VARIABLE STRUCTURE AND DYNAMISM EXTENSIONS TO A DEVS BASED MODELING AND SIMULATION FRAMEWORK

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

BY

FATİH DENİZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

FEBRUARY 2010

Approval of the thesis:

VARIABLE STRUCTURE AND DYNAMISM EXTENSIONS TO A DEVS BASED MODELING AND SIMULATION FRAMEWORK

submitted by **FATİH DENİZ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

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

Prof. Dr. Müslim Bozyiğit
Head of Department, **Computer Engineering**

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

Dr. Mahmut Nedim Alpdemir
Co-supervisor, **TUBITAK UEKAE ILTAREN**

Examining Committee Members:

Assoc. Prof. Dr. Ali Doğru
Computer Engineering Dept., METU

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering Dept., METU

Asst. Prof. Dr. Erol Şahin
Computer Engineering Dept., METU

Dr. Cevat Şener
Computer Engineering Dept., METU

Dr. Mahmut Nedim Alpdemir
TUBITAK UEKAE ILTAREN

Date: 03.02.2010

ABSTRACT

VARIABLE STRUCTURE AND DYNAMISM EXTENSIONS TO A DEVS BASED MODELING AND SIMULATION FRAMEWORK

Deniz, Fatih
M.Sc., Department of Computer Engineering
Supervisor : Assoc. Prof. Dr. Halit Oğuztüzün
Co-Supervisor : Dr. Mahmut Nedim Alpdemir

February 2010, 57 pages

In this thesis, we present our approach to add dynamism support to simulation environments, which adopts DEVS-based modeling and simulation approach and builds upon previous work on SiMA, a DEVS-based simulation framework developed at TUBITAK UEKAE. Defining and executing simulation models of complex and adaptive systems is often a non-trivial task. One of the requirements of simulation software frameworks for such complex and adaptive systems is that supporting variable structure models, which can change their behavior and structure according to the changing conditions. In the relevant literature there are already proposed solutions to the dynamism support problem. One particular contribution offered in this study over previous approaches is the systematic and automatic framework support for post-structural-change state synchronization among models with related couplings, in a way that benefits from the strongly-typed execution environment SiMA provides. In this study, in addition to introducing theoretical extensions to classic SiMA, performance comparisons of dynamic version with classic version over a sample Wireless Sensor Network simulation is provided and possible effects of dynamism extensions to the performance are discussed.

Keywords: Modeling and Simulation, DEVS, SiMA, Variable Structure Models, Dynamic Extensions

ÖZ

DEVS TABANLI BİR MODELLEME VE BENZETİM ÇERÇEVESİNDE DEĞİŞKEN YAPI VE DİNAMİZM DESTEĞİ

Deniz, Fatih
Yüksek Lisans, Bilgisayar Mühendisliği
Tez Yöneticisi : Doç. Dr. Halit Oğuztüzün
Ortak Tez Yöneticisi : Dr. Mahmut Nedim Alpdemir

Şubat 2010, 57 sayfa

Bu tezde DEVS tabanlı modelleme ve benzetim altyapılarında dinamizm desteği ile ilgili formal bir yaklaşım sunulmakta olup, TUBITAK UEKAE tarafından geliştirilmiş olan ve DEVS yaklaşımını esas alan Simülasyon Modelleme Altyapısı (SiMA) üzerinde bahsedilen formal eklentiler gerçekleştirilmiştir. Karmaşık ve değişken sistemlerin benzetim modellerinin tanımlanması ve koşturulması çoğu zaman zorlu bir süreci içermektedir. Bu tür sistemlerin oluşturulmasında kullanılması planlanan benzetim yazılım altyapılarında aranan özelliklerden bir tanesi de değişken yapılı model tanımlanmasına destek sağlanmasıdır. DEVS ortamında dinamizm desteği ile ilgili daha önceden yapılmış çalışmalar bulunmaktadır. Bu çalışmanın önceki çalışmalara önemli bir katkısı da yapısal bir değişiklik sonrasında sistematik ve otomatik bir şekilde yeni eklenen bağlantılara göre modellerin durum parametrelerinin güncellenmesidir. Bu çalışmada, değişken yapılı model kullanımı için gerekli olan formal dinamizm eklentilerinin ve gerçekleştirilme detaylarının açıklanmasının yanı sıra, örnek bir kablosuz algılayıcı ağları benzetimi oluşturulmuş ve bu benzetim üzerinde SiMA'nın dinamik versiyonu ile klasik versiyonunun performansları karşılaştırılmış ve dinamik model kavramının performansta yapabileceği etkiler tartışılmıştır.

Anahtar Kelimeler: Modelleme ve Benzetim, DEVS, SiMA, Değişken Yapılı Modeller, Dinamik Eklentiler

To my family...

ACKNOWLEDGMENTS

I am heartily thankful to my supervisor, Assoc. Prof. Dr. Halit Oğuztüzün, for his professional guidance and suggestions throughout my thesis studies.

I would like to show my gratitude to jury members Assoc. Prof. Dr. Ali Doğru, Asst. Prof. Dr. Erol Şahin and Dr. Cevat Şener for reviewing and evaluating my thesis.

Special thanks to Dr. Mahmut Nedim Alpdemir. His invaluable ideas, suggestions and help have been essential to my research. I appreciate all the time he has spent for the development of my thesis.

I would like to thank TUBITAK UEKAE ILTAREN for deeply supporting my graduate studies. I also want to thank to my colleages at work for their understanding and support during my academic studies.

I am also grateful to Ahmet Kara for his helpful discussions, suggestions and help throughout my thesis studies.

I would also like to express my thanks to my friends, especially Erkay Uzun, Fatih Gökbayrak, Osman Nuri Osmanlı, Enver Kayaaslan and Uzeyir Başer for their invaluable friendship, support and joy they bring to my life.

Finally, I would like to express my thanks to my family, making me who I am now with their love, trust, understanding, encouragement and support throughout my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVATIONS

DEVS	Discrete Events System Specification
SiMA	Simulation Modeling Architecture, Simülasyon Modelleme Altyapısı
DSDEVS	Dynamic Structure DEVS
DynDEVS	Dynamic DEVS
WSN	Wireless Sensor Network
GUI	Graphical User Interface

CHAPTER 1

INTRODUCTION

Analyzing the behavior of complex and adaptive systems through simulation often requires the underlying modeling and simulation approach to support structural and behavioral changes. This requirement may stem from the inherent nature of the real world system under study such as ecological or social systems (as indicated in [22]), it may stem from the modeling and simulation methodology of the analyst or it may be due to the way system modelers approach to the modeling of inherent behavioral complexity of their models. A good example to a combination of the latter two is the case where the simulation study involves a large number of highly complex systems, the analyst wants to observe the behavior of these systems at varying levels of fidelity, and the modeler constructs the models in a way to allow the models to exhibit different observable behaviors during the course of simulation. This particular case implies that models may switch between different behavioral specifications (e.g. fidelity levels) dynamically at run time depending on various triggering events. Allowing modifications to model structures and to internal functional specifications while the simulation is running is a challenging task due to instabilities and inconsistencies this may introduce, especially if the underlying modeling approach does not provide a sound formal basis upon which the run-time infrastructures can be established.

In this study, we take a particular stand to the problem of dynamism support in simulation environments by adopting DEVS based modeling and simulation approach and by building upon previous work on SiMA [13, 14], a DEVS-based modelling and simulation framework developed at TUBITAK UEKAE. Some of the reasons why dynamism support is required in a modeling and simulation framework can be listed as follows:

1. Some simulation scenarios can efficiently be executed only through dynamic structure support. For example, consider a simulation scenario in which two planes follow the terrain at a specific altitude. In this scenario, the provision of the terrain information to plane models could be implemented using an environment model. When this scenario is executed in a high resolution setting, it may be impossible to load the terrain that represents the whole world. So, if the simulation requires the whole world, terrain has to be decomposed into regions, such that, the environment model representing each of these regions interacts with the models representing the planes flying over it. When a plane crosses over a region, the couplings with the region left behind are removed and new couplings with the region plane enters are added.

2. Dynamism support can be used to yield an optimal model execution structure. Adding dynamism support is most likely to increase performance. A model can be loaded into the memory whenever needed and can be removed from the memory after completing its job. Continuing from the previous plane example, in order to use the resources effectively, unnecessary sections of the terrain can be removed from the memory. In addition, through dynamic management of couplings and ports, unnecessary message transfers can also be eliminated, thereby increasing the performance by doing less work in each step.

3. Dynamism support may become a necessity for creating more realistic simulations. Model structure may have to be changed whenever necessary while the simulation is running. For example, if a missile explodes, the model representing it should also be removed from the model structure. Dynamism support is the natural way of handling this kind of situations.

4. Complex systems that require behavioral or structural changes to adapt to changing situations can be modeled more efficiently with variable structure models. Examples of these systems include wireless sensor networks, distributed computing systems and ecological systems.

5. Also, there may be unpredictable changes that need to be modeled at run-time. Consider a human population simulation, in which civilians and combatants

are represented by different models. By human nature, a civilian can change into a combanant according to the occuring events and since human emotions cannot be easily predicted, such cases cannot be easily modeled in the simulation construction phase. In order to allow simulations to adapt these types of unpredictable changes, dynamism support can be used.

We observe that several approaches to dynamism are already proposed in the relevant literature [2, 3, 16, 18, 22]. We note that three distinct categories of change are discussed in those existing approaches: 1 - A change in the overall compositional state of models, 2 - A change in the connectivity relationships (coupling) among the models, 3 - A change in internal functional behavior of the model. We find two of the formal approaches to the variable structure models in DEVS environment particularly relevant to our work. The first one is DSDEVS (Dynamic Structure DEVS), introduced by F.J. Barros [3]. The second one is DynDEVS, introduced by Uhrmacher [22]. A brief introduction to both of these approaches is given in Chapter 3. In addition to these formal extensions, there are approaches which adopt existing formal specifications but contribute through different routes. For instance [20] extend their existing simulation engine by adopting a combination of DSDEVS and DynDEVS. Similarly [12] take a software engineering oriented stand and propose a component-based simulation environment. Our contribution to the work in this field is extending SiMA with dynamism support building upon our specialized basic DEVS formalism with dynamism extensions.

In addition to introducing our approach in a formal way, we also discuss how to implement these formal definitions and what type of functionalities these definitions add over classic SiMA. Introducing a systematic state synchronization mechanism between networks of connected models that works in the opposite direction of the normal message flow, defining a change request message structure that is compatible to a XML Schema, allowing also root coordinator to initiate a structural change step are some of the features added over classic SiMA with the dynamism extensions. In this thesis, it also discussed that, adding dynamism to SiMA does not just gives flexibility to the SiMA users, but allows them to increase performance by using the resources more effectively. For this reason, a sample WSN Simulator is implemented and using different sample scenarios, performance improvement of dynamic version over classic version is shown.

The rest of this thesis is organized as follows: Chapter 2 gives a summary of the relevant background work, Chapter 3 summarizes previous efforts about dynamism support in modeling and simulation environments, Chapter 4 provides a detailed discussion of our approach, Chapter 5 provides analysis and performance measurements of our approach with a case study, Chapter 6 provides conclusions and future work.

CHAPTER 2

BACKGROUND

2.1 DEVS Formalism

The Discrete Event System Specification (DEVS) is a formalism introduced by Bernard Zeigler in 1976 to describe discrete event systems. In this formalism, there are two types of models: Atomic models and coupled models. Atomic models have behavioral logic. On the other hand, coupled models consists of other models and connections between those models, but no behavioral logic. An atomic model in classical DEVS formalism consists of a set of input events, a state set, a set of output events, an internal and external transition function, and an output and time advance function. Formal definition of an atomic model in classical DEVS formalism is described as follows:

M $= \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$
where,
$X = \{(p, v)|p \in InPorts, v \in X_p\}$ is the set of input ports and values,
$Y = \{(p, v)|p \in OutPorts, v \in Y_p\}$ is the set of output ports and values,
S is the set of states,
$\delta_{int} : S \rightarrow S$ is the internal state transition function,
$\delta_{ext} : Q \times X \rightarrow S$ is the external state transition function such that:

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

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

In DEVS formalism, models communicate with each other using their ports, which are interfaces of models. Input ports receive external input events X and output ports send output events Y. Every state $s \in S$ is associated with a time, calculated by time advance(ta) function, which determines the duration of the state. If no external event is fired during this time, first output function (λ) and then the internal transition function(δ_{int}) is executed. If model receives an external event during this time, then the external transition function(δ_{ext}) is executed. Current state of the model is updated in internal and external transition functions.

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

CM = $\langle X, Y, D, \{M_i\}, EIC, EOC, IC, Select\rangle$
where,
$X = \{(p,v) | p \in InPorts, v \in X_p\}$ is the set of input ports and values,
$Y = \{(p,v) | p \in OutPorts, v \in Y_p\}$ is the set of output ports and values,
D is the set of component names,
M_i is the model of component i, for $i \in D$,
EIC, EOC and IC define the coupling structure,

EIC is the set of couplings between input ports of the coupled model itself and input ports of its components.
EOC is the set of external output couplings, which connect its components output ports to models own output ports.
IC is the internal couplings, which connect a components output port to another components input within the coupled model.

Select: $2^D - \{\} \rightarrow D$ is the tie-breaker function and used for ordering simultaneous events. This function is used in the classic DEVS and eliminated in the parallel DEVS. Since SiMA-DEVS discussed in Section 2.2 is an extension of parallel DEVS, this function is not used.

As it can be observed from the coupled model definition, in DEVS formalism, no direct feedback loops are allowed. In other words, no input port of a coupled model can be connected to an output port of the same model.

Complete description of DEVS semantics can be found in [7,24].

2.2 SiMA

SiMA (Simulation Modeling Architecture) [13,14] is a modeling and simulation framework that is based on the DEVS approach as a solid formal basis for complex model construction. SiMA Simulation Execution Engine implements the parallel DEVS protocol which provides a well-defined and robust mechanism for model execution. SiMA builds upon a specialized and extended form of DEVS formalism, namely SiMA-DEVS, which:

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

2- Introduces a new transition function to account for model interactions involving state inquiries with possible algebraic transformations (but no state change), without simulation time advance. In this respect, extends the basic DEVS formalism. This is similar to the notion of zero-lookahead in HLA [10] from a time-management point of view.

An atomic model in SiMA-DEVS formalism is defined formally as follows:

$\langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta, \delta_{df} \rangle$
where,
X: Set of input values arriving from set of input ports, P_{in},
Y: Set of output values sent from set of output ports, P_{out},
P_{in}, P_{out}: Set of input and output ports such that:

$P_{in} : \{(\tau, I_x)\,|\Gamma \mapsto \tau \wedge I_x \subseteq X \wedge \forall x \in I_x, \tau \mapsto x\}$,
$P_{out} : \{(\rho, O_y)\,|\Gamma \mapsto \rho \wedge O_y \subseteq Y \wedge \forall y \in O_y, \rho \mapsto y\}$,

Γ: XMLSchema type system,

τ, ρ: data types valid wrt XMLSchema type system,

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

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

1. Introduces a type discipline to the definition of the externally visible model interfaces (i.e. ports) leading to an information model for the overall system being modeled (coherency in modeling level information space), as well as for the simulation environment (consistency and robustness in run-time-level data space).

2. Facilitates Model-Driven Engineering through well-typed and type system dependent external plugs to enable automated port matching and model composition. In fact, Model-Driven simulation construction pipeline for SiMA is successfully implemented, via a number of tools such as a code generator, a model builder and a model linker. This simulation construction pipeline is discussed in detail in Section 2.2.1.1.

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

Note also that, in addition, SiMA introduces a new transition function, δ_{df}, that enables models to access the state of other models through a specific *type* of port, without advancing the simulation time. As such, it is possible to establish a path of connected models along which models can share parts of their state, use state variables to compute derived values instantly within the same simulation time step. As stated earlier, this is similar to the notion of zero-lookahead found in HLA [10]. One may argue that the zero-lookahead behavior could be modeled by adjusting the time advance function of an atomic model such that the model causes the simulation to stop for a while, do any state inquiry via existing couplings, then re-adjusting the time advance to go back to normal simulation cycle. Although this is possible, it is argued that by introducing a transition function and a specific port type which is tied (through run time constraints imposed by the framework) to that particular transition function several advantages are gained:

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

2. Allowing such communications only to occur through a specific port type (compile time and run-time checks are carried out) the framework is able to apply application independent loop-breaking logic at the ports to prevent algebraic loops, thereby ensuring model legitimacy.

2.2.1 SiMA SOFTWARE ARCHITECTURE

SiMA is a software framework for developing simulation models and executing simulations. There are also other frameworks, such as parallel simulation execution framework for grid environments [8], that SiMA can work in accordance with. As it can be observed in Figure 2.1, SiMA consists of two main layers: *SiMA Core* and *C++ Interface.*

SiMA Core consists of five sub-components that are used for modeling and simulation. *Modeling Framework* is a set of classes and data types to be used in model development. Atomic models and all their subclasses are defined in this framework. *Connection Ports* is the transportation component that contains classes for defining ports and their connection limits. *Simulation Engine Core* uses these two components, *Modeling Framework* and *Connection Ports*, to execute a simulation. In other words, it implements the DEVS simulation protocol and in addition exposes administrative interfaces to manage and track the simulations at run time. The *Distributed Simulation Adapter* is the interface for connecting SiMA simulations with external distributed simulation infrastructures. *Messaging Constructs* is the component that presents all base data type classes and rules for inter-communication of atomic models and *SiMA Core* components.

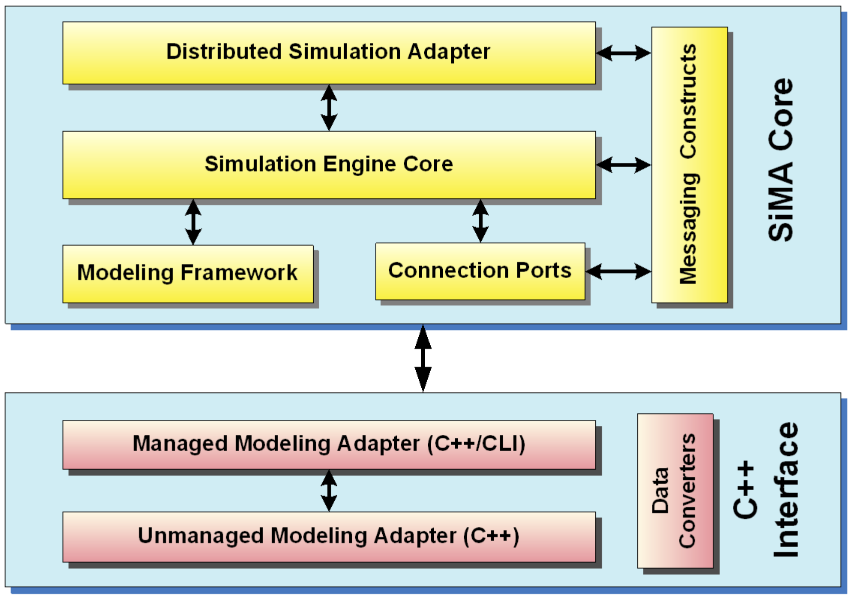


Figure 2.1: SiMA Software Architecture

The *C++ Interface* layer is implemented to allow C++ to be used as a model implementation language for SiMA atomic models. All core SiMA components are developed in .NET but SiMA supports models implemented in both .NET and C++ to co-exist in the same run-time environment during a simulation. However, since there is a strict boundary between their coding environments, various adapters and components that manage the interoperability between .NET and C++ atomic models and SiMA components are implemented in the *C++ Interface* layer.

The *C++ Interface* layer consists of three sub-components. *Unmanaged Modeling Adapter* has the same interface and class hierarchy as the .NET Modeling Framework except it is developed in pure C++ language. *Managed Modeling Adapter* is developed in C++/CLI, which is a special edition of C++ language in .NET, that allows access to both C++ and .NET methods and data types. *Managed Modeling Adapter* handles the interoperability management and delegates all simulation commands to C++ models, and provides all information required by them from the simulation environment. *Data Converters* are special adapters that perform marshalling all values between .NET and C++ data types in both ways. A model developer can use the KODO tool (described in Section 2.2.1.1) to auto-generate these data converters for his/her data types.

2.2.1.1 Model-Driven simulation construction pipeline for SiMA

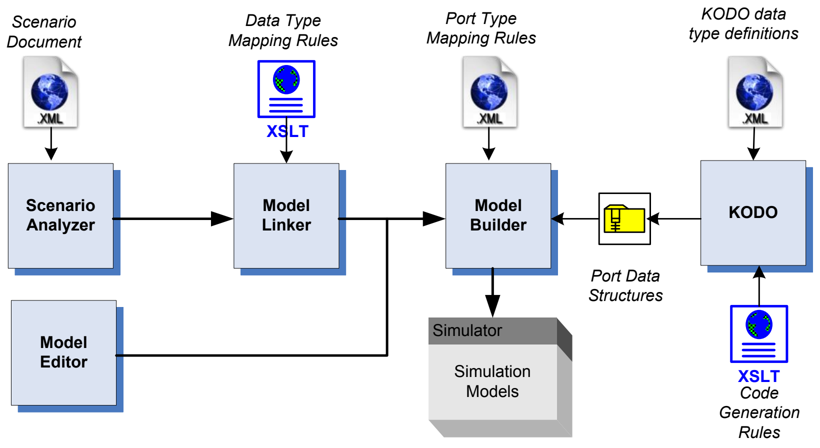


Figure 2.2: Simulation Construction Pipeline

SiMA is supported with a number of tools that collectively allows automatic simulation model construction from previously defined SiMA models. The simulation construction pipeline consists of four steps with the support of five tools (Figure 2.2):

- **KODO:** A model developer using SiMA needs to define data types to be used for model initialization and inter-port communication. Due to .NET/C++ interoperability requirements SiMA needs to apply special handling on port data types for serialization/deserialization of data values flowing between atomic models. Similarly, model initialization phase requires access to input configuration data types. KODO plays its relieving role for the developer by auto generating not only data beans but also additional processing logic for marshaling and type conversion, for access to model input configuration files for state initialization, and for trace generation. KODO allows definition of data types with simple

XML definitions and generates all classes and methods required for the model developer.

- **Scenario Analyzer:** There are two starting points in the simulation construction pipeline. One of those has a more constrained initial specification where a simulation study is defined by a *scenario* that describes all required models, model coupling information and model initialization data. In this case, Scenario Analyzer interprets a scenario definition and identifies all atomic models (from the component library) and defines their composition hierarchy in an intermediate form to be used by the model linker. It is generally an application dependent analyzer that converts the application dependent scenario file to be used by application independent model linker.

- **Model Editor:** The second route to the pipeline is the SiMA Model Editor. This editor provides a context independent model composition environment where both atomic and coupled models can be organized as libraries, thereby facilitating reusability of simulation models. SiMA allows defining atomic models either in C++ or in C# programming languages. Once these models are compiled into library files, they can then be graphically coupled using SiMA Model Editor in hierarchical block diagrams to construct more complex systems. In Figure 2.3, the model structure and couplings for a sample simulation can be seen. Dark rectangular shapes with an 'A' sign on the top-left corners indicate atomic models and lighter rectangular shapes with no signs on the top-left corner indicate coupled models. Couplings are shown with directed arrows between model ports. The direction of the arrow connotes the direction of the data flow. Port names are denoted as labels beneath the ports. The Model Editor produces an intermediate form to be used by the Model Builder.

- **Model Linker:** Model linker reads scenario document and loads the data type mapping rules for the current configuration and then produces two output files. The first file, mapping definition, includes all port connections, names and locations of atomic models and their hierarchy mapping to produce a simulator for the scenario. This mapping definition file is used by the model builder. Second file, SiMA configuration, includes all initialization data for each atomic model in the scenario.

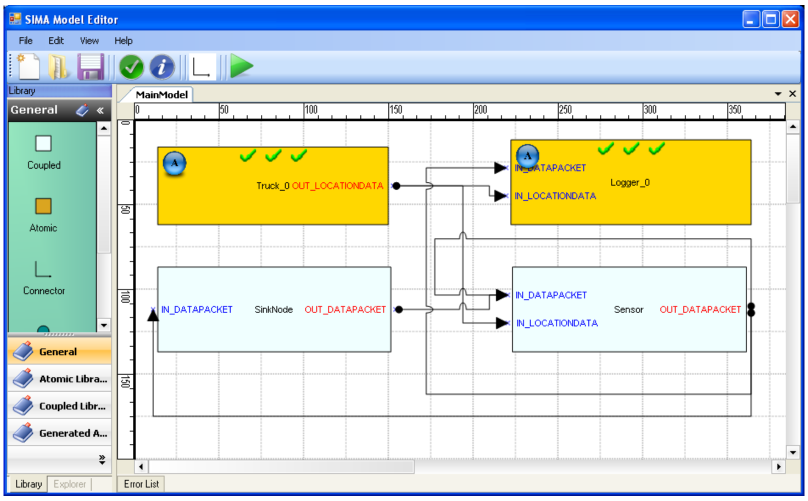


Figure 2.3: Example Model Structure

- **Model Builder:** Builder is the Sima tool that reads the linker output and generates the real model that will be executed by a simulator. Dissimilar to Linker, Builder is a worker that does the given tasks in order. It reads the document and:
  - Constructs atomic models,
  - Creates coupled models with their input and output ports,
  - Applies the requested port couplings. The final output of the model builder is a coupled model that has the exact structure requested in the Model Link Map.

CHAPTER 3

RELATED WORK

In the relevant literature, there are two main approaches to the dynamism support problem. In this section a brief introduction to these formalisms will be given.

3.1 DSDEVS

The Dynamic Structure Discrete Event System Specification (DSDEVS) is introduced by F.J. Barros in 1995 [3–5]. In this approach, atomic model definitions remain the same as classic DEVS formalism, but classical coupled model definition is extended. An executive atomic model, which decides when and what kind of structural operations will take place, is added to each coupled model definition. In other words, each coupled model is associated with a dynamic structure atomic model which handles the structural changes in its associated coupled model. This executive model has no behavioral logic other than structural change operations and its state is the current structure of the associated coupled model.

A DSDEVS network model is formally described as follows: $DSDEVSN = \langle\ X_\Delta,\ Y_\Delta,\ \chi,\ M_\chi\ \rangle$ where Δ is network name; χ is the name of DSDE network executive; M_χ is the model of χ; X_Δ is the set of input events; Y_Δ is the set of output events. M_χ, the model of the network executive χ, is a basic DSDE model and defined as: $M_\chi = \left\langle X_\chi, S_\chi, Y_\chi, \delta_{int_\chi}, \delta_{ext_\chi}, \lambda_\chi, ta_\chi \right\rangle$

M_χ contains information about network composition and coupling. A state $s_\chi \in S_\chi$ has information about the structure of the network model and it is defined as: $s_\chi = \left(D_\chi, \{M_i^\chi\}, \{I_i^\chi\}, \left\{Z_{i,j}^\chi\right\}, SELECT^\chi, V^\chi\right)$ where D_χ is the set of component names;

M_i^χ is the model of component i, for $i \in D^\chi$; I_i^χ is the set of component influencers of i, $\forall i \in D^\chi \cup \{\chi, \Delta\}$; $Z_{i,j}^\chi$ is the i-to-j output to input translation function, $\forall j \in I_i^\chi$; $SELECT^\chi$ is the tie-breaker function; V^χ represents other state variables of the network executive.

In DSDEVS, only the network executive can make structural changes and any change made in one of these 5-tuples $\left(D_\chi, \{M_i^\chi\}, \{I_i^\chi\}, \left\{Z_{i,j}^\chi\right\}, SELECT^\chi\right)$ will be automatically reflected to the structure of the network model. A detailed explanation of DSDEVS formalism can be found in [5], and abstract simulators necessary to simulate DSDEVS models can be found in [6].

3.2 DynDEVS

Dynamic DEVS (DynDEVS) is the second approach for supporting dynamic structure models. It is introduced by A. M. Uhrmacher in 2001 [22]. Unlike DSDEVS, DynDEVS formalism does not introduce a specific type of model (i.e. the network executive model) to apply structural changes dynamically. Instead, transition functions, ρ_α and ρ_N are added to the atomic and coupled model definitions respectively to support dynamism. There are two types of models defined in DynDEVS formalism, which are dynDEVS (atomic) and dynNDEVS (coupled) models. Atomic models in DynDEVS formalism are formally defined as follows: $dynDEVS = df \langle X, Y, m_{init}, M(m_{init})\rangle$ where X,Y are structured sets of inputs and outputs; $m_{init} \in M(m_{init})$ is the initial model; $M(m_{init})$ is the least set having the structure $\langle S, s_{init}, \delta_{int}, \delta_{ext}, \lambda, ta, \rho_\alpha\rangle$ where S is the set of states; $s_{init} \in S$ is the initial state; $\delta_{int}, \delta_{ext}, \lambda, ta$ are the same functions as in classical DEVS formalism; $\rho_\alpha : S \rightarrow M(m_{init})$ is the model transition function. This transition function is capable of making changes only on its own atomic model. It can change or remove itself.

Coupled models, which are composition of components and the links between these components, are described in DynDEVS formalism formally as follows: $dynNDEVS = \langle X, Y, n_{init}, N(n_{init})\rangle$ where X, Y are structured sets of inputs and outputs; $n_{init} \in N(n_{init})$ is the start configuration; $N(n_{init})$ is the least set having the structure $\langle D, \rho_N, \{dynDEVS_i\}, \{I_i\}, \{Z_{i,j}\}, Select\rangle$ where D is the set of component names; $\rho_N : S \rightarrow N(n_{init})$ is the network transition function with $S = \times_{d \in D \oplus m \in dynDEVS_d} S^m$

; $dynDEVS_i$ is the dynamic DEVS models with $i \in D$; I_i is the set of influencers of i; $Z_{i,j}$ is the i-to-j output-input translation function; Select is the tie-breaking function.

In DynDEVS formalism, there are some operational boundaries. Just like atomic models, coupled models in DynDEVS formalism also cannot make structural changes outside of its enclosing model. Also, in DynDEVS formalism, dynamic port management is not supported. Allowed operations include dynamic model and coupling addition and removal. More details about DynDEVS formalism can be found in [11, 22].

CHAPTER 4

OUR APPROACH

4.1 ADDING DYNAMISM TO SiMA-DEVS

Our approach to add dynamism to our basic DEVS model is similar to that of DynDEVS as indicated earlier. To be more precise, we conform to both dynDEVS and dynNDEVS definitions as the underlying formal specification, with some extensions which are given below:

1. We state that structured sets of inputs and outputs are defined in conformance to our strongly typed-port definitions where the formal definitions for P_{in}, P_{out} apply; M (m_{init}) is the least set having the structure $\langle S, s_{init}, \delta_{int}, \delta_{ext}, \lambda, ta, \rho_\alpha, \delta_{df}\rangle$ where $S, s_{init}, \delta_{int}, \delta_{ext}, \lambda, ta, \rho_\alpha$ are the same as in DynDEVS formalism; δ_{df} is the additional transition function defined in SiMA-DEVS. Thus, we ensure that the meaning of a model in DynDEVS is firmly aligned with that of SiMA-DEVS, while maintaining the top-level semantics of the DynDEVS definition. Note that SiMA-DEVS extensions are non-disruptive to the overall semantics of the basic DynDEVS formalism.

2. We introduce a state synchronization mechanism between networks of connected models, to be performed at the end of a structural change phase, in case a model wants to update the values of such state variables that are within the common set of pre-and post change models (i.e. they are not introduced newly after the model's structural transition) but have values that stayed unchanged during pre-change simulation period. This mechanism is instrumental in cases where a model A initializes some of its state variables at the beginning of simulation

but does not receive updates for those variables until some influencer model B goes through a structural change that causes those variables to be updated; or in cases where a new model B is added which introduces a new coupling influencing one of the input ports of model A. One might argue that after the structural change, synchronization of such state variables would already take place as a result of message passing via the coupling links during the normal course of the simulation. However, it is important to note that due to differences in state update rates (i.e. different step sizes), an influencee may have to go through many state updates and produce many output sets before it can receive the required updates from slower influencers; a case which might potentially lead to significant errors in the behavior of the overall simulation, especially if the simulation application is developed for an engineering analysis requiring a high level of behavioral sensitivity. It is worth mentioning that the state synchronization function must be executed as the last step of the structural change transition phase to allow the influencers to perform the necessary state updates before the influencees ask for the latest values of the state variables that need to be synchronized

A variable structure atomic model is defined formally as follows:

$VS\_AM = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta, \delta_{df}, SO, \rho_{\alpha} \rangle$
where,
$X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta, \delta_{df}$ are same as basic SiMA-DEVS formalism,
SO is the set of structure change operations,
$\rho_{\alpha} : S \times SO \rightarrow S'$ is the structure change transition function. ρ_{α} defines a mapping from pre-change state and structure change operations $(S \times SO)$ to post-change state (S').

A variable structure coupled model is defined formally as follows:

$VS\_CM = \langle X, Y, n_{init}, N(n_{init}) \rangle$
where,
$n_{init} \in N\,(n_{init})$ is the start configuration;
$N\,(n_{init})$ is the least set having the structure

$\langle D, M_i, I_i, Z_{i,j}, \gamma, \tau_i \rangle$

where,

D is the set of component names,

M_i is the model of component i, for $i \in D$,

I_i is the set of component influencers of i,

$Z_{i,j}$ is the i-to-j output-to-input translation function, $\forall j \in I_i$,

$\gamma : SO \times S^{N(n_{init})} \rightarrow S^{N(n_{init})}$ is the network change structure transition function where,

$S^{N(n_{init})} = \times_{d \in D} S^{M_d}$

in which $S^{N(n_{init})}$ is the selective topological sum of the states of the models that are to be affected by the structural change operations.

$\tau_i : S_{M_i} \rightarrow S'_{M_i}$ is the state synchronization function where,

$S'_{M_i} = \bigcup_{j \in I_i} \{\Pi_L(\sigma_C(S_j))\}$

In other words, τ_i computes synchronized state (S'_{M_i}) for M_i, by applying selection(σ) and projection(Π) operations on the states of some of the influencers of model M_i, then producing a union of those states. In this definition, L denotes the reduced (projected) tuple for S_j and C denotes the condition of the selection.

In this definition, when structural change is initiated by top-level simulator, γ transition function is executed. A structure change request initiated by top-level simulator is disseminated to the children coupled models in a hierarchical way. Each coupled model, receiving the request, applies the structural changes relevant to it and passes the requests to other children that are relevant(i.e. addressed by the request). On the other hand the network transition function can be executed when structural change is initiated by a leaf-level atomic model. In this case, each coupled model collects request messages from its children, applies changes relevant to it and passes upper-level requests to its parent coupled model.

4.2 OUR IMPLEMENTATION APPROACH

We now set out to describe the principles that govern the run-time algorithms of the SiMA simulation engine when it manages structural change. In SiMA-DEVS, like

DynDEVS, atomic models are responsible for initiating structural changes. There is no dedicated controller model that supervises over atomic models as described in DSDE formalism. This model centric approach seems to be more reasonable since most of the potential change-triggering events that require structural changes from a particular model are naturally handled by the external transition function of that atomic model, and it is that particular model which should have the knowledge of re-structuring itself, whether this re-structuring is a switch to an internally defined different functional model, or a re-adjustment of its port couplings. The only exception where the model-centric approach may become restrictive is the case where a new model (atomic or coupled) is to be added to the simulation. The logic for initiating the model addition may require the aggregation of state variables from many different models, or even it may be a user-initiated request which is not necessarily captured by a single model. In current implementations of DynDEVS namely AgedDEVS and JAMES, the atomic models are assumed to have access to a knowledge base from where they can collect the necessary information to decide for new model additions. Although this approach seems quite reasonable for agent-oriented implementations, it introduces a dependency to a specific architectural and behavioral semantics for simulation applications, which we are inclined to avoid. Therefore, in our approach:

- If an atomic model requires a structural change, it informs its parent coordinator about the type and content of the operations to apply. Coordinators store all structure change requests until all child models complete their operations. These requests will be non-ambiguously aggregated, since SiMA's type-safe model composition semantics enables the resolution of any potential overlapping requests. An atomic model can create and send structure change requests to its parent coupled model, but cannot change the structure itself. Coupled models process these messages and executes the operations restricted to their bounding coupled model and sends the upper level requests to their parent coupled model.

- An application that is running the simulation may require structural changes, too. This request is sent to the root coordinator to be executed over the model structure recursively. Root coordinator implements an interface that allows applications to send their structural change requests to the simulation engine. This operation is applied in two parts:

  - Before applying the change operation, simulation is suspended at the beginning of the next cycle.
  - The change request is processed by the root coordinator and child model operations are sent to the child coordinators recursively, causing all related child coordinators to apply change operations specified in the request.

4.2.1 Operations on Model Structures

There are three types of structural change operations defined in SiMA: Adding/removing a model, adding/removing a coupling and adding/removing a port.

- *Removing a model*: This operation consists of two steps: Removing all the connections from/to the model. Removing the model.
- *Adding a model*: This operation consists of three steps:
  1. Adding a model to the parent coupled model.
  2. Calling `'init()'` function of the newly added model.
  3. Calling `'AdvanceTime(CurrentTime)'` function for synchronization.
- *Adding/Removing a coupling*: Adding a coupling is also a critical operation in our case. After adding a coupling, a process for synchronizing current states of newly connected models is executed. For achieving this, a querying mechanism between connected ports is implemented that operates in the opposite direction of the normal message flow. An input port creates a query and sends this query to the newly connected ports. An answer to this query is generated and sent to the requesting port. These response messages will be handled when the external transition function of the model is executed.
- " *Adding/Removing a port*: This operation supports the addition of new ports to coupled models. Before removing a port, all couplings from/to the port is removed. Note that the new ports must conform to one of the existing port types (i.e. the type space can not be extended at run time).

Our framework does not support the addition and removal of new port types to the type space of the simulation at run-time. One rationale for this is to preserve the models' external identity as advocated by [22]. Another important reason for such a restriction is the implied ambiguities in the run-time behavior of source and sink models of the newly added ports with new port types. To be more specific, say for instance, a new output port of a new type is to be added. This would normally cause new connections to be established between its source and some other sink model. To be able to process the data coming from the new port type, the sink model(s) have to be structurally and behaviorally ready to receive, interpret and process data coming from the new port. In a type-safe environment where port connectivity is regulated and restricted by type compatibility between connected ports (which is the case in SiMA), normally a new port will have to be added to the sink model too. However, both the source and the sink model may not know in advance the processing logic of the information flowing through those new ports. As such, such a support would rely on the pre-existence of sophisticated application-specific semantics within the models. We believe this case should be avoided for generic frameworks and therefore we exclude this functionality. However, we do find addition of ports having a port type already defined in the current type space useful, since it is likely to have an already defined port with the same type in one of the existing models and it is reasonable for a model to add a port to establish a new coupling with an existing model.

For an example where some of these operations are applicable, consider a simulation scenario involving two planes flying in formation. A graphical representation of the models involved in this scenario can be seen in Figure 4.1. When the simulation starts execution, the models representing the planes send their properties to each other from their ports once and subsequently they only send their current locations and directions, which are the only updated parameters of the planes throughout the simulation period. Assume that at some point in time, a third plane is to be added to the simulation to connect to the existing planes. This updated model can be seen in Figure 4.2. Since the first two planes send only their updated parameters, which are location and direction, newly added plane will not be aware of the remaining two planes' properties. Therefore a state synchronization is required.

Dynamic SiMA handles this case by implementing an automated state synchronization mechanism via a querying system between connected port pairs. When a coupling is added to the model structure while the simulation is running, this querying system

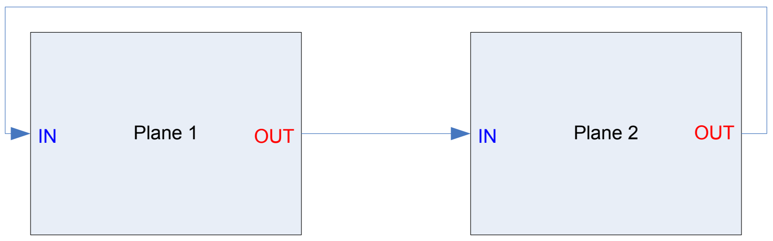


Figure 4.1: Initial Model

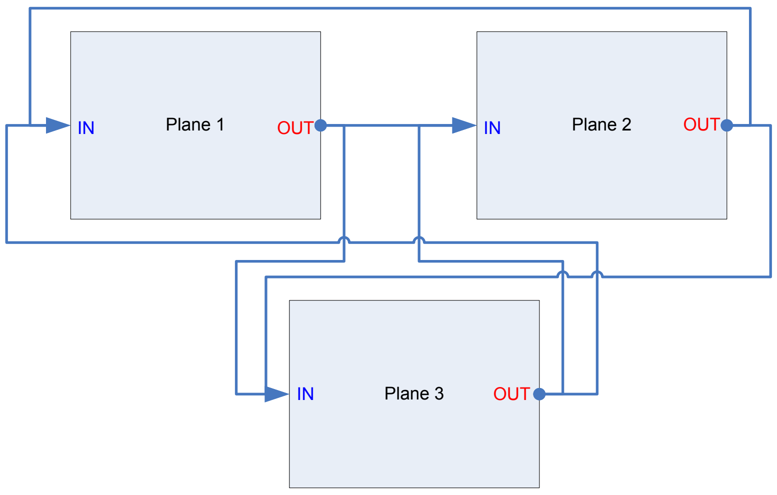


Figure 4.2: Updated Model

automatically works as a service provided by the infrastructure, without incurring an additional implementation overhead on the model developer. A more detailed discussion of the state query mechanism is provided in Section 4.2.3.

4.2.2 SiMA Abstract Simulators Adapted for Dynamism Support

Recall that SiMA is an implementation of SiMA-DEVS formalism as discussed at the beginning of this section. SiMA run-time layer is implemented in C# programming language but it can interface to models implemented in both C++ and C# programming languages. In this section, extensions to abstract simulators required for executing variable structure SiMA models are described in pseudo code format:

Root Coordinator The algorithm that is executed at the top-level root coordinator is shown in Algorithm 4.1. It can be observed in this algorithm that, there are three types of simulation cycles: Change structure initiated by an external request, change structure initiated by a leaf level atomic model and normal simulation cycle.

After a structural change operation, all models that have new couplings will execute state synchronization mechanism, discussed in Section 4.2.3, to update their state information. `'ChangeStructure'`and `'GetNextTime'`functions of both simulators and coordinators correspond to the case when a message of type sc and @ are received from the parent models in [6, 11, 20] respectively.

Coordinator

In `'GetNextTime'`function, shown in Algorithm 4.2, next simulation time and whether any structural change is required at that time is resolved recursively down the model hierarchy and the result is sent back to the parent coordinators up the hierarchy.

If a change structure step is initiated by a leaf level atomic model, in each coupled models coordinator, Algorithm 4.3 is executed. On the other hand, if the step is initiated by top level root coordinator, similar but the inverse algorithm shown in Algorithm 4.4 is executed. The basic idea in both algorithms is applying necessary updates in the coupled model they are associated with and redirecting the remaining requests to the places where they will be applied.

```
while simulation end condition not satisfied do
    if Structure change requested from top level then
        Process change request
        Send subrequests to related child coordinators
        Do state synchronization
        Initialize newly added models
    end if
    $\langle CurrentTime, StructureChangeRequested \rangle \leftarrow MainModel.GetNextTime()$
    Advance simulation time to CurrentTime
    if StructureChangeRequested is True then
        Execute a structural change step
        Do state synchronization
        Initialize newly added models
    else
        Execute a normal simulation cycle
    end if
end while
```

Algorithm 4.1: Root Coordinator

```
for all inner model $M$ do
    $\langle time, structureChangeRequest \rangle \leftarrow M.GetNextTime()$
    if time < minTime then
        $minTime \leftarrow time$
        $commonStructureChangeRequest \leftarrow structureChangeRequest$
    else if time = minTime and structureChangeRequest is True then
        $commonStructureChangeRequest \leftarrow True$
    end if
end for
```

Algorithm 4.2: Recursive Next Time Calculation

```
1: for all inner model $M$ do
2:     if M requested structure change then
3:         Call M's change structure function
4:         Add M's change requests to changeReq set
5:     end if
6: end for
7: Process changeReq set
8: Apply necessary updates in current level
9: Send upper-level operations to parent model
```

Algorithm 4.3: Structure Change in Coordinator - From Bottom

```
1: Process requests
2: for all inner coupled model $C$ do
3:     if a request exists for model C or its submodels then
4:         Send related requests to model C
5:     end if
6: end for
7: Apply necessary updates in current level
```

Algorithm 4.4: Structure Change in Coordinator - From Top

Simulator

In `'GetNextTime'` function of the simulator, shown in Algorithm 4.5, next internal transition time and an indication of whether any structural change request exists at that time are sent to the parent coordinator. Structural change function of an atomic model is executed if and only if its next time is imminent and a structural change request has been made by that model.

```
1: if StructureChangeRequired is True and simulation time = $t_N$ then
2:     Call $\rho_\alpha$
3:     $StructureChangeRequired \leftarrow False$
4:     Send required change packages to parent model
5: end if
```

Algorithm 4.5: Structure Change in Simulator

If the state of an atomic model satisfies certain conditions that require structural changes, atomic model marks itself and informs its simulator to initiate a structural change process and this simulator recursively sends this request to the root coordinator. Structural change requests can be issued by any atomic model during one of its transition functions. These requests are handled in the next internal transition phase. Modifications required for supporting variable structure models can be summarized as follows:

- A property, named 'StructureChangeRequired', is added to the atomic models' simulators.
- Atomic models that may require structural changes while the simulation is running implement ρ_α transition function.
- The get-next-time functions of the coordinators and simulators are modified and they now return a `'StructureChangeRequired'`flag, too. To initiate a structural change, an atomic model simply sets its `'StructureChangeRequired'`flag to true.
- When a structure change request arrives at the root coordinator with the minimum advanced time value, a structure change step is executed. For each atomic

model that requires structural changes at the new current time, the change structure transition function is executed.

4.2.3 State Synchronization Mechanism

In SiMA, there is a state query mechanism between connected ports. A port can create a query and send this query to other source ports to which it is connected. This mechanism works in the opposite direction of the normal message flow and it is instrumental in supporting the implementation of variable structure models. It enables newly added models or newly added couplings to acquire the current state of the simulation. This capability is crucial for SiMA, since ports are managed by event and object managers where object managers send only modified data for efficiency reasons. Therefore a sink model would not have up-to-date values of certain state variables from the source models if before the structural change the sink model did not use those particular state variables. If a model requires the previously updated fields, it can prepare and send a query to gather this information. Implementation details of this mechanism are discussed below.

An interface named `'IPortValueSource'`is defined in SiMA as below:

```
1: QuerySource(destModel:string, destPort:Port):$Message$ []
```

Algorithm 4.6: IPortValueSource Interface

This interface has only one member function which takes destination model and destination port as input parameters and returns port's related data. In a structural change step, after all structural updates are completed, the states of the models are updated accordingly as illustrated in Algorithm 4.7. Each coupled model contains a list of couplings, in its level, that are added in the last structural change step.

Destination ports create queries and send these queries to source ports that are connected to them. When an atomic model receives a query, it sends its state as a response. When a coupled model receives a query, it redirects this query to the source ports that are connected to this port and collects and returns the responses received from those redirected ports. For example, in Figure 4.3 a model named 'D' and a coupling from C's 'Out1' port to D's 'In1' port is added dynamically. In this example,

```
1: for all inner coupled model $M$ do
2:   Call state synchronization procedure of M
3: end for
4: for all new coupling $c$ do
5:   Add destionation port to the affected ports list
6:   Message[] messages $\leftarrow$ c.SourcePort.QuerySource(destination model, destination port)
7:   for all Message m in messages do
8:     Put values of m into destination port
9:   end for
10: end for
11: for all affected port $P$ do
12:   Synchronize sources of P
13: end for
14: Clear new couplings list
```
Algorithm 4.7: Updating States

state synchronization process works as follows;

1. 'In1' port of model 'D' sends a query to 'Out1' port of model 'C'.
2. 'Out1' port of model 'C' redirects this query to 'Out1' ports of model 'A' and model 'B'.
3. 'Out1' port of model 'C' collects response messages from 'Out1' ports of model 'A' and 'B'.
4. Sends these messages back to 'In1' port of model 'D'.

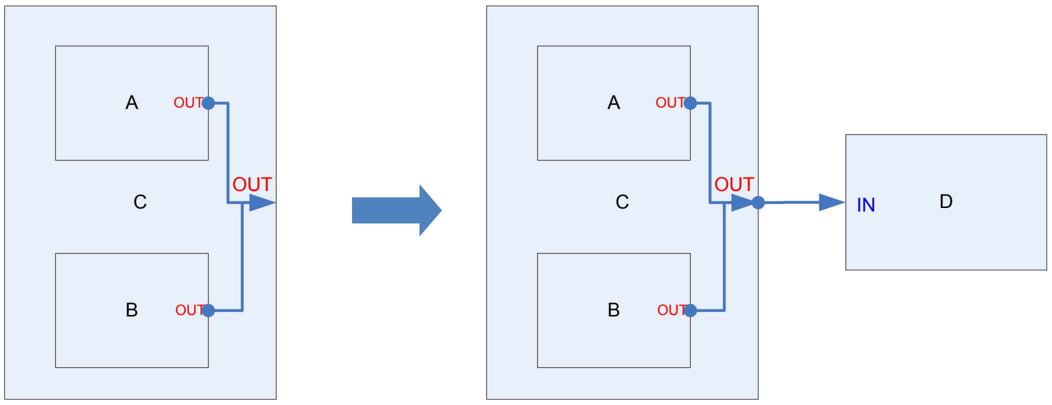


Figure 4.3: Dynamically Adding A Model and A Coupling

State synchronization mechanism works from bottom-to-top. In other words, parent models execute state synchronization mechanism after the completion of all of its submodels. As it was mentioned before, communication in SiMA is bidirectional and destination ports are aware of the source ports they are connected to. However, while updating the states, destination ports are not aware of the source ports that they are newly connected to. As it can be observed in Algorithm 4.7, sources of destination ports are synchronized after the state updating procedure. In this way, SiMA prevents unnecessary queries as well as duplicate messages, therefore implements state synchronization in an efficient way. For example, in Figure 4.4 two couplings are added dynamically; from 'OUT2' port of 'C' to 'IN' port of 'D' and 'IN' port of 'D' to 'IN' port of 'E'. In this example, state synchronization process works as follows;

- From 'IN' port of 'D' to 'IN' port of 'E':

1. 'IN' port of model 'E' sends a query to 'IN' port of model 'D'.
2. 'IN' port of model 'D' redirects this query to 'OUT1' port of model 'C'.
3. 'OUT1' port of model 'C' redirects this query to 'OUT' port of model 'A'.
4. Response messages of model A are sent to the model 'E' from the same path.

- From 'OUT2' port of 'C' to 'IN' port of 'D':

1. 'IN' port of model 'D' sends a query to 'OUT2' port of model 'C'.
2. 'OUT2' port of model 'C' redirects this query to 'OUT' port of model 'B'.
3. Response messages are sent to 'IN' port of model 'D' from the same path.
4. 'IN' port of model 'D' sends a copy of these messages to 'IN' ports of both model 'E' and 'F'.

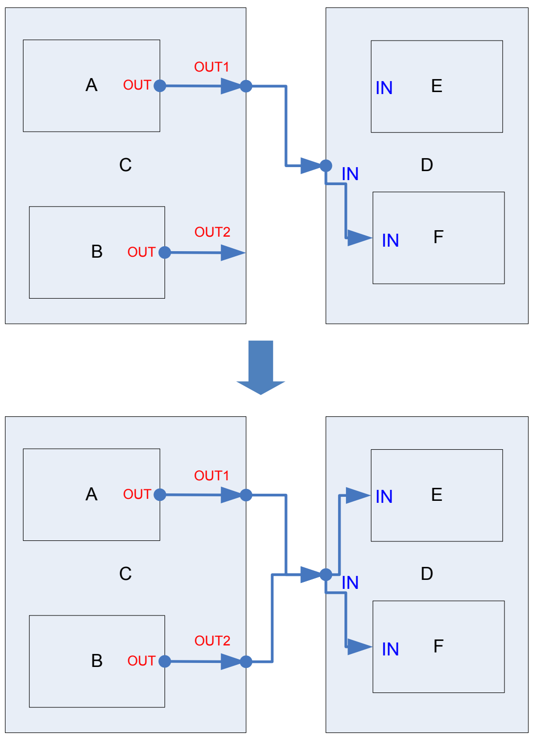


Figure 4.4: Dynamically Adding Couplings

4.2.4 Change Request Message Structure

Change requests in SiMA are defined as XML documents that are compatible to a XML Schema shown in Figure 4.5 and this allows interoperability for structural changes between different simulations.

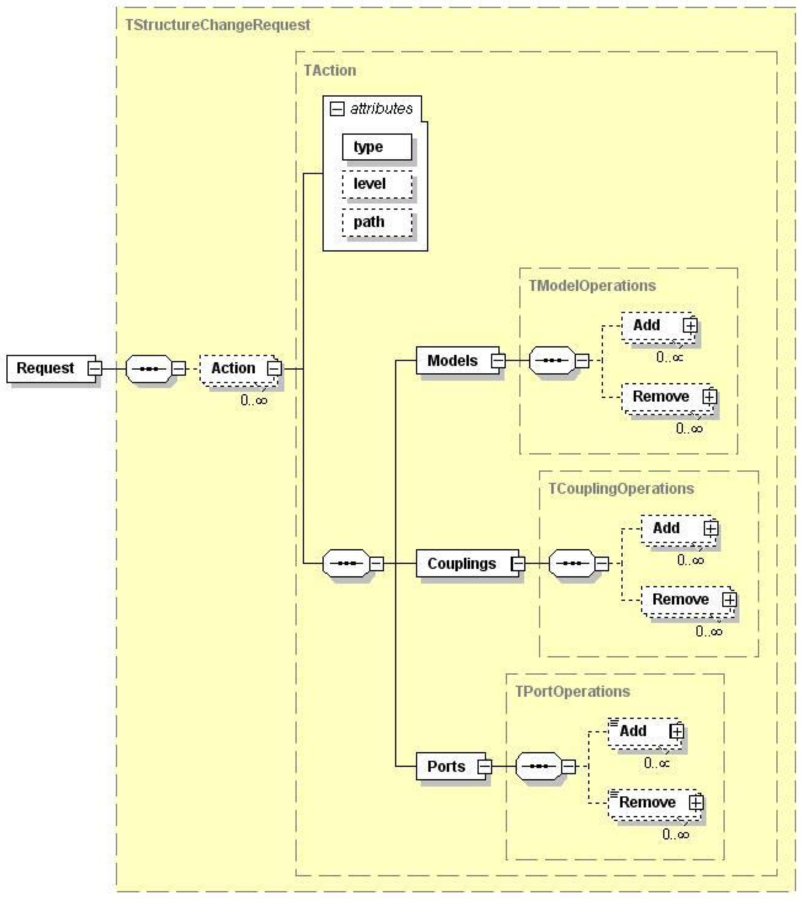


Figure 4.5: Change Request Message Structure

A change request consists of several actions and each action consists of attributes and structure change operations. Attributes of an action specify the destination model that will execute structure change operations defined in the action. An action has three attributes; `type`, `level` and `path`. Level and path are optional attributes and their data types are int and string respectively. Type attribute of an action can have one of the following values: `Relative path`, `Absolute path`, `Relative level` and `Absolute level`. For example, if type attribute is set to relative level and level attribute to one, then the action will be executed in the parent coupled model. If level attribute was set to two, then the action will be executed at the parent coupled model of the parent coupled model and so on.

As it is discussed in Section 4.2.1 and as it can be observed in Figure 4.5, allowed operations in dynamic SiMA are:

- Adding/Removing models
- Adding/Removing couplings
- Adding/Removing ports

In dynamic SiMA, requests can be sent both from top-to-bottom and bottom-to-top (see Section 4.2.6). For both purposes, the same schema, introduced in this section, is used. By using the functionalities of this schema, for messages sent from top-to-bottom, requests can be broken into pieces and for messages sent from bottom-to-top, requests can be combined.

4.2.5 Time Management In Dynamic SiMA

The time management diagram of a simulator in dynamic SiMA can be observed In Figure 4.6. The parts with red color are the dynamism extensions to the time management flow of classical DEVS formalism. After each simulation cycle, whether there exists a structure change request in the next cycle is controlled by the simulator. If there exists such a request and if current simulation time equals to the change request time then the required changes are executed. Changing model structure in SiMA is handled in three steps:

- Changing the model structure
- Synchronizing states (details are discussed in Section 4.2.3)
- Initialization of newly added models. Current times of newly added models are also advanced to the current simulation time.

After changing the structure, this cycle continues with the updated model structure.

4.2.6 Structure Change Types

In DSDEVS, network executive models, which are associated with each coupled model, and in DynDEVS, atomic and coupled models initiate structure change steps. On the

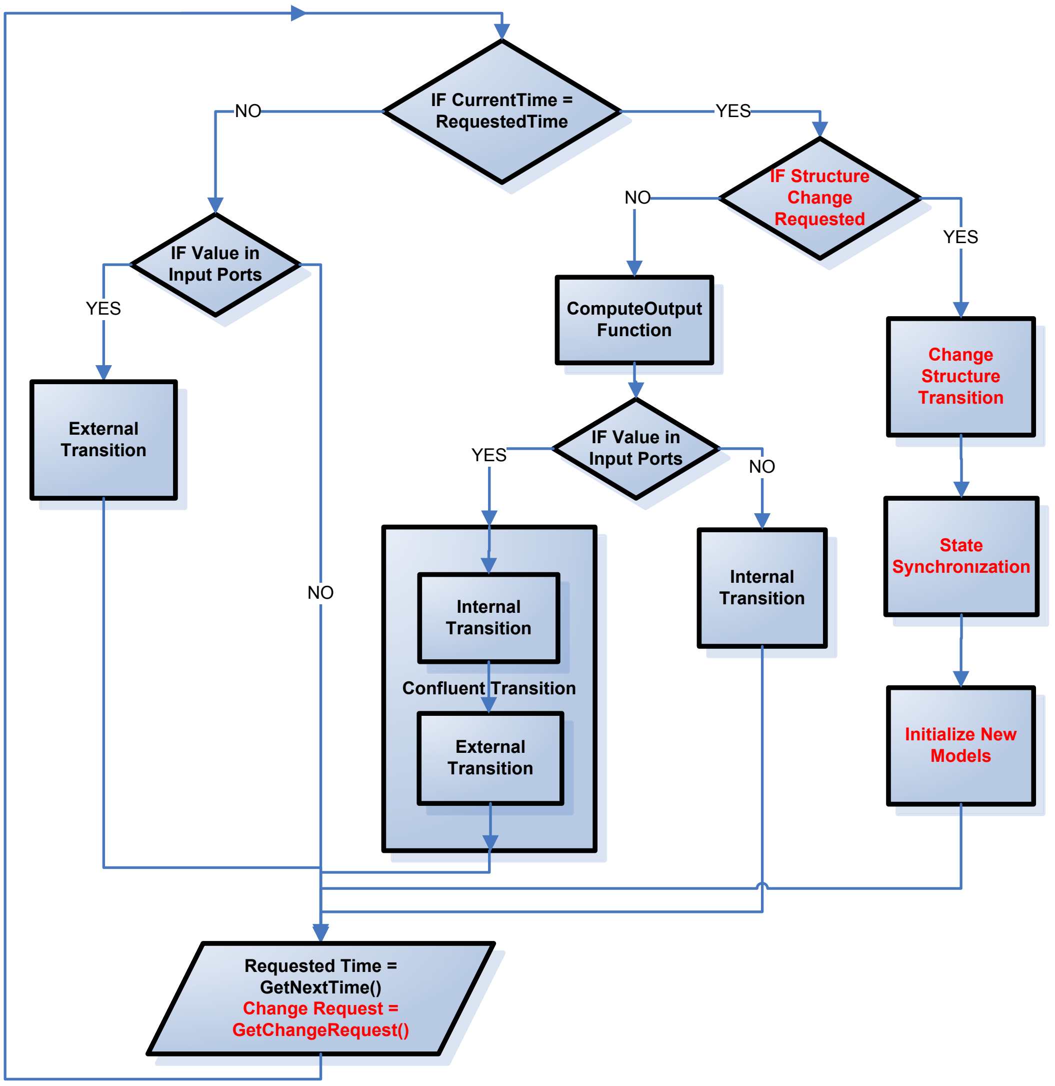


Figure 4.6: Time Management In Dynamic SiMA

other hand, in dynamic SiMA, atomic models, due to their internal logic, and root coordinator, due to an external request can initiate a structure change step. When atomic models initiate a structure change step, request messages are sent from leaf level atomic models to upper level coupled models and when root coordinator initiates a structure change step, request messages are sent from top-most coupled model to the lowest level coupled models.

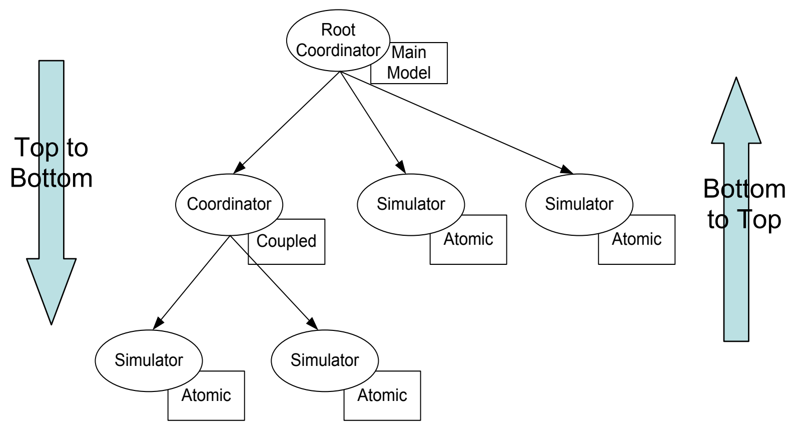


Figure 4.7: Change Request Types

CHAPTER 5

CASE STUDY and PERFORMANCE ANALYSIS

5.1 Wireless Sensor Network Simulation

In this section, performance measurements and analysis of SiMA environment with dynamism extensions will be presented. To conduct the performance tests, a Wireless Sensor Network (WSN) simulation has been developed as a sample case. Models representing the sensors will be implemented in both classic SiMA and dynamic SiMA frameworks and their performances will be compared. A top level visual representation of DEVS models, which are defined according to [1, 19], is given in Figure 5.1. The proposed WSN system consists of five components:

1. **Sensors** sense the movement activities in the environment and can communicate with other sensors within their range. Each sensor is represented by a coupled model and consists of four subcomponents. These components and their inner relationships can be observed in Figure 5.2.

    - **Antenna** is used for communicating with main sensor and other sensors within their range. It is the intermediate model between outside world and the processor. Messages coming from the outside world are sent to the processor and messages received from the processor are sent to the other sensors. Routing protocol is also implemented in this model. For this case study, greedy forwarding which is the simplest form of geographic routing is used. Each node makes decisions according to the locations of its direct neighbors. Each sensor sets the neighbor that has the shortest distance to the sink model as its parent model. More details about greedy forwarding

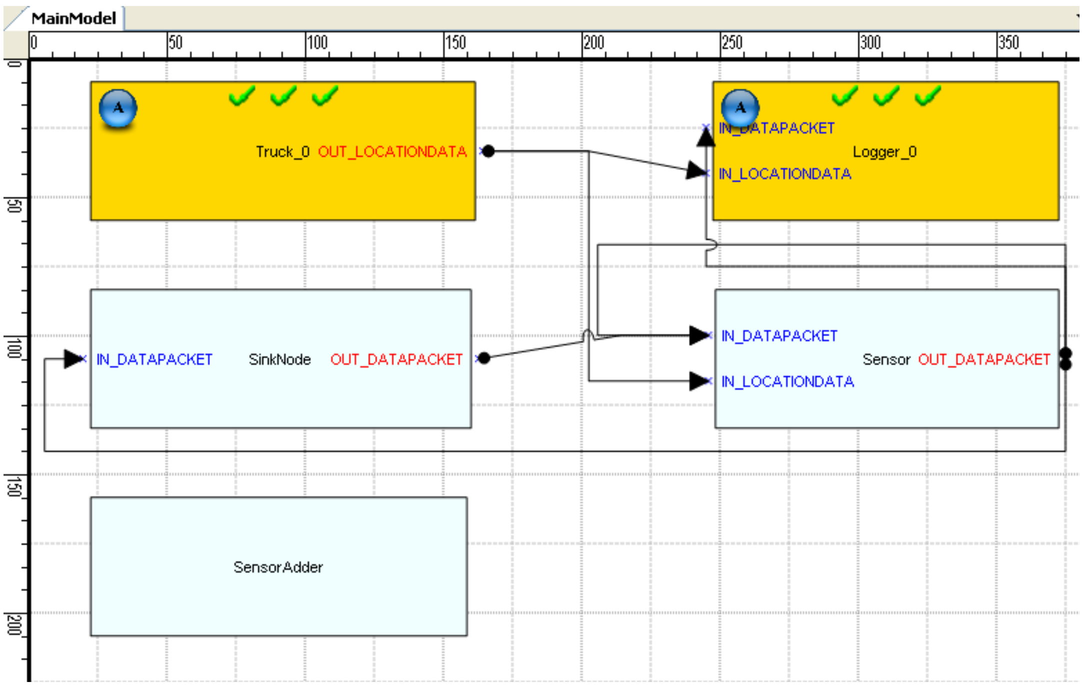


Figure 5.1: WSN Main Model

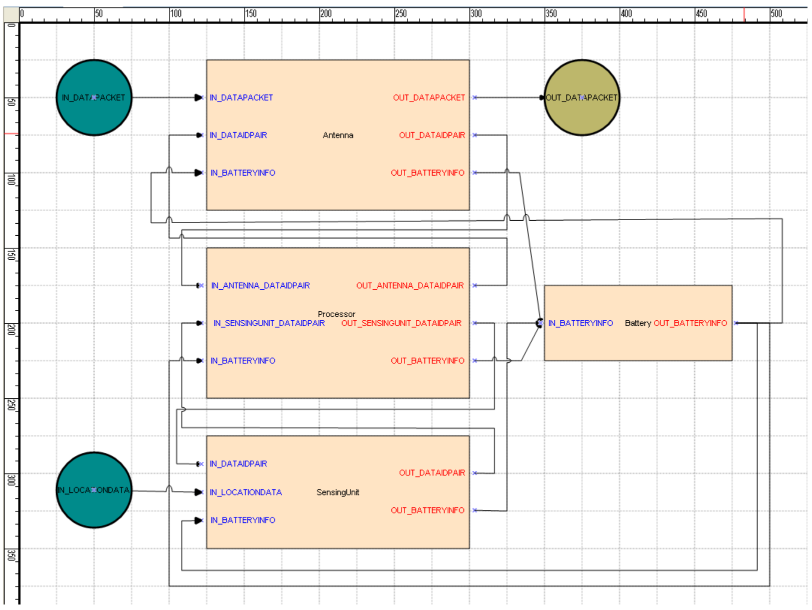


Figure 5.2: Single Sensor Model

can be found in [15].

- **Sensing Unit** is used for sensing the movement activities in the environment. When a sensing unit detects some movement activities it estimates a value between $[0, 1]$ and sends a message containing this data to the processor.
- **Processor** receives messages from both antenna and sensing unit. It creates and sends data messages to the antenna according to messages received from sensing unit. When processor receives a message from antenna which is originated from sink sensor, it sends an activation message to its sensing unit.
- **Battery** is connected to antenna, processor and sensing unit. It has a power and when this power runs out battery model informs other models with an event message and all models in the sensor changes their states to 'Dead' phase. In classic implementation, when battery power runs out, sensor only changes its phase to 'Dead', but still remains in the simulation. However, in dynamic version, sensor is also removed from the simulation structure.

2. **Main Sensor** can communicate with the sensors within its range. It sends activation messages to the sensors and waits for the response messages. Unlike other sensors, main sensor does not sense the environment and it does not contain a battery unit. Main sensor contains two subcomponents. These components and their relationships can be observed in Figure 5.3

   - **Sink Antenna** sends activation messages and listens to the incoming messages within a specific range. It redirects received messages to the sink processor.
   - **Sink Processor** follows trucks movement activities. During the simulation, it collects data messages received from the sensors. At the end of the simulation, it analyzes these messages and determines the locations and times in which the truck is passed. Analyzing trucks location for each time unit is handled as follows:
     - If only one sensor detects trucks location: Truck is estimated to be at exactly sensors location.

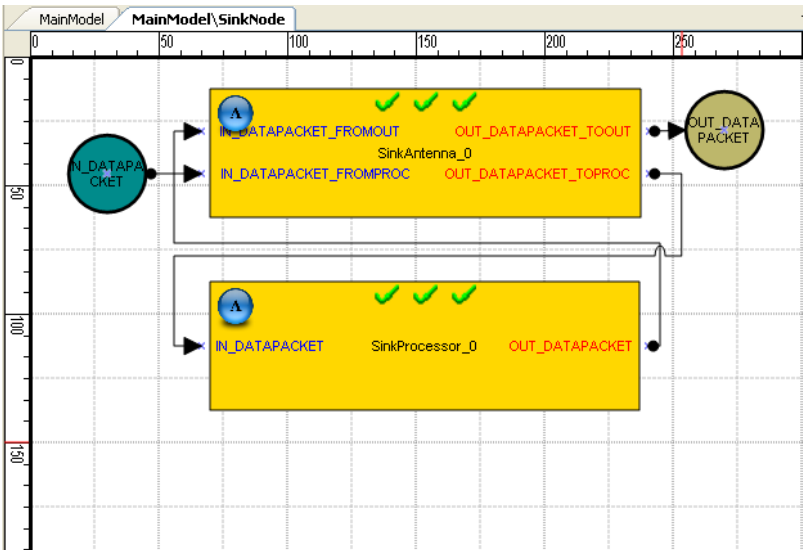


Figure 5.3: Main Sensor

  - If two sensors detect trucks location: Trucks location is estimated to be on the line connecting these sensors. According to the sensors calculated accuracy results, trucks location on this line is determined.
  - If at least three sensors detect trucks location: Three sensors are selected and using trilateration, discussed in **??**, exact location of the truck is determined.

3. **Truck** has a predefined path and follows this path during the simulation. Each sensor has a sensing range and when truck enters into a sensors range sensor prepares a message that contains accuracy data and sends this message to its parent to be delivered to the main sensor.

4. **Logger** saves all the location and data packages, created by truck and sensors respectively, in a file. This file can be used for analyzing simulation results.

5. **Sensor adder:** This model exists only in the dynamic version of the simulation and it is used for adding sensors at runtime.

5.1.1 Simulation Scenario

In this simulation, a wireless sensor network system is constructed. Sensors that are capable of sensing movement activities are randomly distributed. Each sensor has an antenna range, sensing range and lifetime. Sensors can communicate with the nodes within their antenna ranges and can sense movement activities within their sensing ranges. There also exists another type of sensor, namely main sensor (sink sensor). Sink sensor is connected to a computer and collects data messages coming from other sensors. When simulation starts running, a truck that has a predefined velocity and random path starts moving and follows its track during the simulation. When truck enters a sensor's sensing range, the sensor detects truck's location and sends an accuracy value in the interval $[0 - 1]$ to its parent to be sent to the main sensor. At the end, main sensor analyzes collected messages and determines truck's observed path.

In this case study, tests are performed according to the following scenario:

- A truck starts moving along a random path with a constant speed.
- Main sensor creates and broadcasts an activation message.
- When a sensor receives this message, it also broadcasts it to be able to distribute to the maximum number of nodes.
- Sensors that receive the activation message start collecting movement activities from the environment via their sensing units.
- When a sensing unit receives movement activities from the truck, it sends a message to its processor.
- Processor creates data messages according to the collected data from the environment.
- When messages are ready, processors send them to their antenna to be sent to the main sensor.
- Main sensor collects these data messages and understands the current location of the truck.

5.1.2 WSN Simulator Application

WSN Simulator Application is implemented as visual experimentation tool that is used for conducting the simulation runs, observing the behavior of wireless sensor networks via a visually plausible graphical interface and verifying simulation results both through textual output information and visual cues. The tool provides a controlled experimentation environment which ensures that both static and dynamic cases are tested in a systematic and consistent way.

When wireless sensor network simulator first started, the windows shown in Figure 5.4 opens. In this window two buttons with different functionalities exist. If the button with label 'Performance Tests' is clicked, all active windows gets closed and performance tests start executing. Parameters for these tests are read from a previously defined file and results are saved into a file to be used for later analysis. Results of the performance tests are discussed in Section 5.1.3.

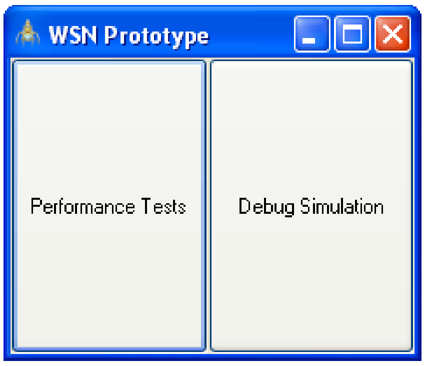
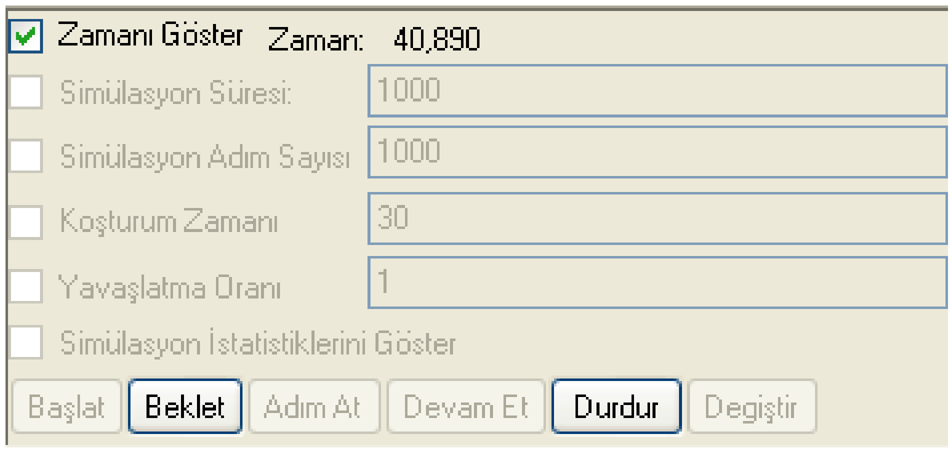


Figure 5.4: Choose Process Type Window

When user wants to debug the simulation or wants to see the behavior of the simulation, he/she clicks on the 'Debug Simulation' button in choose process type window. Then, a new window shown in Figure 5.7 opens. This window consists of five sections:

- **Simulation Parameters:** Values of these parameters are saved into config files and given to the related models in the initialization phase. By changing these parameters different simulations can be executed. For debugging reasons only three parameters, which are sensor count, sensor adder frequency and simulation type(static or dynamic) are set here, but if needed more parameters can be added.
- **Simulation Manager:** Controls a simulation. Before starting the simulation execution, simulation end conditions can be defined by simulation time units, execution time units or in step count. While a simulation is running, by using simulation manager, it can be paused or stopped. In Figure 5.5, simulation manager is shown while the simulation is running. When simulation is paused, allowed operations with simulation manager are: Iterating the simulation for one step, resuming execution, stopping the simulation and finally changing the model structure. In Figure 5.6, we can see the simulation manager when the simulation is paused. Changing the structure defined here is a top-to-bottom structure change operation that is discussed in Section 4.2.6 and structure change requests read from an XML document is sent from root coordinator to the related subcomponents to be applied.
- **Simulation Panel:** Visual representation of the running simulation is shown

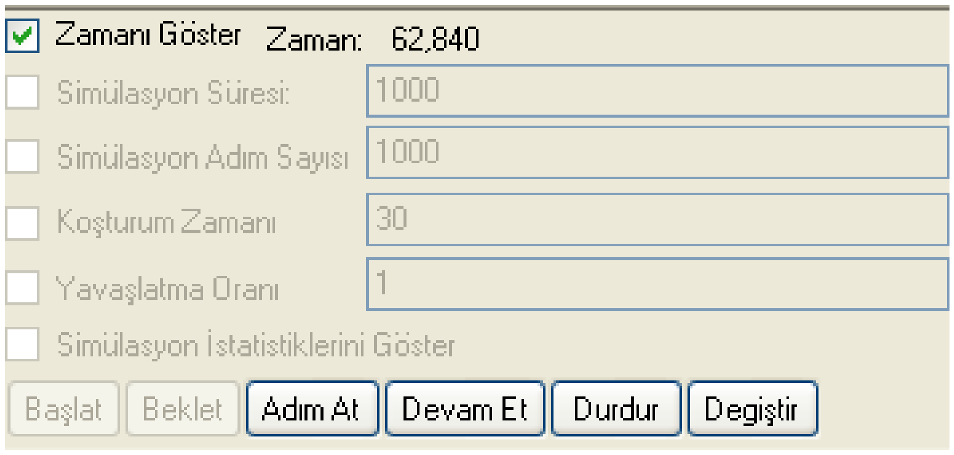


Zamanı Göster [✓] Zaman: 40,890

Simülasyon Süresi:	1000
Simülasyon Adım Sayısı	1000
Koşturum Zamanı	30
Yavaşlatma Oranı	1

Simülasyon İstatistiklerini Göster

Figure 5.5: Simulation Manager While Simulation Running



Zamanı Göster [✓] Zaman: 62,840

Simülasyon Süresi:	1000
Simülasyon Adım Sayısı	1000
Koşturum Zamanı	30
Yavaşlatma Oranı	1

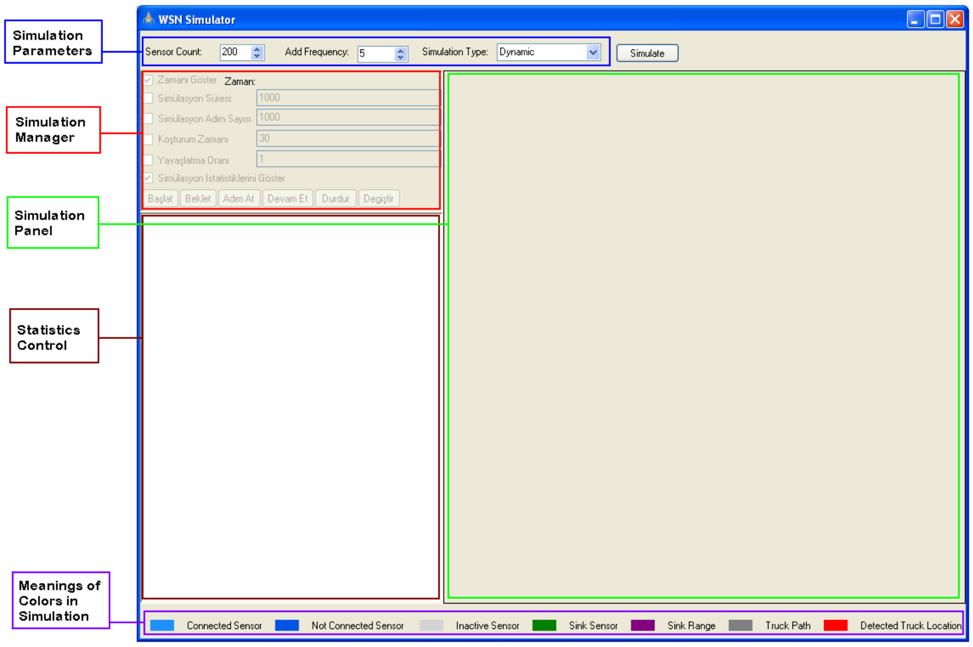
Simülasyon İstatistiklerini Göster

Figure 5.6: Simulation Manager When Simulation Paused

in this section of the screen. Trucks path is shown with a polygon and sensors are represented by dots with different colors according to their states.

- **Statistics Control:** Statistics of completed simulations are shown in this section. By just looking into the visual representation of the simulation it is hard to make sure that both static and dynamic simulations have executed the same simulation. So, statistical data are used for determining this. In addition to the simulation execution time and simulation type, how many times the truck is located by how many sensors is also shown as statistical data and this is the main metric we used for making sure that we are executing the same simulation with different approaches.

- **Meanings of Colors in Simulation:** In order to make simulations more understandable, sensors with different phases, sink sensor and range, truck path and detected truck locations are all represented by different colors.



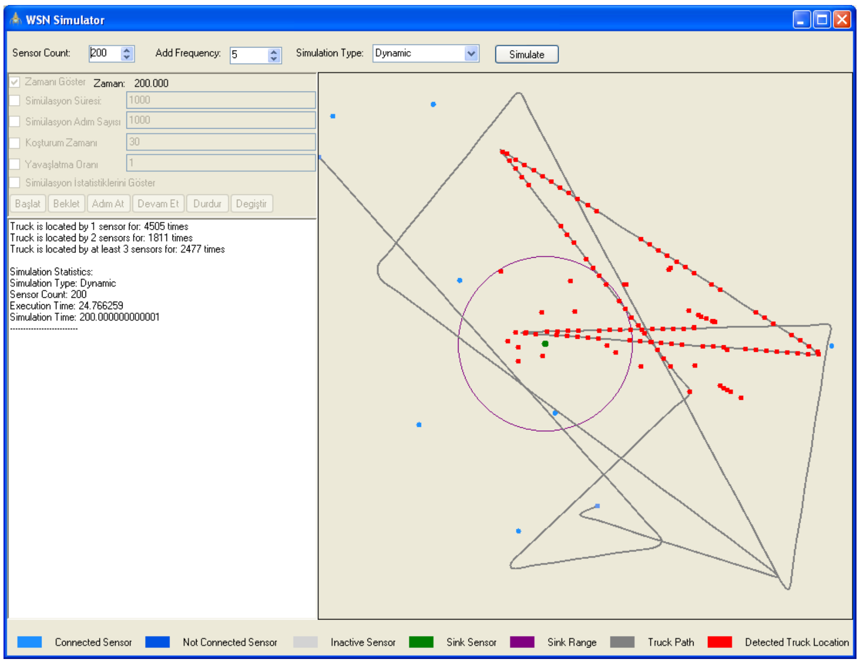
Sensor Count: 200
Add Frequency: 5
Simulation Type: Dynamic

✓ Zamanı Göster Zaman:
Simülasyon Süresi: 1000
Simülasyon Adım Sayısı 1000
Koşturum Zamanı 30
Yavaşlatma Oranı 1
✓ Simülasyon İstatistiklerini Göster

🟦	Connected Sensor
🟦	Not Connected Sensor
⬜	Inactive Sensor
🟩	Sink Sensor
🟪	Sink Range
⬛	Truck Path
🟥	Detected Truck Location

Figure 5.7: WSN Simulator Tool

Figure 5.9 and Figure 5.8 illustrate the simulator after executing static and dynamic simulations respectively. In dynamic approach, after a sensor completes its execution, it is removed from the simulation structure. So, unlike Figure 5.9, in Figure 5.8 no inactive sensors with color gray can be seen. By examining the final state of the simulation panel and the statistics shown in Figure 5.9, it can be observed that the same simulation scenario is executed with both static and dynamic approaches.



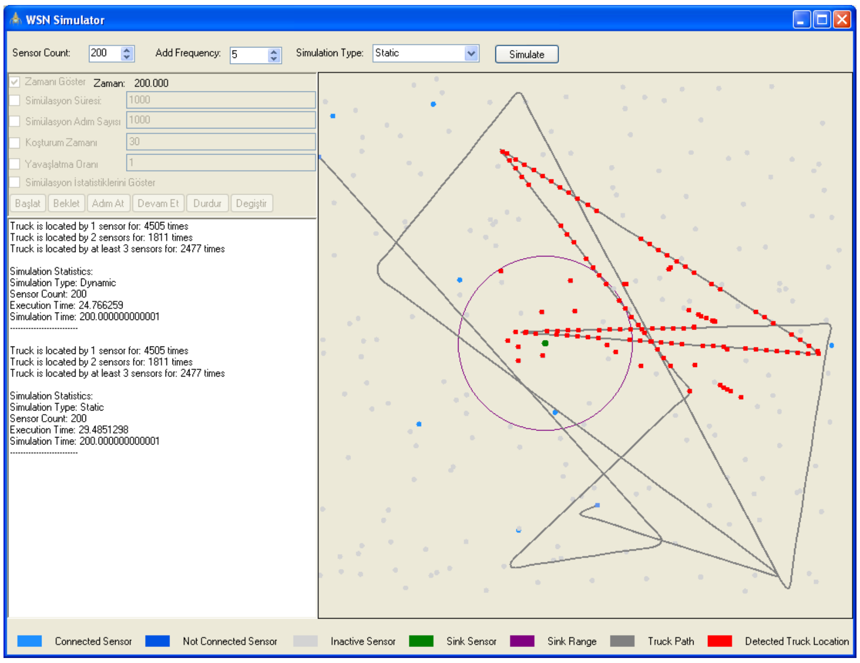
Sensor Count: 200
Add Frequency: 5
Simulation Type: Dynamic
Zamanı Göster Zaman: 200.000
Simülasyon Süresi: 1000
Simülasyon Adım Sayısı 1000
Koşturum Zamanı 30
Yavaşlatma Oranı 1
Simülasyon İstatistiklerini Göster

```
Truck is located by 1 sensor for: 4505 times
Truck is located by 2 sensors for: 1811 times
Truck is located by at least 3 sensors for: 2477 times

Simulation Statistics:
Simulation Type: Dynamic
Sensor Count: 200
Execution Time: 24.766259
Simulation Time: 200.000000000001
------------------------
```

🟦	Connected Sensor
🟦	Not Connected Sensor
⬜	Inactive Sensor
🟩	Sink Sensor
🟪	Sink Range
⬛	Truck Path
🟥	Detected Truck Location

Figure 5.8: WSN Dynamic Simulation



Sensor Count: 200
Add Frequency: 5
Simulation Type: Static
Zamanı Göster Zaman: 200.000
Simülasyon Süresi: 1000
Simülasyon Adım Sayısı 1000
Koşturum Zamanı 30
Yavaşlatma Oranı 1
Simülasyon İstatistiklerini Göster

```
Truck is located by 1 sensor for: 4505 times
Truck is located by 2 sensors for: 1811 times
Truck is located by at least 3 sensors for: 2477 times

Simulation Statistics:
Simulation Type: Dynamic
Sensor Count: 200
Execution Time: 24.766259
Simulation Time: 200.000000000001
------------------------

Truck is located by 1 sensor for: 4505 times
Truck is located by 2 sensors for: 1811 times
Truck is located by at least 3 sensors for: 2477 times

Simulation Statistics:
Simulation Type: Static
Sensor Count: 200
Execution Time: 29.4851298
Simulation Time: 200.000000000001
------------------------
```

🟦	Connected Sensor
🟦	Not Connected Sensor
⬜	Inactive Sensor
🟩	Sink Sensor
🟪	Sink Range
⬛	Truck Path
🟥	Detected Truck Location

Figure 5.9: WSN Static Simulation

5.1.3 Performance Analysis

In classical DEVS formalism, after the initialization phase, simulation goes into a loop where in each cycle, all simulators execute their functions as mandated by the applicable DEVS simulation protocol. The number of simulators for a simulation in classical DEVS is fixed (since the model number is fixed) and does not change while the simulation is running. However, in dynamic version, models can be included into the simulation whenever they are needed and they are removed when they complete their work. Therefore, number of simulators in the dynamic version of a simulation will always be less than or equal to the number of simulators in classical version. As a consequence, the number of calls to simulators in each cycle of the simulation loop is always less than or equal to that of classical approach.

In addition to the above, even though incoming messages are not processed by inactive models, messages are still received by their ports and then they are omitted. Therefore, not only the presence of the inactive models but also unnecessary couplings between those models cause performance degradation during simulation execution. Dynamic DEVS approach eliminates such performance losses through its support for dynamic coupling management.

As it is mentioned in the introduction section, supporting dynamic behavior in simulations is not only required for performance reasons, but also for modeling unexpected behavior and creating more realistic simulations. In our approach, dynamic port management is also supported, but this feature does not have any effect on the performance. This feature can be used for modeling unexpected behavior or creating models that better represent real life entities.

5.1.3.1 Testing Environment

Tests are performed in Windows XP environment on a machine with the following features: Intel(R) Core(TM)2 Quad CPU Q9550 @2.83GHz, 3.93 GB of RAM.

In this case study, execution times of wireless sensor network simulations in different cases are measured. Two metrics are defined for his purpose: 1 - Execution duration of simulations for varying sensor counts and 2 - Execution duraiton of simulation for varying truck step sizes. The rationale behind specifying these metrics and the method adopted for collecting them are given below in sections 5.1.3.2 and 5.1.3.3.

5.1.3.2 Testing Criteria 1 - Sensor Count

Execution times of simulations for varying sensor counts is our first metric. Sensor count is a good measure of structural dynamism in that it indicates the impact of dynamic model inclusion and removal as opposed to a static model coupling structure. As it is discussed in Section 5.1, there are four fixed models in the proposed wireless sensor network system and there are varying numbers of sensors. For example, when there are 40 sensor models in the simulation, number of sensor models will be %90 of whole model structure. In dynamic version, these sensors are included into the simulation when they are activated and removed from the simulation when they are deactivated. However, in classical approach, all models are added to the simulation at the beginning and removed when the simulation is terminated. When a model is needed, its state is changed to active and when it is no longer needed its state is changed back to inactive. In Table 5.1.3.2 execution times of simulations for both static and dynamic approaches and performance improvement of dynamic approach over static approach can be observed. Gain of dynamic approach over static approach is calculated as the increase in velocity of dynamic execution over static one. For example, if dynamic execution completes its work in 3 time units and static execution in 4 time units, this means dynamic approach is %33 faster than the static approach. This also means, dynamic approach improves static approach with %25, but the results shown in Table 5.1.3.2 are calculated according to the first approach, which compares velocities of static and dynamic approaches.

In order to better understand the results shown in Table 5.1.3.2, Figure 5.10 and Figure 5.11 are drawn. In Figure 5.10, execution times of static and dynamic approaches are shown. As it can be observed from the results, execution times of static approach are proportional with the sensor model count, whereas execution times of dynamic approach are almost proportional with the logarithm of the sensor model count. In Figure 5.11, performance gain of dynamic approach over static approach is graphically

Table 5.1: Comparison of the two approaches according to sensor count

Sensor #	Static(s)	Dynamic(s)	Difference(s)	Gain(%)
1	0.30	0.27	0.03	9.60
2	0.47	0.43	0.04	9.74
3	0.72	0.64	0.08	12.18
4	2.04	1.91	0.12	6.53
5	2.55	2.37	0.18	7.46
6	3.65	3.40	0.25	7.35
7	4.50	4.20	0.30	7.17
8	5.68	5.29	0.39	7.36
9	6.79	6.29	0.50	7.93
10	7.87	7.28	0.59	8.07
11	8.97	8.25	0.72	8.76
12	10.62	9.65	0.98	10.13
13	12.24	11.02	1.22	11.11
14	14.26	12.81	1.45	11.31
15	15.78	14.07	1.70	12.11
16	17.26	15.05	2.22	14.72
17	19.02	16.55	2.46	14.89
18	20.70	17.70	3.00	16.95
19	22.37	18.98	3.39	17.83
20	24.16	20.04	4.12	20.56
21	25.89	21.06	4.83	22.92
22	27.99	22.32	5.67	25.38
23	30.14	23.79	6.35	26.69
24	32.44	25.18	7.26	28.83
25	33.95	26.13	7.81	29.90
26	35.82	27.13	8.69	32.03
27	38.20	28.87	9.34	32.34
28	40.53	29.91	10.63	35.53
29	42.42	30.93	11.49	37.14
30	44.51	31.92	12.59	39.43
31	46.61	32.70	13.90	42.52
32	48.04	33.28	14.76	44.36
33	49.69	34.02	15.67	46.07
34	51.37	34.65	16.72	48.26
35	53.48	35.33	18.15	51.37
36	54.70	36.05	18.64	51.71
37	56.39	36.14	20.25	56.01
38	58.01	36.58	21.43	58.59
39	59.22	36.73	22.49	61.24
40	60.77	36.90	23.87	64.71

represented. It can be observed that, performance difference between approaches increases almost linearly with the sensor count increasing.

Another metric that is actually dependent on the sensor count is total number of messages transferred between models. It can be observed that, in static simulations, total number of messages circulating in the simulation is proportional with the square of sensor count. On the other hand, in dynamic simulations, message count does not increases that fast with the sensor count increasing. In Table 5.1.3.2, number of messages for different sensor counts are shown for both static and dynamic simulations. By combining the results shown in Table 5.1.3.2 and Table 5.1.3.2, it can said that, for different sensor counts, static and dynamic simulations create different number of messages and this is the main reason in the performance difference.

Table 5.2: Comparison of the two approaches according to total message count

Sensor #	Message # in Static	Message # in Dynamic	Difference (%)
10	1540071	1484869	3.70
20	5545412	4726308	17.33
30	11003793	7885944	39.53
40	16120844	9250767	74.26

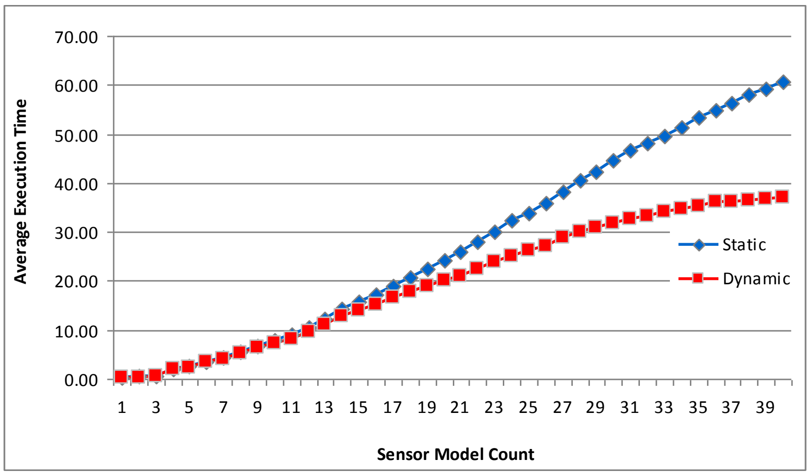


Figure 5.10: Execution times of simulations according to sensor count

5.1.3.3 Testing Criteria 2 - Truck Step Size

The simulation step size of the Truck model is a convenient parameter for increasing or decreasing the number of messages traveling along the couplings between the models, which is instrumental in measuring the effect of dynamic coupling management in

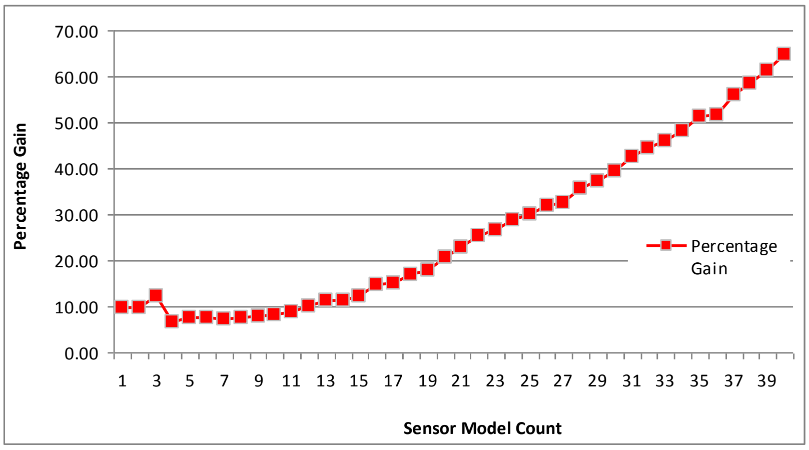


Figure 5.11: Performance advancement of dynamic approach according to sensor count

reducing the message handling cost of the framework. Number of messages circulating between sensors is inversely proportional with the truck step size. So, increasing truck step size will decrease the number of messages and hence amount of data exchanged between the models during the simulation. Tests for measuring this criteria are executed for 10 sensors with varying truck step sizes for both static and dynamic approaches. The impact of truck step size on performances can be seen in Table 5.3.

Table 5.3: Comparison of the two approaches according to truck step size

Sensor #	Truck Step Size(s)	Static(s)	Dynamic(s)	Difference(s)	Gain(%)
10.00	0.001	24.79	17.53	7.26	41.41
10.00	0.002	15.25	11.52	3.73	32.36
10.00	0.003	12.22	9.50	2.72	28.61
10.00	0.004	10.12	7.99	2.13	26.74
10.00	0.005	8.84	7.15	1.69	23.59
10.00	0.006	8.61	7.05	1.57	22.24
10.00	0.007	8.29	6.82	1.46	21.46
10.00	0.008	8.07	6.72	1.34	19.99
10.00	0.009	8.05	6.74	1.31	19.40
10.00	0.010	7.52	6.44	1.07	16.66

Results shown in Table 5.3 are graphically represented in Figure 5.12 and Figure 5.13. According to Figure 5.12, it can be said that performances of both approaches decrease with the step size increasing. This is an expected behavior. When truck step size increases number of messages circulating between models decreases and so performance increases. On the other hand, in Figure 5.13 it can be observed that dynamic approach does not have as much performance loss as static approach. When truck step size decreases, in other words number of messages circulating between sensors increases, the gap between dynamic approach and static approach also increases. So, when the system gets complexer, dynamic approach would be probably a better choice.

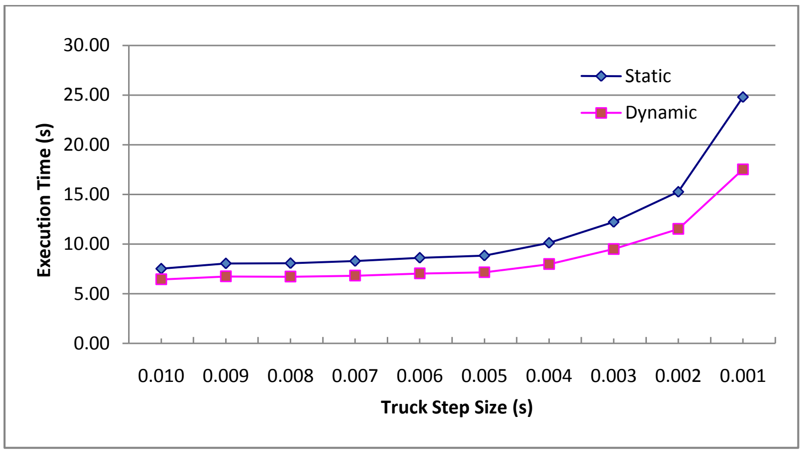


Figure 5.12: Execution times of simulations according to truck step size

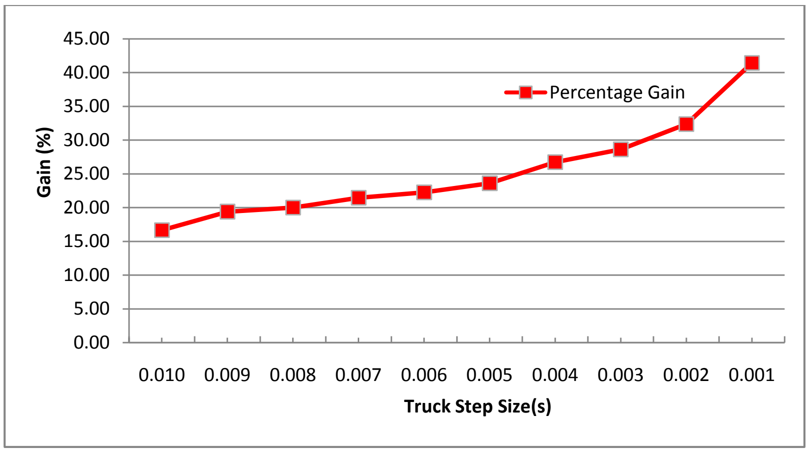


Figure 5.13: Performance advancement of dynamic approach according to truck step size

CHAPTER 6

DISCUSSION AND FUTURE WORK

In this thesis, we have introduced our approach that contributes to the formal representation and implementation of variable structure support to DEVS based modeling and simulation environments.

In order to relate our work to similar work in the literature, we summarized fundamental properties of DEVS formalism and SiMA environment. DEVS is a widely used formalism that is introduced by Bernard Zeigler in 1976 and SiMA is an implementation of an extended form of parallel DEVS formalism, which is developed at TUBITAK UEKAE. In order to formalize required changes for dynamism support we extended DEVS formalism's definition and successfully implemented these theoretical extensions in SiMA framework.

In order to relate our work to similar work in the literature, we have discussed the two most relevant approaches for supporting structure variability in DEVS based modeling and simulation frameworks. These are DSDEVS which is introduced by Fernando Barros in 1995, and DynDEVS which is introduced by Uhrmacher in 2001. To summarize: Our approach to add dynamism to our basic DEVS model is similar to that of DynDEVS as indicated earlier. However, we do not have the operational boundaries that DynDEVS has. Unlike DynDEVS, our approach allows dynamic port management and allows atomic models to make changes other than changing themselves only. It can be said that DynDEVS and our approach are more generalized form of DSDEVS formalism in that if change structure functions are only implemented by one atomic model in each coupled model and only changes related to that coupled model are handled, then our approach reduces to DSDEVS or DynDEVS.

One particular contribution we offered in this thesis is the systematic framework support for post-structural-change state synchronization among models with related couplings. This operation works in the opposite direction of the normal message flow and enables newly added models and newly added couplings to acquire the current state of the simulation. This feature is used and tested in the sample simulations. For example, in the sample WSN simulation, sensors start sensing movement actions in the environment upon receiving activation messages sent from main sensor, indicating that main sensor started execution. Main sensor sends this message only once after the initialization phase and when a model included into the simulation dynamically, it requires to know whether main sensor is executing to start sensing environment. In order to enable newly added sensors to acquire the current state of the simulation state synchronization mechanism is used.

In order to test our approach, we developed a sample Wireless Sensor Network Simulator that uses dynamic SiMA. Since dynamic SiMA is an extended version of classic SiMA, this simulator was able to simulate both static and dynamic simulations. Using this simulator, we executed several scenarios with different parameters and measured the performance according to two different metrics: 1 - the sensor count in the simulation and 2 - the simulation step size of the truck model. As a result, using sensor count metric, we observed in Section 5.1.3 that as the model structure complexity increased, performance difference of dynamic approach over static approach also increased. For this metric, performance of dynamic SiMA show more than %60 percent gain over static version. According to our second metric, we observed that as the truck step size (hence the number of data packets cycling in the simulation) increased, dynamic SiMA improved the overall performance by %40. We argue that these results indicate the utility of dynamism support in improving the performance of the simulations. This performance improvement is mostly gained by variable structure model usage, not because of our theoretical approach or efficient implementation. If we have implemented DSDEVS or DynDEVS approaches we would probably observe similar performance achievements. What we provide with this study is more flexible and robust environment that is also easy to use for dynamic structure management. In our examples, a sensor model, which consists of four atomic submodels and couplings between those models, is included in the simulation when it is required and it is removed from the simulation when its power goes off. In this way, we both get rid of unnecessary simulators and unnecessary message transfers that, according to the statistics, have a huge impact on the performance.

As it is mentioned above, our approach is more generalized form of DSDEVS approach and we can manage structure as it is handled in this approach. For example, in the sample WSN simulations, 'Sensor Adder' model adds new sensors to the simulation and also it does not contain any behavioral logic, other than structural changes. So, this model is similar to the network executive model defined in DSDEVS approach. Unlike DSDEVS, DynDEVS also allows atomic models to make structural changes. However, it only allows atomic models to change themselves. In our approach, we do not have such restrictions. For example, each sensor model contains a battery model that handles sensors power and when power goes off this atomic model creates a change request message that removes its enclosing sensor from the overall structure. A summary of the most important features of these approaches can be observed in Table 6.1.

Table 6.1: Feature comparison of approaches

	DSDEVS	DynDEVS	Our Approach
Adding/Removing Model	Yes	Yes	Yes
Adding/Removing Coupling	Yes	Yes	Yes
Adding/Removing Port	Yes	No	Yes
State Synchronization	No	No	Yes

6.1 Future Work

In order to improve systems performance and in order to provide a more robust environment, we are planning to implement more test simulations. Even though we tried to test all features of the current system over sample simulations, we believe there may still be cases that may cause errors or may have significant effects on the performance. In order to get rid of such problems and improve our system we are planning to prepare and execute more tests.

Also, in this thesis, we implemented our approach by extending classic SiMA that does not support distributed simulations. Our objective for further studies is, implementing our theoretical approach over distributed SiMA and supporting structural changes in a distributed environment.

Our another objective is to support real world applications in the near future, anticipating that there will be room for improvement to the mechanisms we devised. In particular, scenarios where supporting dynamic fidelity-level adjustments of multiple models in a coordinated way is a requirement, are potential use cases where we hope dynamic SiMA would provide a viable solution for application developers.

REFERENCES

[1] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[2] Lassaad Baati, Claudia Frydman, and Norbert Giambiasi. LSIS_DME M&S environment extended by dynamic hierarchical structure DEVS modeling approach. In *SpringSim '07: Proceedings of the 2007 spring simulation multiconference*, pages 227–234, San Diego, CA, USA, 2007. Society for Computer Simulation International.

[3] Fernando J. Barros. Dynamic structure discrete event system specification: a new formalism for dynamic structure modeling and simulation. In *Simulation Conference Proceedings, 1995. Winter*, pages 781–785, Dec 1995.

[4] Fernando J. Barros. The dynamic structure discrete event system specification formalism. *Trans. Soc. Comput. Simul. Int.*, 13(1):35–46, 1996.

[5] Fernando J. Barros. Modeling formalisms for dynamic structure systems. *ACM Trans. Model. Comput. Simul.*, 7(4):501–515, 1997.

[6] Fernando J. Barros. Abstract simulators for the DSDE formalism. In *WSC '98: Proceedings of the 30th conference on Winter simulation*, pages 407–412, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[7] Fernando J. Barros, Bernard P. Zeigler, and Paul A. Fishwick. Multimodels and Dynamic Structure Models: An Integration of DSDE/DEVS and OOPM. In *Winter Simulation Conference*, pages 413–420, 1998.

[8] Cumhur Doruk Bozağaç, Gulsah Karaduman, Ahmet Kara, and Mahmut Nedim Alpdemir. Sim-PETEK : A Parallel Simulation Execution Framework for Grid Environments. In *Proceedings of Summer Computer Simulation Conference (SCSC'09)*, pages 275–282. SCS, 2009.

[9] Fatih Deniz, Ahmet Kara, Mahmut Nedim Alpdemir, and Halit Oğuztüzün. Variable Structure and Dynamism Extensions to SiMA, A DEVS Based Modeling and Simulation Framework. In *Proceedings of Summer Computer Simulation Conference (SCSC'09)*, pages 117–124. SCS, 2009.

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

[11] Jan Himmelspach and Adelinde M. Uhrmacher. Processing Dynamic PDEVS Models. In *MASCOTS '04: Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of*

Computer and Telecommunications Systems, pages 329–336, Washington, DC, USA, 2004. IEEE Computer Society.

[12] Xiaolin Hu, Bernard P. Zeigler, and Saurabh Mittal. Variable Structure in DEVS Component-Based Modeling and Simulation. *Simulation*, 81(2):91–102, 2005.

[13] Ahmet Kara, Doruk Bozagac, and Mahmut Nedim Alpdemir. SIMA: A DEVS Based Hierarchical and Modular Modelling and Simulation Framework. 2. National Defensive Applications Modelling and Simulation Conference, 2007.

[14] Ahmet Kara, Fatih Deniz, Cumhur Doruk Bozağaç, and Mahmut Nedim Alpdemir. Simulation Modeling Architecture (SiMA), A DEVS Based Modeling and Simulation Framework. In *Proceedings of Summer Computer Simulation Conference (SCSC'09)*, pages 315–321. SCS, 2009.

[15] B. Karp and H.T. Kung. Greedy Perimeter Stateless Routing for Wireless Networks. pages 243–254. Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking, 2000.

[16] K. Lee, K. Choi, J. Kim, and G. C. Vansteenkiste. A Methodology for Variable Structure System Specification: Formalism, Framework, and Its Application to ATM-Based Network System. *ETRI JOURNAL*, 18(4):245–264, 1997.

[17] Tuncer I. Ören. Dynamic templates and semantic rules for simulation advisors and certifiers. pages 53–76, 1991.

[18] T. Pawletta and S. Pawletta. A DEVS-based simulation approach for structure variable hybrid systems using high accuracy integration methods. In *Proceedings of the Conference on Conceptual Modeling and Simulation, Part of Mediterranean Modelling Multiconference*, pages 368–373, Genova, Italy, 10 2004.

[19] Hairong Qi, S. Sitharama Iyengar, and Krishnendu Chakrabarty. Distributed sensor networks–a review of recent research. *Journal of the Franklin Institute*, 338(6):655 – 668, 2001.

[20] Hui Shang and Gabriel A. Wainer. A flexible dynamic structure DEVS algorithm towards real-time systems. In *SCSC: Proceedings of the 2007 summer computer simulation conference*, pages 339–345, San Diego, CA, USA, 2007. Society for Computer Simulation International.

[21] Wikipedia. `http://en.wikipedia.org/Trilateration`. Last access date is 12.10.2009.

[22] A. M. Uhrmacher. Dynamic structures in modeling and simulation: a reflective approach. *ACM Trans. Model. Comput. Simul.*, 11(2):206–232, 2001.

[23] Bernard Zeigler and Herbert Praehofer. *Theory of Modeling and Simulation.* Academic Press, January 2000.

[24] Bernard P. Zeigler. *Theory of Modeling and Simulation.* John Wiley, 1976.

[25] Bernard P. Zeigler, Tag Gon Kim, and Chilgee Lee. Variable structure modelling methodology: an adaptive computer architecture example. *Trans. Soc. Comput. Simul. Int.*, 7(4):291–318, 1990.