Distributed Discrete Event Simulation Architecture with Connectors

İsmet Özgür Çolpankan[1,2], Ahmet Kara[2] and Halit Oğuztüzün[1]
[1]*Department of Computer Engineering, Middle East Technical University, Ankara, Turkey*
[2]*TÜBİTAK BİLGEM İLTAREN, Ankara, Turkey*

Keywords: DEVS, SiMA, Distributed DEVS, Connectors.

Abstract: Distributed Discrete Event System Specification (DEVS) environments are developed with various computing, networking and implementation language options. We propose a distributed approach to the Simulation Modeling Architecture (SiMA), a DEVS-based modeling and simulation framework, with software connectors. We employ Windows Communication Foundation (WCF) as the middleware technology. A connector is a first class entity which performs interaction among components, thus, plays an important role in a component-based architecture. We claim that using a connector instead of modifying an already developed model increases the model reusability. We also compare this approach with the existing distributed DEVS approaches in terms of base formalism, network layer technology, model partitioning, remote node synchronization scheme and message exchange pattern.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Complex model hierarchy, high level of detail in models and large simulations cause the processor and memory of a computer to become insufficient to run a simulation in a reasonable time. Therefore, the need to use diverse resources dispersed over a network for a scalable performance leads to development of parallel and distributed simulation systems. Distributed DEVS idea was launched in 1985 by Zeigler (Zeigler, 1985) and until today several distributed DEVS applications have been developed, including: DEVS/CLUSTER (Kim and Kang, 2004), DEVS/RMI (Zhang et al., 2005), DEVS/P2P (Cheon et al., 2004), and DEVS/SOA (Mittal et al., 2009).

In this paper, we propose a distributed approach to enable Simulation Modeling Architecture (SiMA) to execute in a distributed environment. SiMA is a DEVS-based modeling and simulation framework developed in TÜBİTAK BİLGEM İLTAREN. It implements the SiMA-DEVS formalism which is an extended version of Parallel DEVS formalism. Our approach is using Windows Communication Foundation (WCF) (Cheng, 2010) as an underlying middleware technology. WCF offers a set of APIs in the .NET framework (Millas, 2013) for establishing service-oriented applications (Cheng, 2010). Core simulation engine of SiMA was developed in .NET framework, hence WCF might be attuned to SiMA easily. Furthermore, in a WCF-to-WCF application the fastest message encoding formatting and transfer protocol methods can be utilized considering the other facilities that WCF offers.

Our approach is also integrating the concept of software connectors for adaptation of distributed nodes and models. The increase in modeling complexity leads to utilization of already developed reusable models. Furthermore, model reuse saves developers development effort and time, and more importantly regression tests to verify and validate the modified model. However, it is not always feasible to use a legacy model in a new simulation scenario in terms of detail of computation and data types for communication among models. Connectors engage at this point by providing interactions among components by transferring control or data, and playing the gluing role in component-based architectures. Besides, DEVS formalism is highly appropriate for a component-based framework design when each model is considered as a component. Therefore, in our implementation we have adopted connectors as introduced by Kara (Kara et al., 2014) for Distributed DEVS environments to perform data conversions and data marshalling/unmarshalling.

Novelty of our approach is the explicit use of connectors for adaptation of distributed nodes and models via favorable WCF features in a distributed DEVS environment.

Çolpankan I., Kara A. and Oguztüzün H..
Distributed Discrete Event Simulation Architecture with Connectors.
DOI: 10.5220/0005531801300137
In *Proceedings of the 5th International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (SIMULTECH-2015), pages 130-137
ISBN: 978-989-758-120-5
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

The rest of the paper is organized as follows: Section 2 provides an overview of the background of our research, Section 3 provides distributed DEVS approaches related to our research, Section 4 explains our approach in detail, Section 5 presents a case study using our implementation and our discussions about the importance of our approach, and finally Section 6 includes our resultant comments.

2 BACKGROUND

2.1 SiMA

SiMA (Kara et al., 2009) is a modeling and simulation framework that is built on DEVS formalism (Zeigler, 1976). For complex model construction it uses a formalism which extends the Parallel DEVS formalism (Chow and Zeigler, 1994). SiMA has two extensions to the parallel DEVS formalism; strongly-typed inter model connection environment and direct feed through transition function. In port definitions of models there are constraints that makes port types type-safe. Moreover, the new transition function provides that in the same simulation time a model can receive data, make computation on it, and send the modified data without any state change. It has its own port type, direct feed through port, in order to process incoming events and send them in the same simulation time. To avoid deadlocks, an application independent loop-breaking logic is defined in this port type.

2.2 Connectors

Connectors are the architectural building blocks that manage the interactions among components (Amirat et al., 2009). Some examples of interactions are procedure calls, method invocations, data flow, communication protocol, and pipelines. Mehta and his colleagues (Mehta et al., 2000) proposes eight connector types; *procedure call*, *event*, *data access*, *linkage*, *stream*, *arbitrator*, *adaptor*, and *distributor*. In our study, we take the classification of Mehta into account.

3 RELATED WORK

This section summarizes of some Distributed DEVS approaches.

DEVS/P2P (Cheon et al., 2004) is a distributed DEVS implementation that proposes a peer-to-peer (P2P) simulation protocol to operate a DEVS simulation on a distributed and parallel computing environment. The proposed protocol uses advantage of P2P infrastructure, in which inter-connected peers share resources with each other without using any centralized administrative system, to gain optimal performance compared to existing protocols. As middleware it uses JXTA (Wilson, 2002) technology which is a P2P network system implementation.

DEVS/GRID (Seo et al., 2004) uses Grid computing infrastructure for DEVS modeling and simulation activities. DEVS/GRID proposes new functionalities to the existing DEVS M&S frameworks as mentioned by Seo (Seo et al., 2004): "cost-based hierarchical model partitioning, dynamic coupling restructuring, automatic model deployment, remote simulator activation, self-communication setup, M&S name and directory service, etc." As middleware, Globus Toolkit which is widely used in grid computing is used.

DEVS/CLUSTER (Kim and Kang, 2004) transforms hierarchical DEVS model structure into a non-hierarchical one to ease the synchronization of remote models. DEVS/CLUSTER utilizes CORBA as a communication system which is designed to perform collaboration between heterogeneous platforms, different programming languages and operating systems.

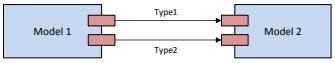
DEVS/RMI (Zhang et al., 2005) focuses mostly on the reconfiguration of the simulation structure dynamically in runtime unlike other approaches. Its underlying communication technology is Java RMI.

DEVS/SOA (Mittal et al., 2009) provides a solution to cross-platform distributed M&S in a client-server architecture using SOA. It uses Java for the implementation. Messages between remote nodes are serialized with SOAP.

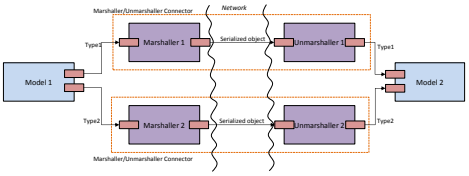
DEVS/PyRo (Syriani et al., 2011) is designed to make quantitative analysis of reliability and performance of different simulator designs with detection of failures in computational and network resources. It is implemented in Python. PyRO which is an RMI-based python implementation is used in middleware layer.

4 DISTRIBUTED SiMA

Distributed SiMA is a framework that enables SiMA to execute in a distributed environment. It also aims to increase model reusability by bringing software connector notion to distributed DEVS environment. Connectors in Distributed SiMA assist inter node communication and adaptation of models in terms of port data types. Distinctively from other distributed



(a) Located in the same node



(b) Located in remote nodes

Figure 1: Marshaller/Unmarshaller Connectors

DEVS approaches Windows Communication Foundation is adopted as an underlying middleware technology. Like SiMA core engine Distributed SiMA has been implemented in .NET *C#*. Besides, WCF is convenient to adapt two .NET products with each other in terms of performance. In Distributed SiMA many WCF features are benefited such as service oriented development environment for simplicity, request/reply message exchange pattern for synchronization, binary encoding over TCP transportation for performance and customized service behavior for object serialization.

4.1 Connectors in Distributed SiMA

Distributed SiMA uses connectors which play important roles in simulation execution. Connectors in Distributed SiMA are the specialized atomic models. There are two kinds of connectors in terms of the tasks they carry out: marshaller/unmarshaller connector and data conversion connector.

4.1.1 Marshaller/Unmarshaller Connector

It consists of two built-in atomic models used in communication among remote models. If there is a port coupling between two models (coupled or atomic) located at different nodes, a marshaller/unmarshaller connector is placed between the models in order to serialize and deserialize the port data. The marshaller part of the connector is placed in the source node, and the source model's output port is connected to its input port. Similarly the unmarshaller part of the connector is placed in the target node, and its output port is connected to the target model's input port. A port coupling of two models in Figure 1a can be transformed for a distributed environment as in Figure 1b with marshaller/unmarshaller connectors.

Conceptually, there is a one connector that performs marshalling and unmarshalling, in addition to networking chores. In terms of implementation, it is composed of two atomic DEVS models (one for marshalling, the other for unmarshalling), and the underlying network. Marshalling/Unmarshalling connector uses direct feed through transition port. Therefore there is no simulation time step loss between remote models despite the connectors between them. *Model 1* sends data to *Marshaller 1*. In the same simulation time step *Marshaller 1* serializes the data and sends it through network to *Unmarshaller 1*. *Unmarshaller 1* deserializes the data and sends it to the *Model 2*. With the help of WCF there is no need for an extra synchronization while sending port data through network.

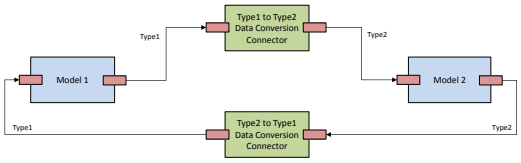


Figure 2: Data Conversion Connectors.

4.1.2 Data Conversion Connectors

Model continuity is a profitable characteristic when huge simulation projects are considered. Already implemented atomic models can be used in other projects without modifications. Modelers might not want to break the integrity of already developed models since the verification and validation time of the modified model may take more time than developing a new model. However, data types processed by the legacy models might not match the data types of the new models. Evidently, composable models may reflect the same real world entities and facts; however, they might have non-identical data representations. These models have to communicate with each other but port data types they are using are not the same. Data conversion connectors (Kara et al., 2014) accomplish this bridging task. They receive a data packet in *Type1*, apply conversion to it, and send it in *Type2* like in Figure 2. They are implemented by the model developers as atomic models. Distributed SiMA provides a base class for the developers to develop connector models that utilize DFT ports for conversion routines.

4.1.3 Connector Roles According to Mehta Classification

Mehta and his colleagues stated eight connector types mentioned in Section 2.2. A connector may have more than one type among them.

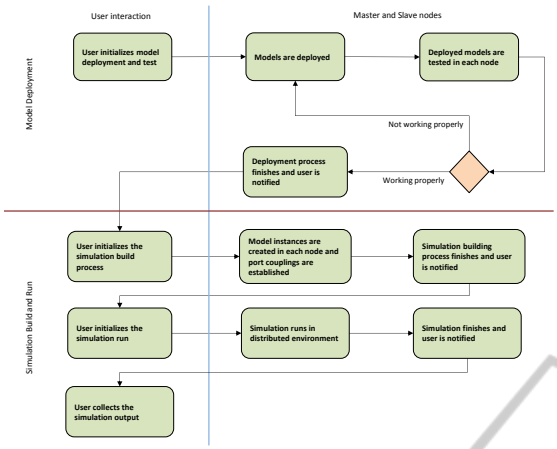


Figure 3: General System Flow Diagram.

Marshaller/Unmarshaller connectors are *procedure call* connectors since they make remote procedure calls. They are also *linkage* connectors as they are not defined as a simulation entity before simulation construction, but they are inserted to simulation when a remote model port data transportation is needed. They establish the WCF communication protocol and serializes data to be sent over network, so they are *adaptor* connectors.

Data conversion connectors are mostly *adaptor* connectors, because they provide data conversion service among components. They also form binding between different typed models and this makes them *linkage* connectors.

4.2 Distributed SiMA Architecture

We used the existing SiMA implementation as a basis to develop Distributed SiMA. We added new packages, extended the core classes, and modified some classes for the distributed version. There is a Master Node managing the whole system by deploying models, building simulation, and executing simulation. Other remote nodes joining the simulation execution are called Slave Nodes.

Distributed SiMA could be examined in two main phases with respect to distributed simulation execution: *model deployment*, and *simulation build and run* as illustrated in Figure 3.

4.2.1 Model Deployment

This phase covers identifying slave nodes, establishing connections, deploying and testing of models in them. Before all of these, slave nodes must be ready for the model deployment. For the model deployment slave nodes need starting the Model Deployment Server and creating a service listener for the connection. We use Windows services that operate in the background, and are managed by the operating system. We developed a Windows service to instantiate the Deployment Service.

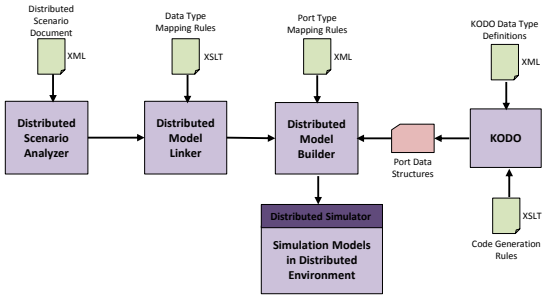


Figure 4: Distributed SiMA Simulation Construction Pipeline.

Slave nodes behave like servers after these steps and start to listen via Deployment Service port for the connection requests. This service is used in the deployment phase of Distributed SiMA. User initializes the model deployment process with the distributed scenario definition document which consists of model definition, coupling information, initialization parameters of models, and endpoint addresses of slave nodes added to coupled or atomic model definitions.

4.2.2 Simulation Build and Run

Model Deployment guarantees that all simulation models are dispatched over the network and tested on the corresponding slave nodes. In this phase distributed simulation will be built and run. Distributed SiMA employs basic SiMA simulation construction pipeline as a base and extends it to build distributed simulation. Again master node maintains the system building process. The distributed version of the SiMA simulation construction pipeline is in Figure 4.

Distributed Scenario Analyzer starts the distributed simulation construction pipeline. In Distributed SiMA, Scenario Document is modified as Distributed Scenario Document that includes the definition of the partition plan. User writes Deployment Service endpoint address of a slave node inside the definition of the model (atomic or coupled) he wants to work as remote. This modification on scenario document is enough to make a basic SiMA scenario distributed. Distributed Scenario Analyzer takes a Distributed Scenario Document file as an input and creates an intermediary data for Distributed Model Linker. The Distributed Scenario Document consists of model definition, model coupling information with remote node endpoint addresses, and initial-

ization data of the models.

Distributed Model Linker is the core component among the others taking role in the pipeline. It prepares the distributed simulation structure to be built. The intermediary scenario received from Distributed Scenario Analyzer is separated into two files in this component. One of the files is the distributed model link map file including the coupling information, model definitions, and port connection definitions using a hierarchical style. The other file is the distributed simulation configuration file consisting of initialization data of the atomic models. User specifies the required data conversion connectors; however, the document does not include marshaller/unmarshaller connector definitions. Distributed Model Linker inserts them where they are required.

Distributed Model Builder is the constructor of the Distributed SiMA. It manages the creation of instances of model classes. It establishes one root coupled model by combining the instances according to hierarchical structure for each node. This hierarchical structure is obtained from the distributed model link map file produced by Distributed Model Linker. For the models to be run in master node, it builds them locally. And for the remote models it commands the slave nodes to build the remote coupled models via Remote Simulation Service. There are also remote model proxies located in the master node for each remote coupled model. Their task is to provide a communication setup in order to communicate with Remote Simulation Service. They are images of the remote coupled models in master node. Distributed Simulator manages the distributed simulation on master node as if it is local. It thinks that all models are located in one computer. Actually remote model proxies create this illusion.

Figure 5 illustrates the configuration of an example system. There is a *Master Node* and two *slave nodes*. *Slave Node 2* has two sessions to manage different remote models. Remote model proxies in *Master Node* connect Remote Simulation Services (RSSs) in slave nodes. Marshaller modules of Marshallar/Unmarshaller Connector (MCs) connect the related Unmarshaller modules of Marshallar/Unmarshaller Connector (UMCs) through Port Services (PSs).

KODO is not adjusted to distributed SiMA since the port and initial data classes generated are used as they are. Actually in WCF user specifies the port data classes transported through network by tagging them with *Data Contract* property. The .NET framework detects that the class will be serialized by WCF. However, we did not develop a new distributed version

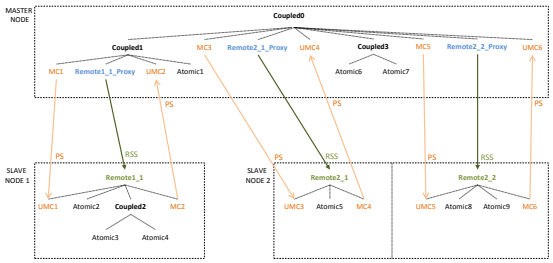


Figure 5: Distributed Simulation Structure Example Overview.

of KODO or modify the generated classes. Because it brings some work load and also ruins the easiness of making an already developed SiMA scenario distributed. One would have to run the new distributed KODO to add *Data Contract* property to all generated classes. We resolved this problem by changing the Port Service data serialization behavior. In default WCF expects to serialize only Data Contracts. With the changing service serialization behavior, all port data classes can be serialized into the *object* class which is the base class of all classes in .NET. This feature of WCF serialization behavior can be used because of WCF-to-WCF communication. There is not a cross-platform communication and we can use the benefits of WCF-to-WCF communication in both data transport optimization and serialization.

Distributed Simulator maintains the centralized view of simulation execution as in SiMA. However, it is distributed since there are remote model proxies in place of the actual models. At the management level Parallel DEVS protocol is applied. Simulation execution is synchronized in terms of simulation time management as a default because of its central architecture and request/reply feature in remote procedure calls of WCF. There is no need for an extra global or local simulation time synchronization among remote nodes.

5 CASE STUDY

This section includes a case study to demonstrate Distributed SiMA. The scenario used in the case study is a wireless ad hoc sensor network (Yick et al., 2008). There are two kinds of sensor models in the scenario: *detailed sensor model* and *regular sensor model*. There is also a *sink model* gathering sensor information. A *logger model* is used to trace the local and distributed models' activities. A *platform model* is used as a target to be detected by sensors and make them send detection information to the sink model. Moreover, there are connectors; Regu-

larToDetailedSensorInfo and DetailedToRegularSensorInfo connector between regular sensor models and detailed sensor models, and DetailedToRegularPlatformInfo connector between platform and regular sensor models. All scenario models and relations between them are observed from Figure 6. As the aim of this case study is not a WSN evaluation, the implementation of models does not reflect the exact calculations needed to be done in a real wireless ad hoc sensor network.

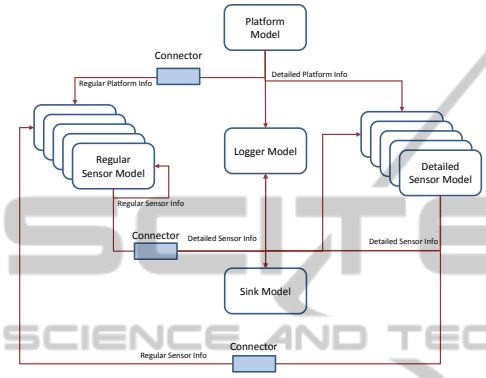


Figure 6: Models in the Case Study Scenario.

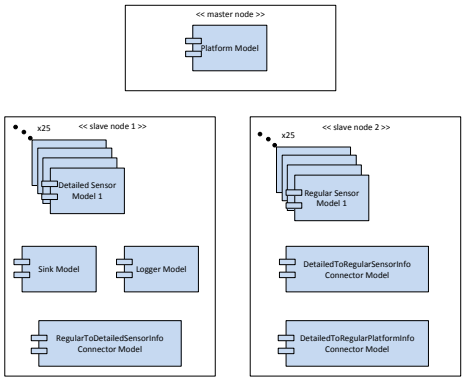


Figure 7: Partition Plan Overview of Case Study Scenario.

Distributed SiMA was tested with 2 slave nodes. Partition plan of the models is shown in Figure 7. Slave Node 1 has 25 Detailed Sensor Models, one Sink Model, one Logger Model and one RegularToDetailedSensorInfo Connector Model. Slave Node 2 has 25 Regular Sensor Models, one DetailedToRegularSensorInfo Connector Model and one DetailedToRegularPlatformInfo Connector Model. Additionally, Master Node runs one Platform Model.

5.1 Discussion

To handle component interactions the use of connectors is the most flexible approach (Allen, 1997).

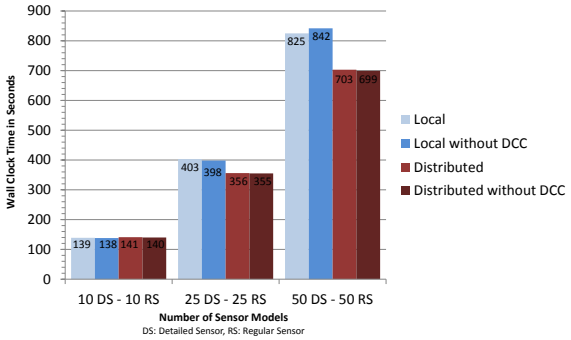


Figure 8: Case Study Scenario Test Results.

Our data conversion connectors are the adaptors that tie two or more atomic model components designed to interoperate. However, it can be considered that adding new models to a scenario brings burden to the simulator and slows down the simulation execution. Moreover, one has to develop the new data conversion connector model.

In our example, there are two already developed and tested scenarios. The first one uses Detailed Sensor Models and the second one uses Regular Sensor Models. In a new scenario we want to use both sensor models. There are two ways in order to prepare a scenario like this: modifying one type of sensor model to convert the received data when it is making calculations and again convert the calculated data back to send, or developing data conversion connectors. We implemented the case study scenario without data conversion connectors. We removed all the data conversion connectors and modified the Regular Sensor Model to adapt the new environment. Input and output port data types changed to detailed ones. Required conversions are done before calculations and before sending the calculated data. By doing this we achieved two results:

1. We executed the simulation with and without data conversion connectors and the simulation execution time did not change. Thus, a connector is not a burden to the simulator.
2. The Regular Sensor Model had 182 lines at first. After the modification 18 lines were changed and 26 new lines are added. As a ratio *10%* of code lines is modified and *14%* of code lines is added. Thus, when atomic models consist of massive data and calculations are considered, modifying them for the adaptation to the new scenario models costs developer more than anticipated.

5.2 Evaluation

We have conducted some tests on the case study sce-

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

nario and obtained the data shown in Figure 8. The scenario is executed both with data conversion connectors as *DCC* and without them. The X axis of the chart shows the number of Detailed Sensor Model as *DS* and Regular Sensor Model as *RS* in the scenario. The Y axis of the chart shows the wall clock time of simulation execution in seconds. The result data shows that when number of sensor models in the scenario increases, Distributed SiMA executes faster than SiMA. Moreover, in distributed simulation execution, the effect of data conversion connectors is unremarkable with only 0.5% increase in execution time. There is an unanticipated result: in the scenario with 50DS-50RS SiMA executed, simulation is executed faster with data conversion connectors. Because with data conversion connectors 200 port connections are established between Detailed Sensor Models and Regular Sensor Models. However, when data conversion connectors are removed 5000 port connections are established and this slows down the execution.

6 CONCLUSION

We have proposed a distributed approach for SiMA via WCF. WCF takes charge in communication and transportation among distributed nodes. Moreover, our approach use connectors which provide adaptation of distributed nodes and models and increase model reuse. Compared to other distributed DEVS implementations Distributed SiMA brings novelty in these three points:

1. Distributed SiMA uses SiMA-DEVS formalism
2. In order to increase model reusability, and distributed node and model adaptation, connectors are used
3. WCF is adopted in the network communication and data transportation layer

In Section 3 we have described various distributed DEVS approaches. And the Table 1 is summing up those information with the Distributed SiMA approach.

A data conversion connector increases model reusability by placing between a new and a legacy model. This also hinders developers from applying regression tests for the validation and verification. If a data conversion connector was not placed, the legacy model would have to be modified to be compatible with the new scenario models. Benefits of connectors in the distributed DEVS setting can be listed as:

1. Modularity and object-oriented design approach usage which is supported strongly by the DEVS (Zeigler, 1976) increases when connectors are used (Kara et al., 2014).
2. Development time is decreased since developing a new atomic model is faster than modifying an already developed atomic model. The existing atomic model would be implemented by another person, so it may be hard to understand the code and change it. Also the model may have a lot of lines of code and again this increases the changing time.
3. When we change an existing atomic model, we disrupt the code integrity. It was a verified atomic model when it was used in previous scenario. Therefore, the modified atomic model needs to be verified, and unit and regression tests have to be done. Connectors save developer from these issues.
4. Since we do not change the existing model and use it as it is, model reusability is promoted.

Modification to the code requires the related validation and verification work to be redone. Moreover, it is easier to develop a data conversion connector in a shorter time. On the other hand, marshaller/unmarshaller connector facilitates the port data serialization/deserialization operations along the network. For example, if a marshaller part of the connector was not placed at the source node, connection with the corresponding unmarshaller part of the connector would have to be made and the data serialization operation would have to be done in source model.

In Distributed SiMA most of the WCF features are utilized. These features and utilization ways are given as:

1. *Service Oriented Development Environment.* This provides a simple development environment. It presents a service interface in a server and a client makes remote procedure calls after connecting the service. In service interface since data type to be sent and received is specified, no extra data type transport definition is needed.
2. *Request/Reply Message Exchange Pattern.* When a client makes a remote procedure call, it waits for a reply from the server. With the help of this no extra synchronization mechanism needs to be implemented. It ensures that the distributed simulation execution is always in sync.
3. *Binary Encoding over TCP Transportation.* This feature is used since all distributed nodes run a WCF application. The main benefit of it is the fastest way of communication among WCF-to-WCF applications.
4. *Extensibility.* In order to make explicit use of binary encoding possible, we customized the service behavior. Thus, a .NET serialized object can be transported through network.

Table 1: Comparing Different Distributed DEVS Approaches.

	Formalism	Middleware Technology	Partitioning	Synchronization Scheme	Message Exchange
DEVS/CLUSTER	DEVS	CORBA	hierarchical to non-hierarchical structure	optimistic	CORBA remote method invocations
DEVS/GRID	DEVS	Globus	cost-based hierarchical partitioning	conservative	GIIS
DEVS/P2P	DEVS	JXTA	autonomous hierarchical model partitioning	conservative	JXTA message format
DEVS/RMI	DEVS	JAVA/RMI	applying built-in partition algorithm	conservative	JAVA serializable object
DEVS/PyRO	DEVS	PyRO/RMI	user specified or automatic	conservative	serialized objects
DEVS/SOA	Parallel DEVS	GIG/SOA	user specified	conservative	JAVA serialization
Distributed SiMA	**SiMA**	**WCF**	**user specified**	**conservative**	**.NET binary serialized objects**

REFERENCES

Allen, R. (1997). *A Formal Approach to Software Architecture*. PhD thesis, Carnegie Mellon, School of Computer Science. Issued as CMU Technical Report CMU-CS-97-144.

Amirat, A., Oussalah, M., et al. (2009). Reusable Connectors in Component-Based Software Architecture. In *Proceedings of the ninth international symposium on programming and systems,(ISPS 2009)*, pages 28–35.

Cheng, S. (2010). *Microsoft Windows Communication Foundation 4.0 Cookbook for Developing SOA Applications*. Packt Publishing Ltd.

Cheon, S., Seo, C., Park, S., and Zeigler, B. P. (2004). Design and implementation of distributed DEVS simulation in a peer to peer network system. *Advanced Simulation Technologies Conference–Design, Analysis, and Simulation of Distributed Systems Symposium. Arlington, USA*.

Chow, A. C. H. and Zeigler, B. P. (1994). Parallel DEVS: A Parallel, Hierarchical, Modular, Modeling Formalism. In *Proceedings of the 26th Conference on Winter Simulation*, WSC '94, pages 716–722, San Diego, CA, USA. Society for Computer Simulation International.

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

Kara, A., Oguztüzün, H., and Alpdemir, M. N. (2014). Heterogeneous DEVS Simulations with Connectors and Reo Based Compositions. In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative*, DEVS '14, pages 1:1–1:6, San Diego, CA, USA. Society for Computer Simulation International.

Kim, K.-H. and Kang, W.-S. (2004). CORBA-based, multithreaded distributed simulation of hierarchical DEVS models: transforming model structure into a non-hierarchical one. In *Computational Science and Its Applications–ICCSA 2004*, pages 167–176. Springer.

Mehta, N. R., Medvidovic, N., and Phadke, S. (2000). Towards a taxonomy of software connectors. In *Proceedings of the 22Nd International Conference on Software Engineering*, ICSE '00, pages 178–187, New York, NY, USA. ACM.

Millas, J. L. L. (2013). *Microsoft .Net Framework 4.5 Quickstart Cookbook*. Packt Publishing Ltd.

Mittal, S., Risco-Martín, J. L., and Zeigler, B. P. (2009). DEVS/SOA: A Cross-Platform Framework for Net-centric Modeling and Simulation in DEVS Unified Process. *Simulation*, 85(7):419–450.

Seo, C., Park, S., Kim, B., Cheon, S., and Zeigler, B. P. (2004). Implementation of distributed high-performance DEVS simulation framework in the Grid computing environment.

Syriani, E., Vangheluwe, H., and Al Mallah, A. (2011). Modelling and Simulation-based Design of a Distributed DEVS Simulator. In *Proceedings of the Winter Simulation Conference*, WSC '11, pages 3007–3021. Winter Simulation Conference.

Wilson, B. J. (2002). *JXTA*. Pearson Education.

Yick, J., Mukherjee, B., and Ghosal, D. (2008). Wireless sensor network survey. *Comput. Netw.*, 52(12):2292–2330.

Zeigler, B. P. (1976). *Theory of Modelling and Simulation*. A Wiley-Interscience Publication. John Wiley.

Zeigler, B. P. (1985). Discrete Event Formalism For Model Based Distributed Simulation. In *SCS Conf. Distributed Simulation*, pages 3–7.

Zhang, M., Zeigler, B. P., and Hammonds, P. (2005). DEVS/RMI-An Auto-Adaptive and Reconfigurable Distributed Simulation Environment for Engineering Studies. *ITEA Journal*.