makes little sense without a real world human. This will of course possibly remove the option of virtual time.

9.2.3 Couplings and Signal Timing

Above, simulation time is discussed with regards to having real world components in the loop. A question then arises with regards to how this affects the couplings and signal timing between components.

As real world components are the only components which absolutely need to be in real-time, it can be argued that as long as all interaction with these components meet their deadlines, other components may not. Figure 9.3 presents three different scenarios, where two components are coupled together and sequentially treat a given input. d1 and d2 are deadlines of the respective components.

The scenarios here presented show three types of systems. That is, systems meeting their deadlines perfectly, systems that are too fast and systems that are too slow.

The first scenario, to the left, shows a situation with a fast and a timely system. As can be seen here, the Pressure Sensor is finished processing ahead of its deadline and has to wait for the proper time before its output is legal. The Navigation Processor, then takes over and does its processing in time for deadline 2.

In the centre scenario however, the Pressure Sensor is a slow system which is not able to finish on time. Further, the Navigation Processor is timely, but will also miss its deadline.

The third scenario also has a slow Pressure Sensor, but here the Navigation Processor is fast enough to make up for this delay, making sure that deadline 2 is met.

As shown here, both the first and the third scenario presents systems which global behaviour is satisfactory with regards to real-time simulation. Although the inner deadline is not met in the latter scenario, the outer is.

As it is always possible to delay too fast components the first situation here will never create any real issues. The third scenario however, is not without challenges.

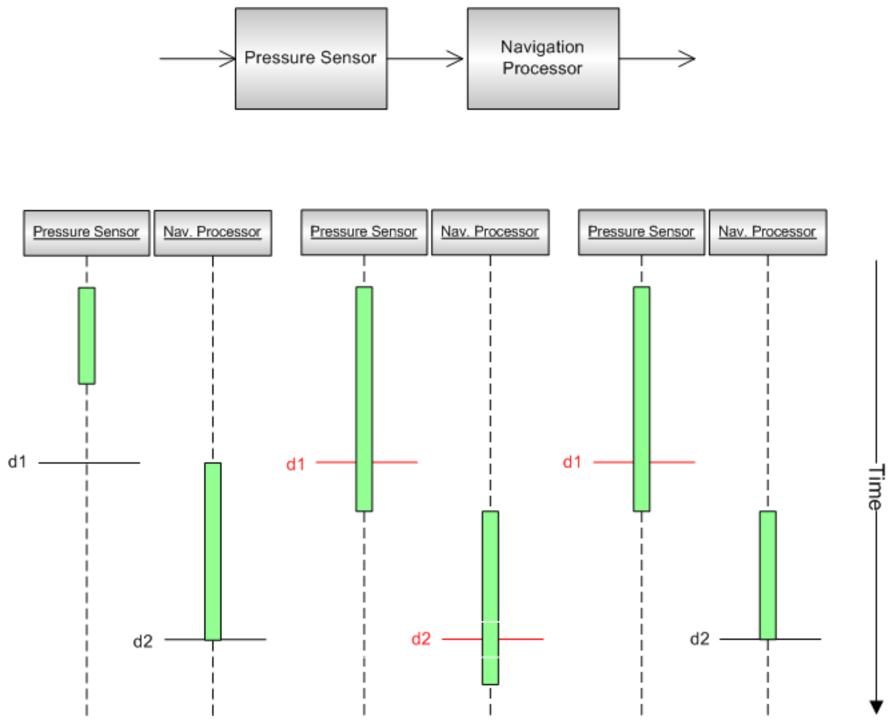


Figure 9.3: Real-Time Scenarios

The behaviour shown here is very simple. What happens if each component repeats its action in a cycle? If this was the case, it can be shown that if the component consistently spend more time than specified in each state, the delay will accumulate making the scenario impossible.

However, if both components share the same update frequency, so that for each state transition in the Pressure Sensor, the Navigation Processor also makes a state transition, the issue can be solved by adjusting deadline 1.

Unfortunately, it may seem that most of the components in the modelled system, including the Pressure Sensor and the Navigation Processor, have a behaviour that repeats itself in a loop, disallowing slow systems. Also many of the systems in the model are squeezed between real world components, further strengthening this requirement.

Also note that there might be time discrepancies due to processing of state transition functions, output functions and message passing. Especially in distributed environments this would become significant. Figure 9.4 shows an example of a hard real-time component between two simulated components and how this behaviour can be envisaged.

Notice here, that it is important that not only the output of the Navigation Processor is available, but that it is available to the real world component, the Map. This again argues that the message needs to be sent from the Navigation Processor no later than equal to the delay due to distributed message passing [4]. Further, the output of the real world component will arrive at the Human too late. As long, as the Human does not miss its deadline this could however be argued to be allowed.

[4]Here, messages need to be time stamped so that receiving component know when the input is legal.

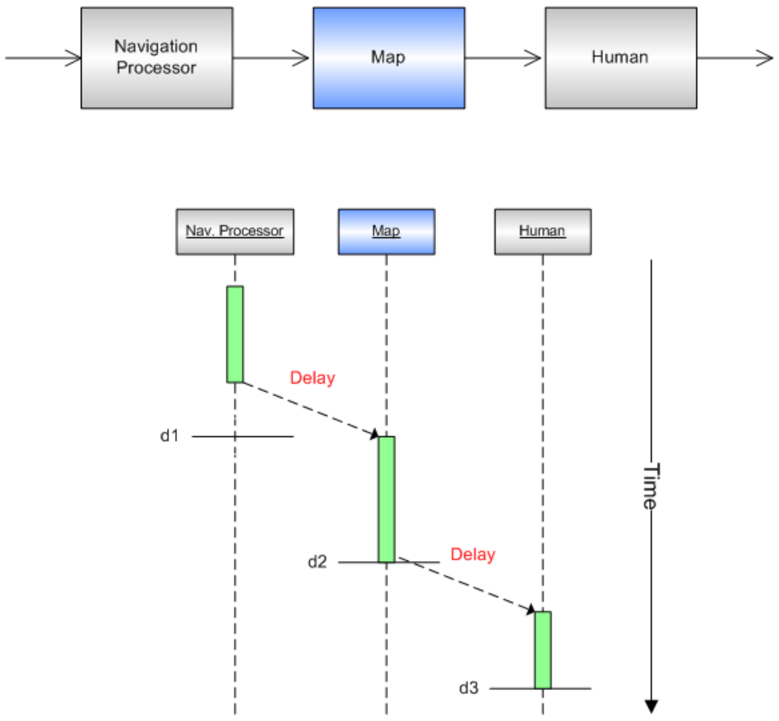


Figure 9.4: Real-Time Simulation and Delays

This can however be argued to be the responsibility of the real-time protocol used.

A final requirement in this regard is that two consecutive real world components, or two simulated components which exactly meet their deadline, cannot communicate if such delays are present. This argues for using real communication lines between the real world components. With regards to the simulated case, this needs to be dealt with through modification of the models or increased processing power.

In the model provided in Chapter 8 both the GPS and System Clock as well as the Human and the Steering Console communicate with each other directly. This communication should therefore follow the real communication lines as suggested above [5].

9.2.4 Operating Parameters

Also the experimental frame, and hence the operating parameters, is affected by the type of simulator chosen.

Naturally, one might argue that the operating parameters reflects the real operating parameters of the system and inserting real world components therefore should not affect these values. However, this need not necessarily be true. For example, one might imagine a simulation of malfunctioning components where the values from a given component are not normal.

[5]With regards to the communication between the Human and the Steering Console this naturally implies visualization and input through real devices.

Using real hardware in the loop might for this reason reduce the scope of scenarios that can be simulated since their behaviour is limited to normal behaviour. Of course, it is possible to use faulty hardware in the loop. However, it is unlikely that there exist malfunctioning hardware components for all possible scenarios.

On the other hand, simulated components do not have this limitation with regards to adaptation. Models and simulators can more easily be changed to suite the needs. Note however that the simulators need to be well validated and verified.

Further, allowing humans in the loop has some of the same properties as using simulated components. The reason for this is that given the proper interface, human behaviour is adaptable too.

Real-World GPS

Using a real-world GPS presents an issue with regard to the experimental frame. The reason for this is that a real sensor must be stimulated by real input. Picture the following; The simulator and its real-world GPS is located at FFI while simulating a submerged submarine somewhere on the west coast. Naturally, the data from the GPS would not reflect the simulation scenario studied.

It can however be argued that the GPS could be located onboard a real submarine. However, there would still be an issue with regards to synchronising the synthetic world in the simulation to the real-world seen through the GPS.

For this reason it is here recommended to only use simulated sensors. This would ensure that the environment seen from the submarine simulation is consistent and flexible to any scenario envisaged.

9.2.5 Areas of Usage

Due to the different properties with each scenario as described above, different areas of usage might be more suitable with a given type of simulator than others. An overview is given in Table 9.3.

Table 9.3: Areas of Usage

Areas of Usage	**Only Simulated**	**Human**	**Hardware**	**Human and Hardware**
Experimentation	YES	?	?	?
SBA	YES	?	?	?
Training	NO	YES	NO	YES
Demonstrator	YES	?	?	?
System Integration	YES	YES	YES	YES
Stress Testing	YES	YES	?	?

With the case of experimentation, only using a fully simulated simulator enables full control of the experimentation. That is, including humans and hardware in the loop might limit the scenarios under which the system can be experimented with. On the other hand, introducing these objects might also increase the credibility

Simulation-based acquisition is similar to experimentation and, hence, has the same restrictions. Training however differs slightly. The reason for this is that training can only take place where humans are present.

Using the simulator as demonstrator also has similar restrictions as experimentation. The limitation here becomes that with humans or hardware in the loop, simulations need to be in real-time.

System integration and testing scenarios can be achieved by any simulator. The question is here more what component is being tested. This component can be both simulated and real world component.

Also in stress testing limitations occur. The reason for this is that real hardware has limited behaviour. That is, the behaviour is always correct and it may be difficult to test with behaviour that is outside the range of normal behaviour.

9.2.6 Lumped Hybrid Model

To distinguish between the different types of simulator components the notation used needs to be extended. Figure 9.5 shows an overview of the new notation with colour coding of the system components.

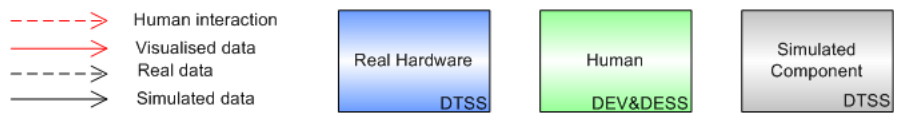


Figure 9.5: Real Hardware Notation

As can be seen here, it is chosen to distinguish between simulated components, real hardware and humans. Although both humans and real hardware can be argued to be real world components they have slightly different properties.

In addition, it is chosen to distinguish between four types of data flows. These are real data flows, simulated data flows produced from simulations, human interaction data flows which refers to signals produced by humans and visualised signals also used in combination with human components.

Taking into the account the components that are to be simulated by their real world counterparts, Figure 9.6 presents an overview of a lumped hybrid model which also indicate what type of simulator that should be used.

As can be seen here, the human component is here depicted to be simulated by a real human, and the Map component, the Steering Console, the System Clock and the GPS by real hardware.

The influence of having real world components in the system is also shown through the different types of data flows. Note that real data can be argued to be free of simulation error, but at the same time need interfaces which should be described as part of the simulator. Further, these are also the couplings which might induce real-time constraints.

The issues related to simulation time is however not covered in this abstract lumped model implying that a supervisory real-time control is chosen. Although one could perhaps include delay boxes along all couplings, this could be argued to be an issue for the simulator. Note that RTDEVS could also be a possibility here.

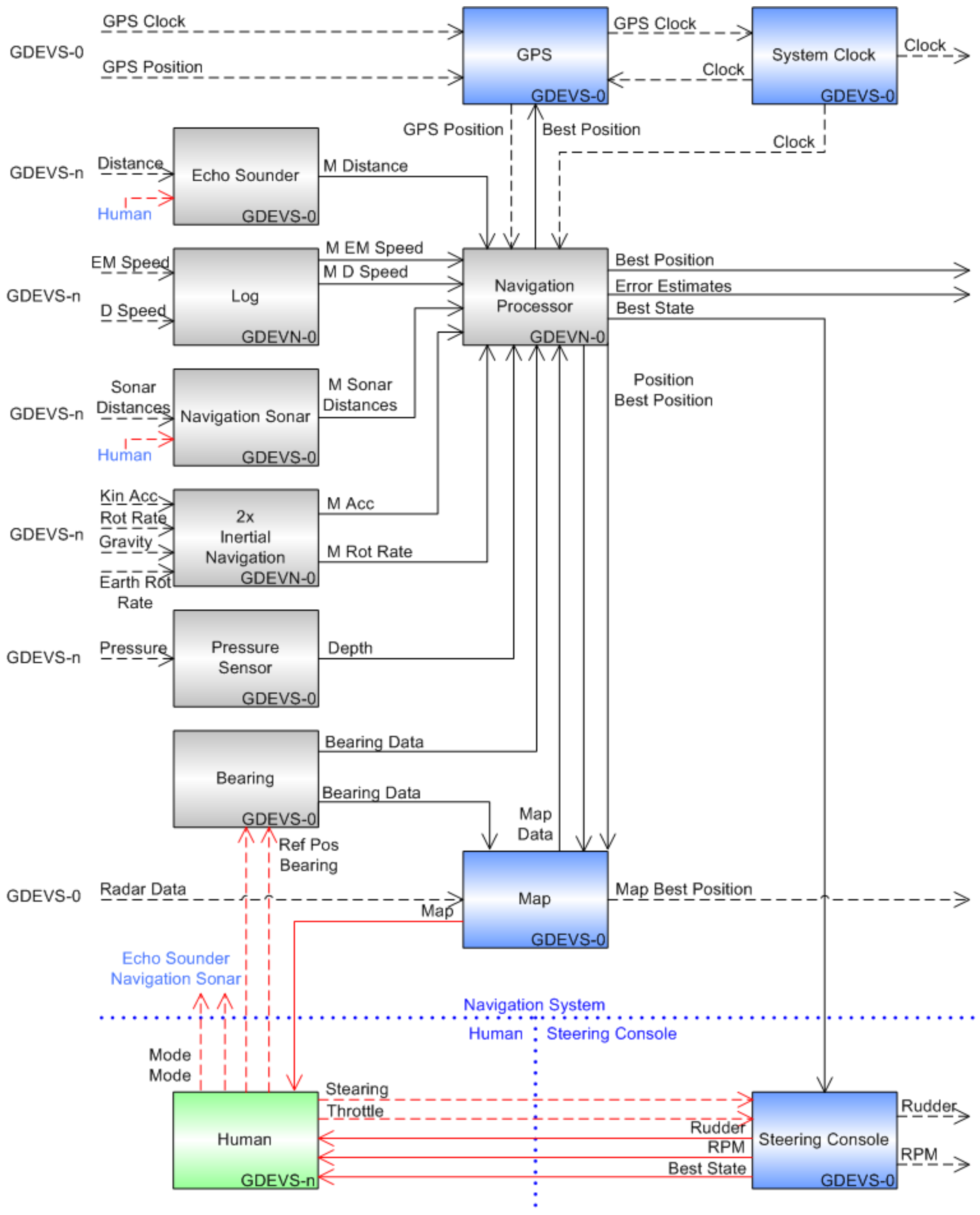


Figure 9.6: Lumped Hybrid Model

Further, with regards to previous section, that real-time simulation requires that the middleware supports real-time execution. More on real-time distributed middleware can be found in [16].

Finally, note that for simplicity reasons, the operating parameters of the real world components are unchanged.

9.3 Discussion

As shown above, legacy components and technologies may have many influences on the simulator to be built. But, even if these restrictions were not present, the process of creating the simulator would not be trivial. The reason for this is that inherent in the model itself, several issues may be lie dormant.

For example since FFI aims to simulate the specified system with 100% accuracy, all systems need to be specified at a high level of detail. This implies that the behaviour of the simulations need to be equal to or as close as possible to the behaviour of the real world system. Naturally, this is an optimistic goal and it will most likely affect many of the choices made.

Further, the time scale of the different components differ arguing for a DEVS-based approach. The highest update frequency is however at 100Hz. Combined with the real-time requirements, this implies that the simulator needs to conform to this requirements with regards to both timeliness and efficiency.

Finally, also the simulator needs to be able to handle the complexity of the model. As specified in the model in Chapter 7, the Navigation Processor will have a large number of input ports. The Navigation Processor simulator will therefore need to handle updates of all of these ports simultaneously and within the scope of the real-time constraints. Questions like whether the component is too complex for single machine processing, whether it will it slow down the simulation too much and what might happen when data on all inputs arrive simultaneous, must be dealt with in order to appropriately simulate this component.

9.3.1 Operating Parameters

The operating parameters are closely linked to the experimental frame. The model description made in Chapter 7 describes the value ranges of the normal behaviour of each component. Together these can be argued to represent the normal operating parameters of the system. Table 9.4 presents an overview of these parameters.

Issues with regards to computational complexity and real-time constraints might however influence these values. Simulator architecture and processing power here becomes important. For example it might not be feasible to process many distributed events in 100 Hz and at the same time use real world components in the loop.

Table 9.4: Operating Parameters

I/O	**Name**	**Value Range**	**Real-Time Constraints**
Input	GPS Clock	>0	
Input	GPS Position	[x,y,z] where x, y and z are given in m	
Input	Doppler Speed	[x,y,z] where x, y and z are [-25, 25] m/s	cont
Input	EM Speed	[-25, 25] m/s	cont
Input	Sonar Distances	Not decided	cont
Input	Kinematic Acceleration	[-0.5, 0.5] g	cont
Input	Rotation Rate	[-50, 50] deg/s	cont
Input	Earth Rotation Rate	[-15.04, 15.04] deg/h	cont
Input	Gravity	[-1, 1] g	cont
Input	Pressure	>0 bar	cont
Input	Distance	[0, 1000] m	cont
Input	Radar Data	Not decided	
Output	Clock	>0	100Hz
Output	Best Position	[x,y,z] where x, y and z are given in m	100Hz
Output	Map Best Position	[x,y,z] where x, y and z are given in m	Not decided
Output	RPM	Not decided	Not decided
Output	Rudder	[w,x,y,z] where w, x, y and z are [-35, 35] deg	Not decided
Output	Error Estimates	Not decided	

9.3.2 Other Legacy Components

Although not studied here, other legacy components can range from models from previous projects to commercial products and is closely related to the topic of reuse.

Two special requirements in this regard stand out. The first is that the legacy component needs to sufficiently represent the intended system. Secondly, the component needs to conform to the coupling requirements given, that is, it needs to be applicable to the rest of the simulator system.

With regards to DEVS, both of these issues can be treated provided that a specification at the appropriate level is created of the component. The first, can be studied as a morphism while the second is deals with the applicability relation.

Further, some of the requirements related to real world components may also be valid for these components. For example, there might be a need for using an interface or wrapper to allow communication between the legacy component and the simulation to take place. For example, the component of interest can use a different syntax or be specially tailored to work within a different environment. A interface component will then be needed to translate between this component and the rest of the simulator system.

Further, limitations in operating parameters as well as time constraints can also be adopted due to limitations within the legacy component. Both of which need to be thoroughly studied before the component is used.

Chapter 10 Analysis and Simulation Credibility

Based on the experiences made in analysing the specified system, as well as the review of related topics found in literature, the usefulness of DEVS with regards to the analysis of this subset as well as establishing the simulation credibility will here be explored. In doing so, the simulation aspects presented in Chapter 7 will act as a starting point of this discussion.

At the end of this chapter, a short discussion about theory and practice will be conducted, before lessons learned and a view on scale-up is presented.

10.1 Representative Specifications

As stated earlier, the DEVS-based theory provides a rigorous foundation, rooted in mathematical system theory, for specifying systems. Special concepts and formalisms are further created for this purpose. However, is it fair to say that DEVS allows for specifying all envisaged systems in an acceptable manner?

Naturally this is difficult to answer and probably also a subjective issue. However, as indicated in the model and the examples of fully specified components given in Appendix A, the subset of formalisms studied in this thesis allow for presenting a general description of the subset studied.

A key observation with regards to this subset is however that most of the components envisaged are discrete in nature. This simplifies the specification task greatly as DEVS provides with an universal way of specifying discrete event systems. Further, as mentioned in Chapter 5, discrete systems can be represented and simulated without error.

The subset is however not without continuous components. As can be seen in the model in Chapter 8, both the Human component and much of the input from the Experimental Frame is continuous. In DEVS this behaviour is usually specified by a DESS or DEVDESS formalism.

Although it is unknown if all continuous systems can be specified by these formalisms, a greater concern is that this behaviour cannot be executed on a computer. For this reason, DEVS provides with several alternatives including Quantized Systems and Generalized DEVS, see Figure 10.1. A significant aspect of these alternatives is that the error introduced by the simplification towards discrete behaviour can be studied and controlled.

Further, with respect to the subset studied, two issues can be worth mentioning. The first is related to the Human component. Although this component can be argued to have both a continuous and discrete time base, the behaviour is non-deterministic.

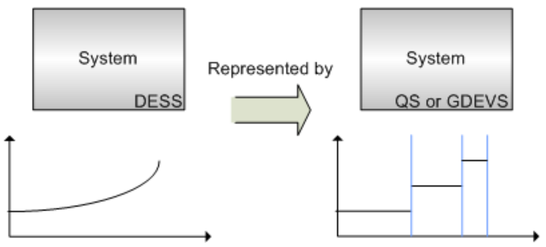


Figure 10.1: DESS Represented by QS or GDEVS

Capturing all of the behaviour possible by this component will most likely never be possible. However, limited scenarios may be specified by the DEVDESS formalism [1].

Further, the Navigation Processor was specified by a modular MFN formalism. It can however be argued that a non-modular formalism would be more appropriate for this purpose due to the shared variables between its inner components. Note however that this would only simplify the specification, using a modular approach as done here is also valid.

Summing up, it can be argued that DEVS do allow for the creation of a representative specification of the envisaged system. All components were accounted for and it was possible to associate a formalism to each envisaged component. Whether these specifications are representable in respect to the source system is however a topic related to validation and will be discussed later.

10.2 Complexity and Maintenance

With regards to dealing with complexity, the DEVS-based theory provides with a number of options. First, provided that an experimental frame has been defined, scope has been formally established helping to focus the modelling and simulation task. The separation of model and simulator, as shown in Chapter 4, also provides a separation of concerns by allowing one to focus on one task, one part of the problem, at the time.

Both models and simulators can here be created by domain experts, further increasing the probability of success. That is, ideally the model can be specified by someone who is an expert on the topic, while the simulator can be implemented by a computer scientist. This does however assume that the expert is familiar with the DEVS notation. For this reason, a graph notation as shortly presented in Chapter 5 should be explored. Note that the notation should however be extended to allow multi-formalism modelling if possible.

Further, the modular and hierarchical nature of the formalisms in DEVS, as shown in Chapter 8, allows for a divide and conquer strategy which in turn allows for splitting the system of interest up into manageable parts. It can even be argued that DEVS forces the modeller into this way of thinking, as only limited functionality can be included into a basic model without making it look complex, see examples in Appendix A.

Rapid model development and easy maintenance and modification of models are

[1] To accommodate these uncertainties, perhaps Fuzzy DEVS can be applied [38]. This formalism is however not studied here.

also needed to support calibration and exploration of new concepts. The longer it takes to modify a model, the fewer combinations can be experimented with. In this regard the separation between model and simulator made in DEVS becomes useful again. The model description allows for easy analysis and modification of the component as only the model behaviour is specified in the formalism. That is, the complex logic of the simulator is not present. Further, if an abstract simulator engine is used, no additional development is needed. The model can simply be inserted into the engine to run, provided that the model class is supported by the engine.

10.3 Composability

As previously mentioned, it can be argued that DEVS lends itself towards modular descriptions which can be composed into hierarchical systems. The system theory foundations, and in particular the closure under coupling property, ensure that these couplings are correct. However as stressed in literature, real composability is hard to achieve [5]. Often even modular systems only work together if all components were specially developed to fit together.

The reason for this is for example that the syntax used to describe the data is different, different granularity can be used, different value ranges or trajectories can be allowed and the scope, given by he input/output ports, of the components might differ.

As shown in Chapter 8, DEVS unambiguously specifies the input/output ports of each component, as well as the legal value ranges and trajectories in this regard allowing formal analysis of the relation between two components [2]. Analysing the applicability relation between two components can therefore be achieved in a formal manner.

This is closely related to auditing self-consistency. Through the formal specifications of DEVS, one can systematically match output and input between components. For example in the model provided in Chapter 8, there is a need for matching the possible output from the Doppler Log to the allowed input to the Kalman Filter to ensure correct behaviour, see Figure 10.2.

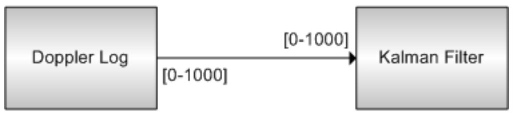


Figure 10.2: Auditing Self-Consistency

Although not shown in the figure, the first criteria for comparison is naturally if the ports, or more correctly the data, match each other. If the first system outputs position and the second accepts speed, there is not likely to be a semantically correct coupling.

Given that the data outputted by the Doppler Log is semantically equal [3] to the data accepted at the Kalman Filter, the matching of value ranges and trajectories can be done.

[2]Note that trajectories are not covered in the architectural description given in this thesis. However, provided that a higher level specification is created, it is always possible to infer a lower one. This ensures that given a fully specified component, one can always infer the trajectories.

[3]Need not necessarily be the same number of ports. A wrapper can be used to aggregate or split up complex data types.

It is however not always the case that the value ranges or trajectories need to be equal. Although one component can be more 'capable' than the other, simulations can still be completed, but with a limited experimental frame.

Figure 10.3 presents an example of this situation where System 1 only sends its input out again. As can be seen, the composed input is limited by the System 2.

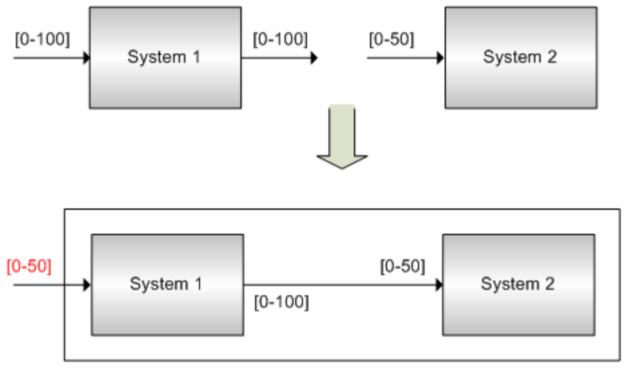


Figure 10.3: Auditing Self-Consistency with Unequal Value Ranges

In some cases however this matching will only provide an indication towards whether the coupling holds. The reason for this is that in the end it is the trajectories and not only the values ranges which determine if the coupling between the components is sound. A similar matching can also be done with trajectories to ensure consistency. Note however that it in some cases might be unfeasible to match all possible trajectories, especially when dealing with continuous systems.

Note that also legacy components can be analysed in a similar fashion, provided that proper specifications are created.

10.4 Legacy Components and Technologies

Chapter 9 introduced legacy components and technologies into the modelled system, resulting in lumped model being created.

Although concepts from the DEVS-based literature were used in discussing some of the requirements found in the specified subset, the discussions indicate that many of the topics could not be coupled to DEVS. For example, the lumped model taking into consideration a distributed computing middleware only takes into consideration the discretization which was needed even without such a middleware present. Parallelism could also be introduced by for example using the PDEVS formalism. However, this is not necessarily a property of the middleware either.

Real world components introduced issues with regards to simulation time and real-time constraints. This, as well as the delays due to the middleware now becoming visible, is not shown in DEVS either. For this reason a colour coding was used in the lumped model to identify type of real world component as well as real world data.

The reason for these aspect not being covered can be argued to be that both the distributed computing middleware and the real-world components is part of the simulator realm and not the model. In such a mapping, both of these issues would have to be studied as part of the simulator relation during verification. Although this can be argued to be fair, it may also lead to less control of the aspects.

The RTDEVS formalisms, described in Chapter 5 is an approach aiming at bridging this issue, at least with regards to the real-time constraints. This formalism could have been applied in the lumped model, but was at the time chosen not to.

It is at this time unclear which approach is the better. Today's formalisms can be argued to not include all the aspects of interest, but developing new formalisms, including more of the functionality of the simulator, might relax the separation between model and simulator and, hence, the separation of concerns.

As suggested in 5, the lumped model approach could be an alternative in this regard. While having one base model to be the abstract representation of the source system, the lumped model could function as a abstract representation of the simulator, including many of the issues mentioned above. Ultimately, this could result in a new step with regards to the separation of concerns. However, this topic needs to be studied in detail due to the importance of the separation of concerns and will therefore be a natural further study.

10.5 Simulation Credibility

As stated in [38], much of the work related to modelling and simulation is related to studying the relations between the different entities. Due to the formalness of DEVS this might also be the place at which the DEVS-based literature is most useful.

DEVS specifies the dynamics of a system in an unambiguous fashion which can be argued to be a sound foundation to build credibility on. In Chapter 6 validation and verification were shown to be important in this regard, relating to the modelling and simulation relation in the modelling and simulation framework.

Figure 10.4 presents a more complete overview of the framework including the associated specifications, relations, possible errors and the lumped model. Note that the Frame specifications, see Chapter 4, can be outlined in a similar fashion.

In establishing the credibility of a simulator, the relation here called Main Relation can be argued to be the one which is the most important, comparing the source system to the simulator. DEVS however has introduced the separation of model and simulator and allows for studying this Main Relation through the modelling and simulation relations shown in the figure.

The reason for doing this is that one can focus at certain aspects of the simulation at each stage. For example, with the modelling relation one is only concerned with how well the model represents the source system, not how it is later implemented. Further, this separation might also allow us to identify where possible inequalities are more precisely. For this reason, also the error associated with the different relations are marked in the figure.

It has been suggested in this thesis that the model be divided into a base model and a lumped model. The rationale for this is that this allows for focusing on representing the source system as exactly as possible in the base model, while in the lumped model's focus is on adapting this model towards simulation. Note that this further allows for studying the error, here called Lumped Error, which can be introduced in this transformation for example due to discretization or non-determinism due to real-time constraints.

As mentioned in Chapter 6, DEVS allows for studying these relations through abstract morphisms at the appropriate levels. Through these morphisms one can argue

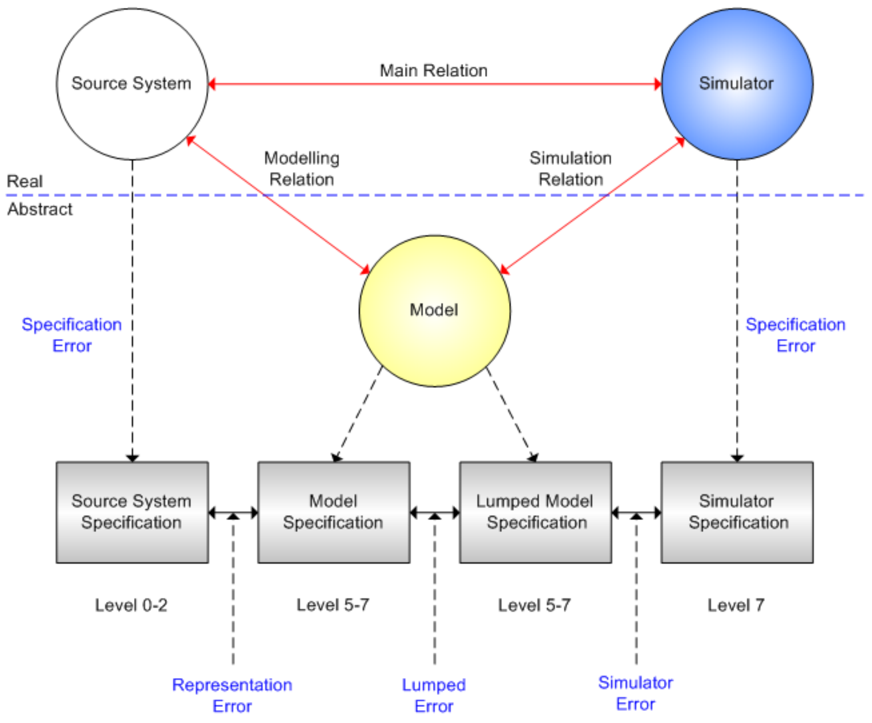


Figure 10.4: Relations and Associated Errors

that the goal is to build credibility in a step wise fashion. Further, it can be argued that by only taking small steps towards the actual simulator, it is more likely that the simulation will end in success and eventual errors are identified.

Several strategies can be envisaged in this context.

Validation:

1. Formally study the morphism between the source system and the model.
2. Assume well-verified simulator and compare the model to the source system through testing.

Verification:

1. Formally study the morphism between the model and the simulator.
2. Assume well-validated model and compare model and simulator through testing.

In addition, a final strategy involves formally studying the morphism between the source system and the simulator directly [4].

The relations that need to be studied to establish credibility can also be summed up like shown in Figure 10.5. Note that the horizontal axis is accumulative. That is, component 3 need to accommodate both component 1 and 2.

As shown here, four components, C1 to C4, and their specifications are listed. The respective morphisms and applicability relations that need to be confirmed is also shown, as well as the relations to the experimental frame.

[4]Note that source system and simulator can be compared directly through testing. This does however not necessarily involve DEVS

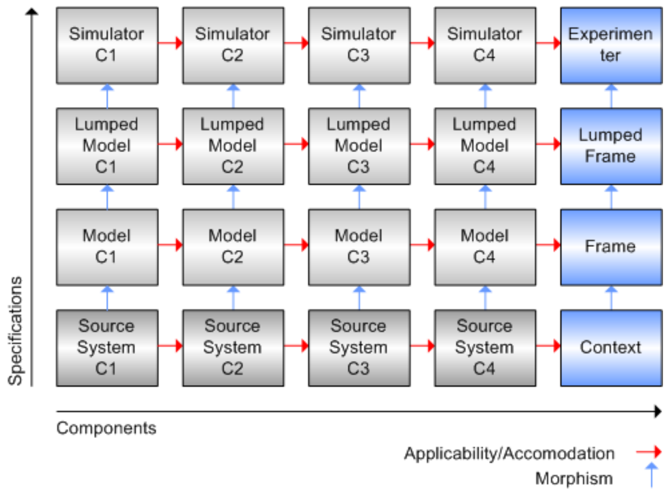


Figure 10.5: Overview of Relations

10.5.1 Well-Known Source System

The submarine combat system to be simulated at FFI, and hence also the subset described in Chapter 8, consists mainly of a well-known man-made source system. This can prove to be valuable knowledge with respect to the simulation credibility.

As the inner structure of the system can be argued to be known, the source system can be specified at a higher level than usual. Further, by exploiting this to create a similar structure in the model and later in the simulator, the structural validity will increase and further potentially the predictive validity of the simulator.

The model given in Chapter 8 is in itself an example of this. By choosing components similar to the ones found in the real source system, credibility can be argued to increase as the layout of the components is the same.

Further at a lower level, by applying real Kalman Filter algorithms in the Kalman Filter component, it can be argued that the component is more likely to produce correct output given input not earlier tested.

10.5.2 Closure Under Validation

Due to the closure under coupling property, it can be argued that coupling well validated models together will result in a well validated resultant. In the DEVS-based literature no behaviour is specified in the coupled models, see Chapter 5, implying that the resultant's behaviour shall be equal to the behaviour shown by these separate components together.

Note however that there is a limitation here with regards to the coupling. Using the example above; if both the Doppler Log and the Kalman Filter is well-validated their resultant can be argued to be well-validated, given the constraints of the applicability relation.

If this requirement is met, DEVS allows for a component-based approach towards validation, where each component can be validated separately and composed into a working system directly.

10.5.3 Legacy Components

With regards to legacy component and especially real-world components, these can be argued to be their own simulators. As mentioned in Chapter 9, this implies that validation and verification may not be needed. However as also mentioned, these components need an interface towards the simulation. This coupling should therefore be verified at the global level. Note that this can be done formally through DEVS.

Further, with respect to the applicablity relation, DEVS, can help establish that the component will work with the rest of the global simulator. If specified at level 0-1 this can be done in a normal manner. An example here is the Steering Console. If the real console is to be used within the simulator, it can be argued that only its behaviour need to be specified. How it actually works does not matter. However this behaviour needs to be mapped towards all communicating components to establish the proper operating parameters (experimental frame) for the global simulator.

10.6 Theory and Practice

In general it may seem that DEVS can be very useful in analysing systems to be simulated. As discussed above, the formal nature of the methodology provides a rigorous tool for this purpose.

A question can however be asked with regards to whether it is too formal and hence impractical. For example, manually writing models in the notation shown in Appendix A might become tedious, not to mention the task of establishing morphisms and applicability. Proper tools and notation might however ease this issue. For example, a graph based notation as well as a well designed tool for model creation and management, including establishing morphisms and applicability relations, would be useful.

Further, as shown in Figure 10.4, the DEVS analysis with regards to credibility is completely abstract. Many may state that abstract representations will always only be simplifications of the real world. For this reason one might ask: can one guarantee that all real world aspects are taken into consideration in the abstract representations?

The answer to this is not known. As can be seen in the figure, the specifications might incorporate errors from the specification task, the proofs may include errors in them selves and finally also the results of the proofs can be interpreted wrongly.

With all this in mind, the abstract analysis does however allow for a analysis not possible otherwise. For example, the analysis can be conducted independently of the simulator, given that the specifications are available, as well as absolute proofs can be completed.

10.6.1 Lessons Learned

In the following the key lessons learned will be summed up. The lessons learned here are based upon both the discussions made in this chapter as well as experiences made of the work carried out in this context.

- **Conceptual Framework**
  The basic framework of modelling and simulation provides concepts and relations which can be useful within any modelling methodology.
- **Support for Multi-Formalisms Modelling**
  Not only is DEVS universal for all discrete event systems, but continuous time systems can be specified, simulated and studied through continuous, discretized, quantized systems as well as GDEVS.
- **Separation of Concerns**
  The separation of model and simulator provides a separation of concerns which can prove valuable in both mastering complexity as well as correctness. As both the model and simulator can be created separately, these tasks can be done by their domain experts respectively.
- **Dynamics**
  It is clear from the discussions made in this thesis that control of the dynamics is an absolute necessity with regards to a recommended methodology.
- **Distributed Support**
  From a modelling perspective DEVS has no concept of distributed models or simulators. DEVS can however aid composability, due to its formal view on applicability and accommodation. Further, DEVS can be, and has been, combined with various distributed computing middleware on a simulator level in the past. This is however not studied in this thesis.
- **Real World Components**
  Real world components can be argued to be simulators within the DEVS methodology. However, many of the properties and special requirements associated with these components are still hidden to the modeller in DEVS. Note that RTDEVS has been developed, trying to bridge this gap.
- **Lumped Model**
  It is currently unknown how far one should go in the modelling domain to try and capture all aspects of the simulator. The lumped model approach, defining an entity which purpose is to represent a model at a level closer to the simulator, may prove valuable, but needs more study.
- **Simulation Credibility**
  DEVS provide concepts and relations for this purpose. Through formal analysis and proofs, credibility can be strengthened.
- **Ease of Use**
  At present time DEVS may need to be extended with notation and tools focusing at ease of use.

10.6.2 View on Scale-Up

Although the analysis performed here is based upon the study of a small subset of the submarine combat system, the universality of DEVS indicates that the findings in this chapter will also generally apply for the simulation of the complete system.

A remark in this context is however that simulations of continuous components, as well as simulation incorporating other legacy components and technologies, should be studied further.

The need for formally establishing all relations also indicates that using DEVS at a larger scale might be impractical unless proper tools are obtained.

Finally, it is here suggested that a more complete study is performed, studying DEVS through all phases of a simulation project. Not only would such a study increase the foundation to draw conclusions about the usefulness of DEVS, but one can study the specifications and relations needed throughout the project in more detail.

Part IV

Conclusion and Further Work

Chapter 11
Conclusion

Simulation of complex systems for the purpose of experimentation and exploration of new concepts is becoming more and more important. In this regard, it has been established a need for a formal and unambiguous methodology to facilitate this task.

FFI is currently developing a test facility for submarine combat systems which is to be used in concept development and later acquisition of military technology. This study has applied and explored the usefulness of the DEVS-based theory with regards to a subset of this system, with the goal of achieving experiences as well as to provide recommendations with regards to this new theory.

The findings of this study argue that the DEVS can be a valuable methodology for this purpose. The theory presents a conceptual framework, unambiguously identifying concepts within the domain. It was also found that representative specification could be made of all components, as well as it can be argued that the formal aspect in DEVS can aid in both dealing with complexity, composability and in establishing credibility.

It was however discovered that there is an issue with regards to how far DEVS should go in supporting simulator related aspects. For example, support for distributed computing middleware and real-time constraints. A solution here is suggested by adopting a lumped model, defined to be a specification taking into consideration more of the simulator properties. Research should however be made to further study this issue.

A complete overview of discussions in this regard, as well as lessons learned, is found in Chapter 10.

Review of Goals

A review of the goals of this study presented in Chapter 1 is provided in the following.

- **Identify relevant DEVS concepts**
  A literature study was performed for this purpose and relevant entities, relations and formalisms has been established. The result of this work can be found in Part II of this thesis.
- **Create an architectural description using the DEVS-based theory**
  A general model has been created identifying the aspects mentioned in the assignment text. Some value ranges and real-time constraints are however not decided at present time and is therefore not included.
- **Identify operating parameters and special requirements**
  Operating parameters and special requirements have been discussed with re-

gards to distributed computing middleware and real world components. Note that no other legacy components were identified within the subset studied.

- **Explore usefulness with regards to analysis and credibility**
  Based upon the experiences made with regards to part 1 and 2 of the assignment text, the usefulness of DEVS for this purpose has been discussed in general. The identified simulation aspects of the subset in Chapter 7 is here used as a starting point for these discussions.
- **Identify areas of further study**
  Areas of further study have been suggested throughout this thesis, and will be summed up in Chapter 12.

Recommendations

In general is seems like DEVS is a valuable tool in studying systems similar to the one studied in this thesis as well as for helping in establishing simulation credibility. Especially with regards to couplings and composability DEVS might prove helpful. The formal specifications and definitions can ensure that consistency is achieved.

No commercial tools are however known and for this reason using DEVS throughout the TEKULA project might become challenging. A study could here be performed surveying potential tools.

Based upon the previous discussions, it is however recommended that DEVS be applied as a general methodology for studying and describing the submarine combat system dynamics. Especially the separation of model and simulator can be argued to be a valuable strategy, allowing rapid model development and maintenance.

Further, the concepts of the modelling and simulation framework might also be used as a conceptual framework as it presents an unambiguous framework where discussion and understanding can be achieved.

The need for accuracy with respect to continuous systems should also be explored. DEVS provides with several strategies towards representing and controlling such system behaviour. Both GDEVS and QS present interesting strategies in this regard.

Finally, although DEVS seems reasonable for studying the system of interest, UML should still be used with regards to developing simulators or abstract simulators. For example could this be done by using the DEVS specification of a component as a starting point.

The relation to FEDEP and use of middleware should also be explored further.

Chapter 12 Further Work

This thesis tries to shed light on the usefulness of using DEVS in the context of simulating a submarine combat system. Since, this has been an exploratory study, several directions towards further study can be envisaged.

A natural direction is to extend the work done here and study the use of DEVS through the full life cycle of a project. Such a study would benefit from a more rigorous foundation on which to draw general conclusions as well as the true potential of using the DEVS specifications and relations through all phases would become clearer.

Further, it should be mapped how far DEVS should go in covering simulator related issues. As mentioned, RTDEVS tries to bridge this gap, but it can be argued that including more of the functionality of the simulator also challenge the separation of concern aspect. The lumped model approach suggested here should also be explored in this context.

Other possible further work is briefly described in the following.

Formalisms

Much research have been conducted on formalisms in later years, extending and integrating formalisms. In this context it is noticed a need for developing a parallel and real-time equivalent to PDEVS and RTDEVS with regards to GDEVS. This would allow usage of GDEVS also in distributed and real-time environments.

Further, several strategies related to representing continuous behaviour in a discrete manner has been presented. A thorough comparison of strategies, including GDEVS and QS, should here be performed, resulting in a recommendation towards when and where the different strategies might prove the most valuable.

Finally, also as PDEVS was presented as a revision of DEVS in [2], its validity within DEVDESS and MFN should be explored. The rationale for this is that although DEVS and PDEVS can represent the same set of systems, the explicit serialization presented in DEVS can be argued not to be needed. Using PDEVS directly, within a multi-formalism simulation, could ease the modelling task.

Recommended Development Process

With regards a recommended development process, a study should be conducted to establish such a process with regards to using DEVS. Alternatively, as suggested in Chapter 4 such a process should be seen in the light of the FEDEP process. Note however that the latter is meant for distributed simulations.

Tools

Although this thesis has not focused on simulators and tools, it is suggested that well-designed tools be explored and compared. Development of tools to support the usage of DEVS is a necessity and focus should be on ease of use.

Especially in this context, a tool for model creation and management, with functionality for establishing morphisms and applicability relations would be useful, if not essential.

Currently, no known commercial products are known to the author.

Other

Finally, the alternative usage of GDEVS, presented in Chapter 5 should be studied for the purpose of establishing its usefulness.

Part V

Appendix

Appendix A
Specification Examples

This appendix will provide with some examples of components specified by the formalisms found in the DEVS-based literature.

A.1 Model

The models provided in Chapter 8 and 9 present a number of components which types cover most of the types found in the DEVS-based literature. Below, full specifications of some of these components are provided as examples on final specifications.

A.1.1 System Clock

$DEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$

```
$X = \{update \mid update \in R\}$
$Y = \{clock \mid clock \in R\}$
$S = \{time \mid time \in R,$
        $rt \mid rt \geq 0\}$
$\delta_{ext}(update, e, time, rt)$
        $time = update$
        $rt = (rt - e) * \varepsilon$
$\delta_{int}(time, rt)$
        $time = time + rt$
        $rt = 0.01 * \varepsilon$
$\lambda(time, rt)$
        $clock = time$
$ta(time, rt) = rt$
```

ε is here the internal error of the System Clock component. e is the current duration of the event.

A.1.2 Pressure Sensor

$DTSS = (X, Y, Q, \delta, \lambda, c)$

$X = \{pressure \mid pressure \in R\}$

```
$Y = \{depth \mid depth \in R\}$
$Q = \{d \mid d \in R\}$
$\delta(pressure, d)$
        $d = f(pressure)$
$\lambda(d)$
        $depth = d$
$c = 0.1$
```

The function f is here the pressure-to-depth function.

A.1.3 Echo Sounder

$$DEVDESS = (X^{discr}, X^{cont}, Y^{discr}, Y^{cont}, S, \delta_{ext}, C_{int}, \delta_{int}, \lambda^{discr}, f, \lambda^{cont})$$

```
$X^{discr} = \{mode \mid mode \in [OFF, SINGLEPING, ON]\}$
$X^{cont} = \{distance \mid distance \in R\}$
$Y^{discr} = \{m\ distance \mid 0 \leq m\ distance \leq 1000\ and\ \infty\}$
$Y^{cont} = \{\}$
$S = \{md \mid 0 \leq md \leq 1000\ and\ \infty,$
        $mm \mid mm \in [OFF, SINGLEPING, ON],$
        $rt \mid rt \geq 0\ \}$
$\delta_{ext}(mode, distance, e, md, mm)$
        $mm = mode$
$\delta_{int}(mode, distance, e, md, mm)$
        $rt = 0$
        $if\ (distance > 1000)$
          $md = \infty$
        $else$
          $md = distance$
          $if\ (mm == SINGLEPING)$
          $mm = OFF$
$\lambda^{discr}(e, md, mm)$
        $m\ distance = md$
$\lambda^{cont}(e, md, mm)$
$f(e, md, mm)$
        $if\ (mm == ON)$
          $d\ rt\ /\ dt = 1$
        $else$
          $d\ rt\ /\ dt = 0$
$C_{int}(md, mm)$
        $mm == SINGLEPING\ ||\ rt == timeInterval$
```

The timeInterval constant here refers to the frequency of normal execution of the echo sounder.

A.1.4 Inertial Navigation

For this example only one accelerator and one gyroscope is used. Note that in the Inertial Navigation model given in Chapter 8 there shall be three of each.

Accelerator

$DTSS = (X, Y, Q, \delta, \lambda, c)$

X = {*gravity* | *-1* ≤ *gravity* ≤ *1*,
kin. acceleration | *-0.5* ≤ *kin. acceleration* ≤ *0.5* }
Y = {*m acceleration* | *-1.5* ≤ *m acceleration* ≤ *1.5* }
Q = {*m* | *m* ∈ *R*}
δ(gravity, kin. acceleration, m)
m = $f_{acc}(gravity, kin.acceleration)$
λ(m)
m acceleration = *m*
c = *0.1*

f_{acc} is the function for calculating measured acceleration.

Gyroscope

$DTSS = (X, Y, Q, \delta, \lambda, c)$

X = {*earth rot rate* | *-15.04* ≤ *earth rot rate* ≤ *15.04*,
rot rate | *-50* ≤ *rot rate* ≤ *50* }
Y = {*m rot rate* | *-50* ≤ *m rot rate* ≤ *50* }
Q = {*m* | *m* ∈ *R*}
δ(earth rot rate, rot rate, m)
m = $f_{rot}(earthrotrate, rotrate)$
λ(m)
m rot rate = *m*
c = *0.1*

f_{rot} is the function for calculating measured rotation rate.

Inertial Navigation

$DTSN = (X, Y, D, M_d, EIC, EOC, IC, c)$

X = {*gravity* | *-1* ≤ *gravity* ≤ *1*,
kin. acceleration | *-0.5* ≤ *kin. acceleration* ≤ *0.5*,
earth rot rate | *-15.04* ≤ *earth rot rate* ≤ *15.04*,
rot rate | *-50* ≤ *rot rate* ≤ *50* }
Y = {*m acceleration* | *-1.5* ≤ *m acceleration* ≤ *1.5*,
m rot rate | *-50* ≤ *m rot rate* ≤ *50* }
D = {*gyro, acc*}
M_d: *gyro* = *Gyroscope*,
acc = *Accelerator*
EIC = {*((N,kin. acceleration), (gyro,kin. acceleration)),*
((N,gravity), (gyro,gravity)),
((N,earth rot rate), (acc,earth rot rate)),
((N,rot rate), (acc,rot rate)) }
EOC = {*((gyro,m rot rate), (N,m rot rate)),*
((acc, macceleration), (N,m acceleration)) }

```
$IC = \{\}$
$c = 0.1$
```

N here refers to the coupled model itself.

A.1.5 GPS

```
$DTSS = (X, Y, Q, \delta, \lambda, c)$

$X = \{$GPS Clock | GPS Clock > 0,
        GPS Position = [x,y,z] | where x,y and z $\in$ R,
        Clock | Clock > 0,
        Best Position = [x,y,z] | where x,y and z $\in$ R$\}$
$Y = \{$GPS Clock' | GPS Clock' > 0,
        GPS Position' = [x,y,z] | where x,y and z $\in$ R$\}$
$Q = \{$pos = [x,y,z] | where x,y and $\in$ R,
        cl | cl > 0$\}$
        oldGPS | oldGPS > 0$\}$
$\delta$(GPS Clock, GPS Position, Clock, Best Position, cl, pos, oldGPS)
        if (GPS Clock != oldGPS)
            oldGPS = GPS Clock
            cl = oldGPS
            pos = GPS Position
        else
            cl = Clock
            pos = Best Position
$\lambda$(cl, pos, oldGPS)
        GPS Clock' = cl
        GPS position' = pos
$c = 0.1$
```

A.2 Lumped Model

In the lumped models presented in Chapter 9 the base model has been transformed into GDEVS. For this reason it will be shown some examples of fully specified GDEVS specifications below.

Note that GDEVS of order 0 is the same as DEVS. No examples of purely discrete systems will therefore be given.

A.2.1 Pressure Sensor

```
$GDEVS = (XM, YM, S, \delta_{ext}, \delta_{int}, \lambda, ta)$

$XM = \{$pressure = (a, b) | a, b $\in$ R $\}$
$YM = \{$depth | depth >0 $\}$
```

```
$S = \{a' \mid a' \in R,$
        $b' \mid b' \in R,$
        $rt \mid rt \geq R,$
        $duration \mid duration \geq R\ \}$
$\delta_{ext}((a, b), e, a', b', duration, rt)$
        $a' = a$
        $b' = b$
        $rt = 0$
        $duration = 0$
$\delta_{int}(a', b', duration, rt)$
        $duration = duration + rt$
        $rt = 0.1$
$\lambda(a', b', duration, rt)$
        $depth = f((a' * duration) + b')$
$ta(a', b', duration, rt) = rt$
```

Note that pressure consists of the tuple (a, b) which represents pressure(t) = (a * t) + b. The Coeff function is not included here. This function transforms between coefficients and a continuous signal.

A.2.2 Echo Sounder

$$GDEVS = (XM, YM, S, \delta_{ext}, \delta_{int}, \lambda, ta)$$

```
$XM = \{distance = (a, b) \mid a, b \in R,$
        $mode \mid mode \in [OFF, SINGLEPING, ON]\ \}$
$YM = \{m\ distance \mid 0 \leq m\ distance \leq 1000\ and\ \infty\ \}$
$S = \{a' \mid a' \in R,$
        $b' \mid b' \in R,$
        $duration \mid duration \in R\ \}$
        $mm \mid mm \in [OFF, SINGLEPING, ON],$
        $rt \mid rt \in R\ \}$
$\delta_{ext}(mode, (a, b), e, a', b', mm, duration, rt)$
    if $(a' \mathrel{!}= a\ ||\ b' \mathrel{!}= b)$
        $a' = a$
        $b' = b$
        $duration = 0$
    if $(mode \mathrel{!}= mm)$
        $mm = mode$
        if $(mm \mathrel{!}= OFF)$
            $rt = 0$
            $duration = duration + e$
        else
            $rt = \infty$
$\delta_{int}(a', b', mm, duration, rt)$
    if $(mm == SINGLEPING)$
        $rt = \infty$
        $mm = OFF$
    else
        $duration = duration + rt$
        $rt = timeInterval$
```

```
λ(a', b', mm, duration, rt)
        m distance = (a' * duration) + b'
ta(a', b', mm, duration, rt) = rt
```

Bibliography

[1] Young Kwan Cho, Xiaolin Hu, and Bernard P. Zeigler. The rtdevs/corba environment for simulation-based design of distributed real-time systems. *SIMULATION*, 79, 2003.

[2] Alex Chung Hen Chow and Bernard P. Zeigler. Parallel devs: a parallel, hierarchical, modular, modeling formalism. In *Winter Simulation Conference*, pages 716–722, 1994.

[3] Leonardo Chwif, Marcos Ribeiro Pereira Barretto, and Ray J. Paul. On simulation model complexity. In *WSC '00: Proceedings of the 32nd conference on Winter simulation*, pages 449–455, San Diego, CA, USA, 2000. Society for Computer Simulation International.

[4] Mariana C. D'Abreu and Gabriel A. Wainer. Hybrid dynamic systems: models for continous and hybrid system simulation. In *Winter Simulation Conference*, pages 641–649, 2003.

[5] Paul K Davis and Robert H. Anderson. Improving the composability of dod models and simulations. *The Journal of Defense Modeling and Simulation*, 1, 2004.

[6] N. Giambiasi and J.C. Carmona. Generalized discrete event abstraction of continuous systems: Gdevs formalism. *Simulation Modelling Practice and Theory*, 14, 2006.

[7] Norbert Giambiasi, Bruno Escude, and Sumit Ghosh. Gdevs: A generalized discrete event specification for accurate modeling of dynamic systems. *Trans. Soc. Comput. Simul. Int.*, 17(3):120–134, 2000.

[8] Ezequiel Glinsky and Gabriel A. Wainer. Extensions: performance analysis of real-time devs models. In *Winter Simulation Conference*, pages 588–594, 2002.

[9] McKenna Iain H. and Stephen Little. Developing tactics using low cost, accessible simulations. In *Winter Simulation Conference*, pages 991–1000, 2000.

[10] Joon Sung Hong, Hae-Sang Song, Tag Gon Kim, and Kyu Ho Park. A real-time discrete event system specification formalismfor seamless real-time software development. *Discrete Event Dynamic Systems*, 7(4):355–375, 1997.

[11] Xiaolin Hu. A simulation-based software development methodology for distributed real-time systems, 2004.

[12] Simulation Interoperability Standards Organization Inc. Rpr-fom version 1.0 siso-std-001.1-1999, 1999.

[13] Yong Jae Kim and Tag Gon Kim. A heterogeneous simulation framework based on the devs bus and the high level architecture. In *Winter Simulation Conference*, pages 421–428, 1998.

[14] Ernesto Kofman, J.S. Lee, and B.P. Zeigler. Devs representation of differential eqation systems: Review of recent advances. In *European Simulation Symposium*, 2001.

[15] Yvan Labiche and Gabriel Wainer. Towards the verfication and validation of devs models. In *Open International Conference on Modeling and Simulation (ISIMA)*, 2005.

[16] Thom McLean, Richard Fujimoto, and Brad Fitzgibbons. Middleware for real-time distributed simulations: Research articles. *Concurr. Comput. : Pract. Exper.*, 16(15):1483–1501, 2004.

[17] Dale K. Pace. Modeling and simulation verification and validation challenges. `https://www.dmso.mil/public/library/projects/vva/found_04/pace_ms_challenges_jhuapl_tech_digest_preprint.pdf`.

[18] Trevor Pearce, Amir Saghir, and Gabriel Wainer. Hla to simulate computer systems at the hardware platform level. In *Simulation Interoperability Workshop*, 2004.

[19] Trevor W. Pearce. Simulation-driven architecture in the engineering of real-time embedded systems. In *Simulation Interoperability Workshop*, 2004.

[20] Michael Pidd. Five simple principle of modelling. In *Winter Simulation Conference*, pages 721–728, 1996.

[21] Michael Pidd and R. Bayer Castro. Hierarchical modular modelling in discrete simulation. In *Winter Simulation Conference*, pages 383–390, 1998.

[22] Robert G. Sargent. Verification, validation, and accreditation: verification, validation, and accreditation of simulation models. In *Winter Simulation Conference*, pages 50–59, 2000.

[23] Hessam S. Sarjoughian and Bernard P. Zeigler. The role of collaborative devs modeler in federation development. In *International Conference on Web-Based Modeling and Simulation*, 1999.

[24] Simon J. E. Taylor, Peter Lendermann, Ray J. Paul, Steven W. Reichenthal, Steffen Straßburger, and Stephen John Turner. Panel on future challenges in modeling methodology. In *Winter Simulation Conference*, pages 327–, 2004.

[25] MÄK Technologies. *MÄK Technologies.* `http://www.mak.com`.

[26] Andreas Tolk and James A. Muguira. M and s within the model driven architecture, 2004.

[27] M.K. Traore and A. Muzy. Capturing the dual relationship between simulation models and their context. *Simulation Modelling Practice and Theory*, 14, 2006.

[28] Gabriel A. Wainer. Devs standardization study group, interim final report, 2005.

[29] William F. Waite and David H. Smith. Sba/seba: Implementing the inevitable. In *Huntsville Simulation Conference*, 2001.

[30] Gwendolyn H. Walton, Robert M. Patton, and Douglas J. Parsons. Usage testing of military simulation systems. In *WSC '01: Proceedings of the 33nd*

conference on Winter simulation, pages 771–779, Washington, DC, USA, 2001. IEEE Computer Society.

[31] Eric Weisstein. *Wolfram MathWorld*. `http://mathworld.wolfram.com/`.

[32] G. Zacharewicz, N. Giambiasi, and C. Frydman. Gdevs/hla environment: A time management improvement. In *IMACS'2005 - the 17th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation*, Paris, France, 11–15 juillet 2005.

[33] Zeigler. Extending the devs-scheme knowledge-based simulation environment for real-time event-based control. *IEEE Transactions on Robotics and Automation*, 9, 1993.

[34] Bernard P. Zeigler. Verification and validation of devs models: Applying the theory of modeling and simulation to the needs of simulation based acquisition. In *Summer Computer Simulation Conference*, 2000.

[35] Bernard P. Zeigler. Devs today: Recent advances in discrete event-based information technology. In *MASCOTS*, pages 148–, 2003.

[36] Bernard P. Zeigler. Embedding devdess in devs, 2006.

[37] Bernard P. Zeigler, Doohwan Kim, and Stephen J. Buckley. Distributed supply chain simulation in a devs/corba execution environment. In *Winter Simulation Conference*, pages 1333–1340, 1999.

[38] Bernard P. Zeigler, Tag Gon Kim, and Herbert Praehofer. *Theory of Modeling and Simulation*. Academic Press, January 2000.

[39] Bernard P. Zeigler and Sarjoughian Hessam S. Implications of m and s foundations for the v and v of large scale complex simulation models, 2002.

[40] Bernard P. Zeigler and Hessam S. Sarjoughian. Distributed simulation: creating distributed simulation using devs m and s environments. In *WSC '00: Proceedings of the 32nd conference on Winter simulation*, pages 158–160, San Diego, CA, USA, 2000. Society for Computer Simulation International.

[41] Bernard P. Zeigler and Sankait Vahie. Devs formalism and methodology: unity of conception/diversity of application. In *WSC '93: Proceedings of the 25th conference on Winter simulation*, pages 573–579, New York, NY, USA, 1993. ACM Press.