Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

DE^{2}M: A SOLUTION FOR ANALYZING SUPPLY CHAIN

María de los Milagros Gutiérrez

CIDISI. Dep. Sistemas
Universidad Tecnológica Nacional. Facultad Regional Santa Fe
Santa Fe 3000 ARGENTINA.

Horacio P. Leone

INGAR . CIDISI
Conicet. Universidad Tecnológica Nacional. Facultad Regional Santa Fe
Santa Fe 3000 ARGENTINA.

ABSTRACT

Nowadays, the incremental use of component-based simulation presents a new challenge to overcome. So, the researchers and software developers are putting attention to solve problems as interoperability and reuse of components. In the supply chain context, this simulation paradigm is very valuable because allows us to develop independent models and then interconnect them using middleware software such as HLA, CORBA among other. However, the interoperability of these simulation components is a mayor problem to overcome. In this paper, we present an environment to analyze supply chain using this paradigm. We propose to use the SCOR model as roles that the component can play in conjunction with an Ontology base on SCOR. The modules are developed using DEVS formalism and HLA as middleware.

1 INTRODUCTION

A recurring problem in supply chain simulation is the need to reuse piece of simulation software existing in the supply chain' members. In this way, the stand-along simulators need to be integrated to enable joint execution. The main goal is to integrate the models from different members into one integrated model. This is known as "distributed simulation" (Verbraeck 2004). In this context, model behavior is exposed to other simulation software via network simulation protocol. In literature, supply chain simulation is usually treated in two different ways: (i) using a single model pointed out the SC (Liu et al. 2004; Byrne et.al. 2004; Von Mevius et.al. 2004; Biswas et.al. 2004; Cope et.al. 2007), representing each member as a node and the relations among them as arcs and running the model a single machine. (ii) using several models running in a distributed environment (Jian et.al 2004). It is necessary to collect information from members in order to develop the simulation model with the first approach. In this case, the existing simulation models are useless. Nevertheless, this approach has not interoperability problems. We believe that the second perspective is better in the supply chain context. However, there are a few number of studies describing supply chain simulation in a distributed environment. One of the reasons is the difficulty in provide interoperability among simulations models, especially when they are developed using CSP (COTS simulation package).

In recent years, the Simulation Interoperability Standards Organization's (SISO) and the CSP Interoperability Product Development Group (CSPI PDG) have been working in the development and standardization of a set of Interoperability Reference Models (IRM) (Taylor et al. 2007). This IRM provides a frame of reference that allows for interoperability capacity among CSP. This fact will help in the incremental use of distributed simulation in the area of supply chain.

This work presents an environment that allows for supply chain distributing simulation. We see a supply chain (SC) as a virtual enterprise: " a temporary network of independent companies linked by information technologies to share skills, costs and access to respective markets" (Villa 2002). Based on it, our proposal raises the development of an enterprise model (EM) instead of a supply chain model. In this sense, each member is responsible for developing its EM. Then, each SC member joins its EM in a Federation representing the SC.

Then, the EM can run in a local environment or in a distributed one. The execution in a local environment allows us to analyze and improve the innermost processes before analyzing the relation with other members in a distributed environment. In this work, the distributed simulation uses HLA as middleware. Particularly, we used poRTIco (portico 2007) an implementation of RTI with free software license. A simulation model is developed once and it can be used in both environment because we have used a components-based simulation where the models are separated from simulation engine.

In this new approach, the use of EM has a significant impact. On one hand, the EM represents the internal processes based on global information but in a way that is hidden from SC external processes. On the other hand, the role that an EM plays, manifest the role in the collaboration process at the same time it sets the interaction with other simulations.

This work is organized as follows. Section 2 depicts the architecture, its objectives and components. Section 3 describes the mechanism to obtain the federate representing the EM in a distributed environment. Section 4 presents an application example. Finally, Section 5 discusses conclusions and future work.

2 DE^{2}M ARCHITECTURE

Figure 1 shows the architecture of the environment called DE^{2}M - Distributed and Executable Enterprise Model. In our proposal, we have decided to choose a two-layer architecture, in order to hide the simulation process details. The top layer - Conceptual Model Layer – characterizes the knowledge about an organization in term of processes, tasks, resources, and objectives. The bottom layer - Simulation Layer - represents the behavior of the organization in term of events, ports, process, queue, state transition, and simulation time.

The first layer is the one the user works with. It implements the Document-View design pattern (Buschmann, Meunier, Rohnert, Sommerlad and Stal 1996). The functions that are offered, are related to the development of the EM using a business-oriented language, in this particular case Coordinates language (Mannarino et.al. 1999). Then, the user can develop the model in an easy and friendly way using an already known vocabulary (like task, resources, processes, states) and a graphical interface. We obtain as a result an EM called conceptual model, since it points out the conceptualization of the organization. This model has no simulation information. The simulation layer provides functionality to create the SM, to execute it either locally or in a distributed environment, and to compute metrics. This layer implements the MSVC design pattern (Nutaro et.al. 2004). It is responsible for translating the conceptual model in a simulation model without user intervention. So, this environment is oriented to business expert more than simulation expert.

As regards Simulation layer, the Enterprise Simulation Model component is based on the DEVS formalism (Zeigler, Praehofer and Kim 2000). Its classes extend from the DEVSJAVA 3.1 framework (DEVSJAVA 2004) and define the building-blocks used in the development of simulation model, such as AtomicTaskDevs, TaskVersionDevs, ResourceDevs, among others. The Coordinator component is the engine of the simulation model. It covers the root coordinators. There are two root-coordinator: one in a local and other in a distributed environment (Coordinator and CoordinatorE2M classes respectively). The Simulator component covers the engine associated with each building-block participating in SM. Then, each atomic model has a simulator associated and each coupled model has a coordinator associated. Finally the View component is made up of entities that appear in the user interface showing the concepts (task, resources, states), graphics and results. In order to update the view, we have used the observer design pattern (Gamma, Helm, Johnson and Vlissides 1996).

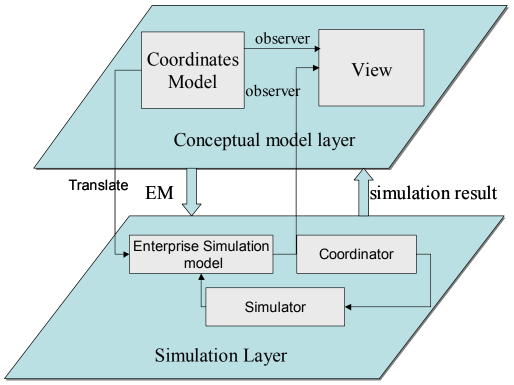


Figure 1: DE^{2}M architecture

Figure 2 shows the methodology used in this proposal in order to generate a distributed and executable enterprise model. First of all, the user develops the conceptual model. It is made up of task models, resource models and resource life cycle models representing the different views of the EM.

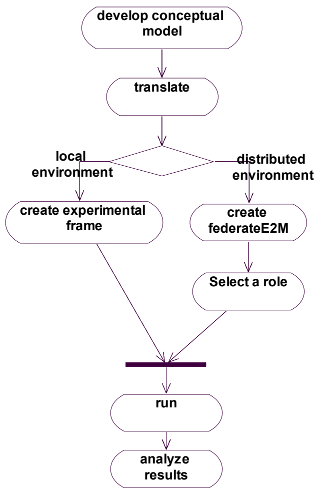
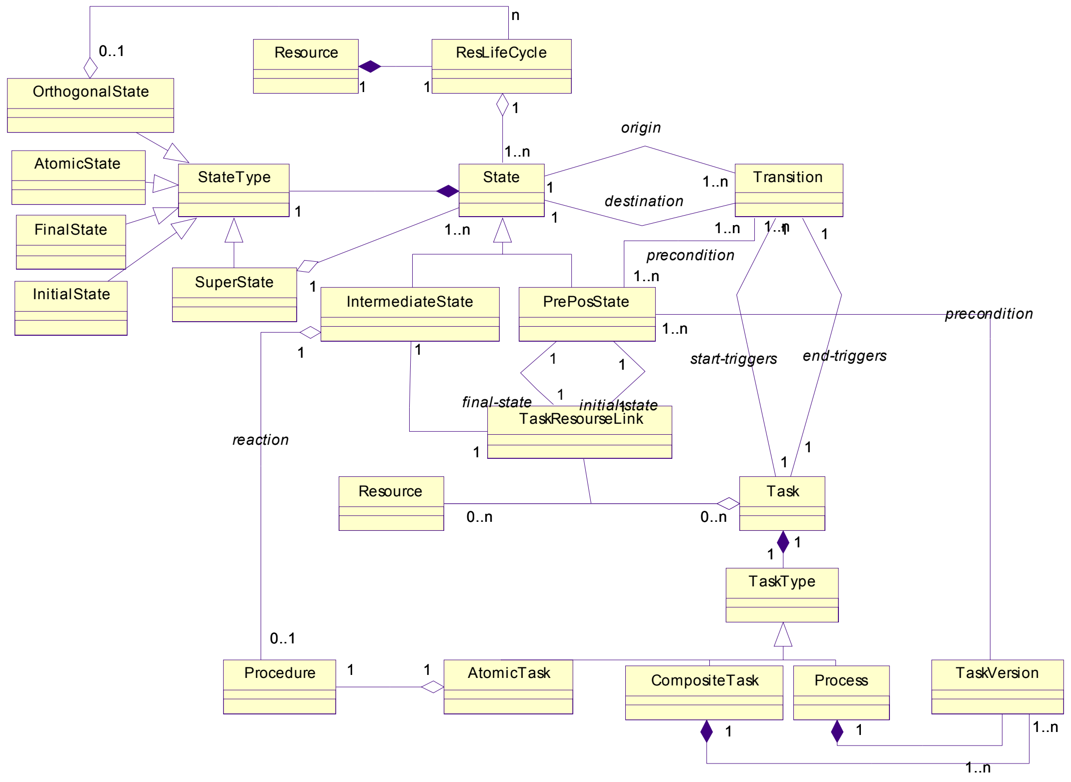


Figure 2: Methodology to develop a DE^{2}M

Secondly, the simulation model is developed automatically without user intervention. The conceptual model is translated in a SM. All the information needed to develop SM can be found in the conceptual model. In this sense, each component in the conceptual model has a DEVS model associated. Then the SM is developed coupling these models. Thirdly, the user has to choose execute the SM in a local or distributed environment, according to her/his objectives: to analyze the inner most process of the enterprise in a partial way (the lower level models) or to analyze the relationship with external processes. In the first case, the user sets the simulation objectives and the simulation time. From this information, the environment develops the experimental frame. In the second case, the environment develops a federate, which has the SM as simulation code. If the execution is done in distributed environment, it is needed select a role playing by the enterprise in the supply chain. These roles have been defined according to SCOR model (SCOR 2006). Based on the selected role, the federate knows which interaction sends and receives. Finally, the SM is executed and the results are shown through the View. The user can modify the simulation objectives in order to change the experiment or modify the conceptual model to analyze a different model.

2.1 Conceptual Model Layer

This layer has two components: Coordinates Model and View. Coordinates Model has been developed based on Coordinate language, a modeling enterprise language, and defines the building-blocks to develop an EM. Figure 3 shows the class diagram of this component. The main building-blocks to generate the conceptual model are task, resource, process, task version and links. There are two type of links: (i) temporal relationships, that related tasks between them and represent the sequence in which the tasks must be executed: meet, before, during, ending, are one of this type; and (ii) task-resources relationships, which relates tasks with resources. These links represent how the task uses the resource: use, modify, create, and eliminate, among others. The Coordinates language supports task description at multiple levels of detail. Task is a recursive composed concept. A Process must be the root in a task decomposition hierarchy. A Composite Task, by contrast, may occupy any position in a decomposition hierarchy. Both, the Process and the Composite task are described through one or more TaskVersion, each one encapsulating a particular way of accomplishing the task. TaskVersion, may differ, for instance, in the specific combination of Resources they use, the subtasks they are decomposed into, the conditions under which they may be carried out, the view they adopt of the organization, etc.



.

Figure 3: Coordinates model class diagram

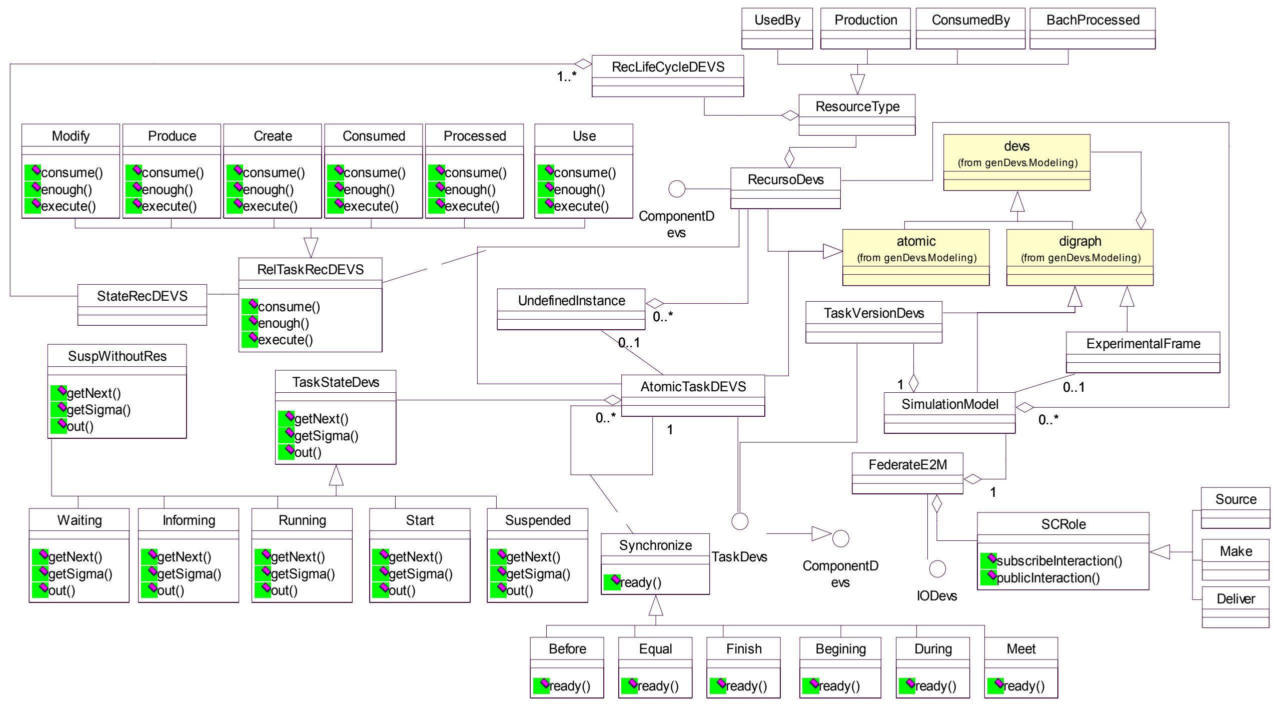


Figure 4: Simulation model component class diagram

A TaskVersion is described in terms of a particular set of Tasks, Resources, TemporalLinks and TaskResourceLinks. Each entity in the Coordinates Model component has a view element associated that is part of the View component. We have used the observer design pattern (Gamma, Helm, Johnson and Vlissides 1996) in order to show the changes in the enterprise model. This layer has been presenting in (Gutierrez et.al. 2001).

2.2 Simulation Layer

The Simulation layer has services to translate the conceptual model into a simulation model and to execute it in a local machine or in a distributed environment. This layer is based on DEVS formalism. There are two main reasons to choose this formalism. The first one is the hierarchical construction style of the DEVS models. And the second is the fact that the simulation models are decoupled from the simulation engine. These characteristics allow us to generate the SM only once, and use it in both, local and distributed environment with different associated engines.

The Simulation layer has three components: Enterprise Simulation Model, Simulator and Coordinator.

2.2.1 Enterprise Simulation Model Component

This component defines the building-blocks used in the development of simulation model. Figure 4 shows the Simulation model class diagram. The classes TaskAtomicDevs, TaskVersionDevs, ResourceDevs are DEVS model representing the behavior of the entities in the conceptual model. It has been using the State design pattern in order to represent the states which a task can be found. The resources were classifying according to the relation with task. In this way, a resource could be used, consumed, produced by tasks. Each type of resource has a life cycle associated characterizing its behavior. For instance, a resource that is used by tasks, will have a type UseBy attached and, the life cycle associated will have two states: idle and busy. The class Translator is responsible for translating the conceptual model in a SM. It uses special modeling and simulation knowledge in order to take the correct decisions when it is developing the SM. This knowledge allows for detecting inconsistencies in the model and informing about them. As result, Translator gets an instance of SimulationModel representing the enterprise behavior.

A SimulationModel depicts the highest level process. It consists of a TaskVersionDevs representing the tasks decomposition and ResourceDevs instances taking the place of resources found in the conceptual process. In a local environment, this model has an ExperimentalFrame associated that feeds the SM. In a distributed execution, an instance of FederateE2M is generated. It characterizes an HLA-compliant federate whose simulation code is the simulation model and can interact with other federates under HLA. In this way, this federate has a Federate Ambassador associated that implements the call-back function setting by HLA specification. With the aim of provide a semantic integration among federates, indicating how to interpret the messages interchanged we have used an ontology called SCOntology (Gonnet, Vegetti, Leone, Henning 2006). Based on it, the FOM (Federation Object Model) has defined. It establishes a consensual and precise meaning of the information interchanged among participants of a SC. In addition to the use of the ontology, we have defined roles that a federate can play into a supply chain (SC). The role classes are defined based on SCOR processes: make, sources and deliver. The Source role contains interaction that procure goods and/or services. The Make role include interactions that transform goods into higher-valued products. Finally, the deliver role includes interactions that provide finished or intermediate products and services. All roles include interactions to represent the return process. These classes are responsible for publishing and subscribing interactions. A FederateE2M has a SCRole associated representing the role that the federate plays in the supply chain. That is to say, when a federate selects a role, it sets the interaction that the federate can publish and subscribe according to the FOM. Then, a federate can play different roles in supply chain context, and it can change the interaction that publishes and subscribes dynamically changing its associated role.

2.2.2 Simulator Component

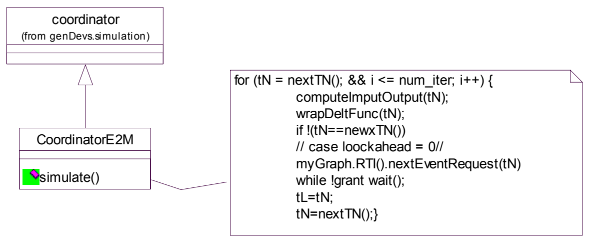
This component involves the engines needed to interpret and run the SM. Due to the fact that a SM is a DEVS model, and its components are DEVS models too, we used the simulators and coordinator defined in the genDevs.simulation package (DEVSJAVA 2004) to interpret the SM. In this context, the atomicSimulator is assigned to stand alone atomic models. The coordinator is assigned to coupled models when they are not part of another model. The coupledSimulator and coupledCoordinator are assigned to atomic models and coupled models respectively when they are part of a high level coupled model.

2.2.3 Coordinator Component

This component is responsible for controlling the simulation execution. This architecture allows for defining two different executions: local and distributed. In this way, there are two root-coordinators directing the simulation in each particular environments: coordinator, provided by DEVSJAVA framework, and CoordinatorE2M respectively.

Figure 5 shows the classes of this component. In the local environment, the coordinator is in charge of directing simulation run. But in the distributed environment, the CoordinatorE2M does it.

CoordinatorE2M overwrites the simulate method in order to solve problems with time management.



```
for (tN = nextTN(); && i <= num_iter; i++) {
            computeImputOutput(tN);
            wrapDeltFunc(tN);
            if !(tN==newxTN())
            // case loockahead = 0//
            myGraph.RTI().nextEventRequest(tN)
            while !grant wait();
            tL=tN;
            tN=nextTN();}
```

Figure 5: Control component class diagram

There are two generic approaches to time management: conservative and optimistic (Fujimoto 2003). Both schemes are seen like advancing the time forward, while they process events with temporary marks. In the first case, causality violation is strictly avoided (that is to say, the cause always appears before the effect); whereas the second approach admits the violation temporarily, but when it is detected, is rectified using roll back. A third scheme is the parallel DEVS protocol, which can be seen as an extremely optimistic risks free scheme, where roll back is no implemented in the components (Zeigler, B. G. Ball, H. Cho and H. Sarjoughian 1998). When a component detects having receive a message in its past, it informs a bad synchronization, but it does not do anything to correct it. The root coordinator is in charge of synchronization.

On the other hand, HLA allows federates select the time management scheme. When a federate adheres to a federation, it has to establish how the federate will receive and send events from/to other federates. So, it is responsible for managing the internal simulation time and its synchronization with others federates. Table 1 shows a comparison between Parallel DEVS protocol and HLA standard to distributed simulation.

These differences must be consider at the time of generating the HLA federate. Since DEVS does not implement rollback, and since time management is centralized in the root coordinator, the way to communicate and to interact with the RTI is using a conservative scheme. That is to say, whenever DEVS simulation needs to advance time, it will have to ask the RTI for grant. Using this scheme, the RTI and the federate can guarantee no send an event in the past. The coordinatorE2M, extends coordinator (a class belong to DEVSJAVA framework) and overwrites the simulate method in order to modify the DEVS simulation cycle.

The federate FederateE2M is conservative, therefore when it joins to a federation, it defines itself as time Constrained and time Regulated. So, during the normal cycle of simulation, the coordinatorE2M calculates the time of the next event and, before advancing the next step, it asks for time advance grant invoking the nextEventRequest(tn) service.

Table 1: Comparing Parallel DEVS and HLA

Characteristic	DEVS	HLA
Time management	Optimistic, without roll-back	Optimistic and conservative
Communication	messages	interaction and attribute value updated
Time advance and coordination	Root coordinator	RTI
Message sent	Regulated by Coordinator.	Schema publish/subscribed

Figure 6 shows a collaboration diagram depicting the interaction among models, engines and controllers in a distributed environment in case of time management.

The aRootCoordinator is an instance of CoordinatorE2M and it is responsible for directing the simulation run in the distributed environment. The aFederate (an instance of FederateE2M) is responsible for interacting with other federates and it interacts with RTIAmbassador and FederateAmbassador in order to achieve its objective. The SimulationModel is the enterprise behavior and has a coupledCoordinator associated.

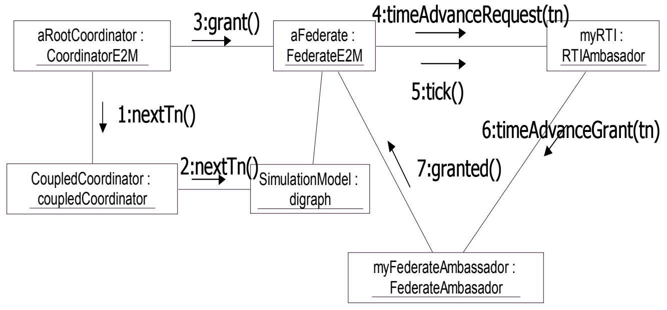

aRootCoordinator : CoordinatorE2M
3:grant()
aFederate : FederateE2M
4:timeAdvanceRequest(tn)
myRTI : RTIAmbasador
5:tick()
1:nextTn()
6:timeAdvanceGrant(tn)
7:granted()
CoupledCoordinator : coupledCoordinator
2:nextTn()
SimulationModel : digraph
myFederateAmbassador : FederateAmbasador

Figure 6: Time management in distributed environment

ARootCoordinator asks for next event time to the coupled coordinator associated, this message is propagated in the simulation engine hierarchy. When the aRootCoordinator has the Tn value, before advancing the simulation time, it asks for grant to the aFederate. After this, aFederate invokes the services timeAdvanceRequest and waits for grant. After a few seconds, myRTI sends the grant invoking the call back function timeAdvanceGrant. Then, myFederateAmbassador informs about grant to aFederate, which in turn, returns the grant to the aRootCoordinator. Finally, the root coordinator follows the simulation cycle.

3 FEDERATE STRUCTURE

In this section we explain in more detail the distributed simulation. In this context, we have created a special model call FederateE2M that represents the HLA-compliant federate. It can run under HLA and interact with RTI. In this architecture we have used poRTIco, a free-use RTI implementation.

According to (IEEE 2000) a federate is “an application that may be or is currently coupled with other software applications under a Federation Object Model Document Data (FDD) and a Run Time Infrastructure (RTI)”. Figure 7 shows the FederateE2M structure according to federate definition. The federate has a Federated Ambassador, a simulation code (SimulationModel in the picture), and an RTI-Ambassador.

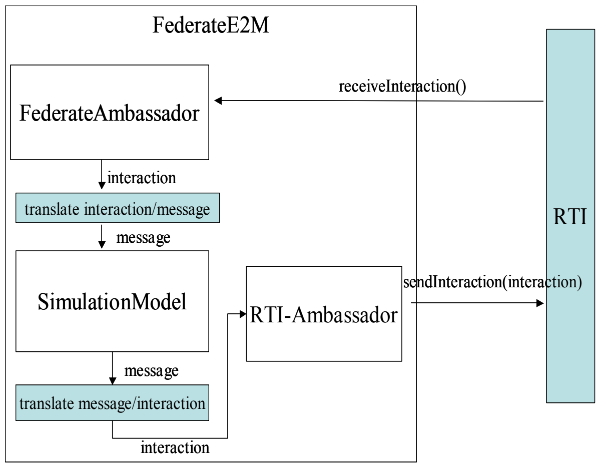
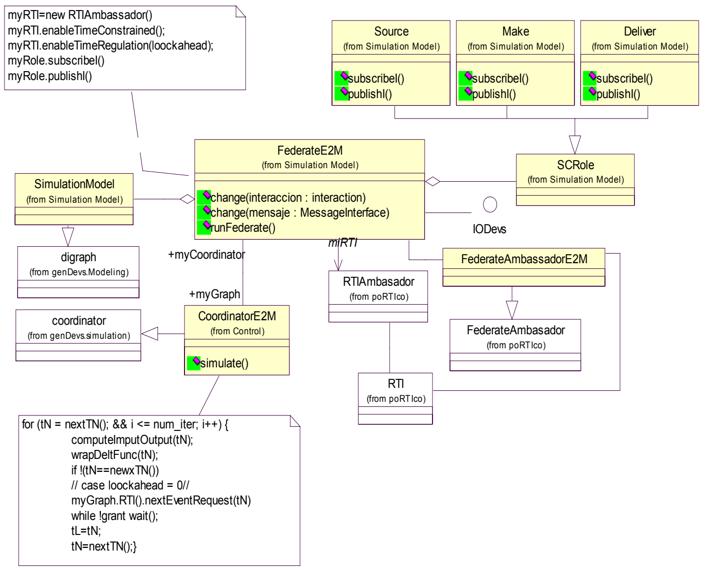


Figure 7: FederateE2M structure

The Federate Ambassador should implement in a particular way, the call back functions which are set by HLA interface specification. This Federate Ambassador must interact with the simulation code to inform the interactions received, the time advance grant, the attribute value update and the synchronization points. The framework poRTIco provides the RTI-Ambassador and the RTI codes. The simulation code is an instance of SimulationModel (the same as in a local environment). The functions “translate interaction/message” and “translate message/interaction” transform the HLA-interactions into DEVS-messages and vice versa. When a federate receives an interaction from other federates, it must be changed into a message that can be understood by the federate simulation code. In the same way, when a federate sends interactions towards other federates. The messages from SimulationModel must be translated into interaction, with in term will be sent to RTI to publication.

From this general abstraction, the class diagram shown in figure 8 defines the class FederateE2M. It is compound of a SimulationModel and implements the IODevs interface. That it to say, this class is a DEVS model and a federate too. Then, in a distributed environment FederateE2M, is the highest-level DEVS model. This DEVS model can run under HLA because it has a special coordinator associated. The coordinatorE2M, extends from coordinator (a class belong to DEVSJAVA framework) and overwrite the simulate method in order to incorporate modifications in the DEVS simulation cycle. The federate FederateE2M is conservative, therefore when it joins to a federation, it defines itself as time Constrained and time Regulated. So, during the normal cycle of simulation, the coordinatorE2M calculates the time of the next event and, before advancing the next step, asks for time advance grant invoking the nextEventRequest(tn) service. The associated SCRole allows federate to subscribe and to publish interactions according to the role playing in SC. In this way, when the federate is running, it delegates in its role the responsibility to subscribe and publish interactions. Then the role will invoke the method subscribeInteraction for each interaction that the federate is interested to receive and publishInteraction for each interaction that the federate will send to other federates.



```
myRTI=new RTIAmbassador()
myRTI.enableTimeConstrained();
myRTI.enableTimeRegulation(loockahead);
myRole.subscribeI()
myRole.publishI()
```

```
for (tN = nextTN(); && i <= num_iter; i++) {
        computeImputOutput(tN);
        wrapDeltFunc(tN);
        if !(tN==newxTN())
        // case loockahead = 0//
        myGraph.RTI().nextEventRequest(tN)
        while !grant wait();
        tL=tN;
        tN=nextTN();}
```

Figure 8: FederateE2M class diagram

4 EXAMPLE

Let’s consider a SC made up of a factory, a wholesaler and a retailer. Each participant is responsible for developing its EM and selecting a role. We analyze the Factory. This enterprise, using the environment develops its EM. As an example figure 9 shows the snapshot of the environment where the user has developed a diagram as part of the task model. The diagram represents a production process. The EM obtained can be analyzed and the results can contribute in processes improvement. Then, a distributed simulation can be executed. In this way, a role is selected. The factory plays a role Make, and the FederateE2M is developed. The role represents the collaboration that each member is commitment to achieve and set the interaction among simulations.

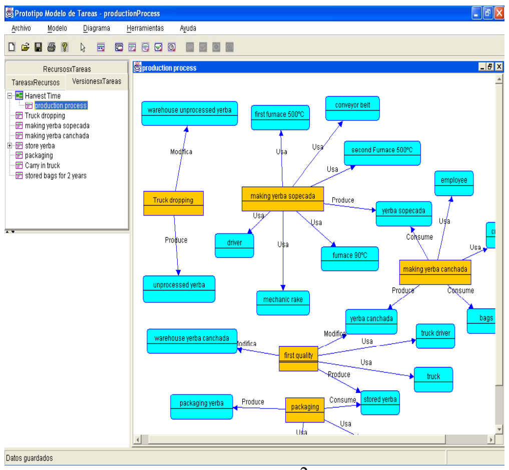


Figure 9: Snapshot of DE^{2}M task diagram.

We consider the following scenario: the RTI is running in a machine, a federation has been creating and federates have been joining to federation. In this example the goal of the collaboration is to satisfied the customer order in less than 7 day, from the order entry until the goods is deliver to the customer.

In this scenario, our federate is ready to join the federation. Then, when it receives the run command, it will try to join a federation. Figure 10 shows the federates joining a federations.

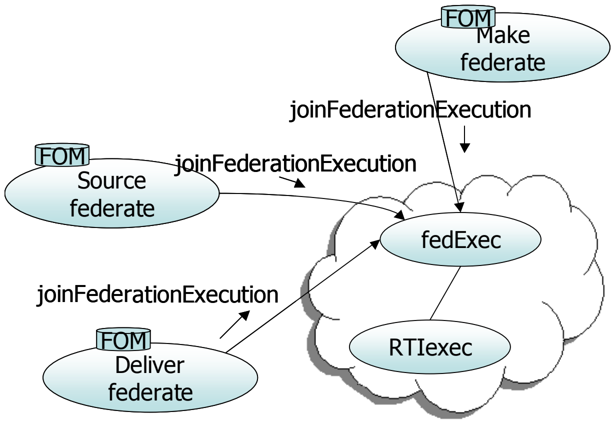


Figure 10: A Federation execution.

Once the federate joins the federation, as part of the initialization process, the federate role executes the subscriptions and publications of interaction. The result in figure 11 shows that the 73% of the orders were satisfied in time and in full, 18% suffered delay and 9% were not deliver at all. Base on this result each member can analyze in deep its internal process as related the inventory level and order fulfill in order to improve it.

In other situation, the factory can play another role in different SC. Then, the EM can be reused. When the federate will be created, it will have another role associated. The mechanism to subscribe and publish are solve by role selected.

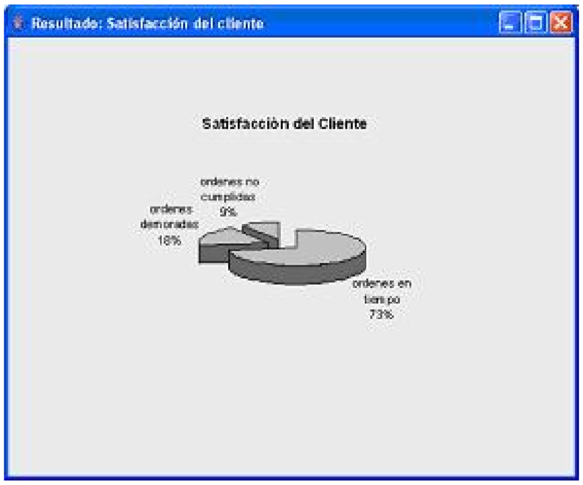


Figure 11: Supply chain simulation result.

5 CONCLUSION

This work presents an integrated environment that allows for modeling and simulation enterprise models. The importance of this tool in supply chains simulation is emphasizing. In this way, the SC was considered as a virtual enterprise and its EM is made of members' EM.

This environment gives support: (i) to define EM using a business-oriented language and (ii) to analyze it through simulation. In this way, it helps the entrepreneur to introduce their knowledge in an easy and friendly way and then to analyze the dynamic behavior in both local and distributed environment.

The Different views, allow user to attack the problem of complexity in the development of EM, where each one depicts different aspects of the Company, without losing sight of the model that these views integrated are together.

The simulation model is built from the integration of existing knowledge in different views, this help us to:

- use the results of analysis in decision making, as it takes into account all aspects of the Company
- have an integrated view of the generated model.
- maintain the consistence among the different views.

In order to achieve this integrated environment, we used DEVS formalism as a simulation tool. This formalism provides a well-defined architecture to specify models of a wide range of modeling paradigms. Since the generation of executable model is almost automatic, the user does not need to be a simulation expert in order to use this tool.

The Simulation layer has been designing with the MVSC design pattern. It allows for reusing the EM in distributed environment.

Since the simulation model is developed using DEVS, in order to run under HLA specification, we have defined the structure of a particular federate that uses DEVS in the simulation code. Then, it has shown how DEVS simulation cycle changes to interact with RTI. In this way, the different companies that have their simulation models can connect among themselves through HLA federation. As a result we get the supply chain simulation model in a distributed environment. So, the existing simulators were reused. An special FOM has been defining using an ontology with the purpose of establishing a consensual and precise meaning of the information interchanged among SC participants. The defined roles allow federates do not worry about its interaction with other federates. The role can be changed, changing the interaction that the federate publishes and subscribes without modify its code.

On the other hand, this proposal is not less important when it is used in a networked company. Thus, the construction of an executable enterprise model in a modular way has important advantage. From the point of view of the networked company, each federate represents an independent part of the company. In this way, the use of the environment helps to reduce the cost and time that is needed to develop DE^{2}M, and simultaneously it facilitates the EM modification and test.

ACKNOWLEDGMENTS

The authors want to acknowledge the "programa de becas de doctorado para docentes de la UTN" that makes this project possible. This work has also been supported by CONICET, UNL, UTN and ANPCyT (PICT 12628).

REFERENCES

Biswas, S., and Y. Narahari. 2004. Object Oriented Modeling and Decision Support for Supply Chains. *European Journal of Operational Research*. 153: 704-726. Elsevier.

Buschmann, F., R Meunier, H. Rohnert, P. Sommerlad and M. Stal. 1996. Pattern-Oriented Software Architecture, A System of Patterns. England: Wiley.

Byrne, P., and C Heavey. 2004. Simulation, A Framework for Analysing SME Supply Chain. In *Proceedings of the 2004 Winter Simulation Conference,* ed R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, 1167-1175. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Cope, D., M. Fayes, M. Mollaghasemi, and A. Kaylani. 2007. Supply Chain Simulation Modeling Made Easy: an Innovative Approach. In *Proceedings of the 2007 Conference on Winter Simulation*, ed. S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 1887-1896. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

DEVSJAVA 3.1 2004 URL: http://www.acims.arizona.edu/SOFTWARE/software.shtml#DEVSJAVA

Fujimoto, R. M. 2003. Distributed Simulation Systems. In *Proceeding of the 2003 Winter Simulation Conference,* ed. S. Chick, P. J. Sanchez, D. Ferrin, and D. J. Morrice, 124-134. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1994. *Design Patterns. Elements of Reusable Object-Oriented Software*, USA, Addison-Wesley.

Gonnet, S. M. Vegetti, H. Leone, and G. Henning. 2006 SCOntology: A Formal Approach Toward a Unified and Integrated View of the Supply Chain Adaptive Technologies and Business Integration. Chapter VII.

Gutierrez, M. G. Mannarino, and H. Leone. 2001. Coordinates Workbench: an Object-Oriented Architecture of a Tool for Conceptualizing an Organization. *International Conference of the Chilean Computer Science Society, IEEE Computer Society*, Los Alamitos, USA.

IEEE 1516-2000, March 2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules.

Jian, J., H. Zhang, B. Guo, K. Wang, and D. Chen. 2004. HLA-Based Collaborative Simulation Platform for Complex Product Design. In *Proceedings of the 8th International Conference on Computer Supported Cooperative Work in Design*. IEEE 2: 563-566.

Liu, J., W. Wang, Y. Chai, Y. Liu. 2004. EASY-SC: A Supply Chain Simulation Tool. *Proceedings of the 2004 Winter Simulation Conference*, ed. R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, 1373-1378. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Mannarino, G., G. Henning, and H. Leone. 1999. Coordinates: A Framework for Enterprise Modeling. *Information Infrastructure Systems for Manufacturing. IFIP-Kluwer Academic Publishers*.

Nutaro, J., P. Hammonds. 2004. Combining the Model/View/Control Design Pattern with the DEVS Formalism to Achieve Rigor and Reusability in Distributed Simulation. *The Society for Modeling and Simulation International*. (1).

poRTIco version 0.7 October 2007 URL: http://www.porticoproject.org [access October 25, 2007].

SCOR, 2006 version 8.0 Overview. URL: http://www.supply-chain.org [access February 02, 2008].

Taylor, S., N. Mustafee, S. Turner, M. Low, S. Strassburger, and J. Ladbrook. 2007. The SISO CSPI PDG Standard for Commercial Off-The-Shelf Simulation Package Interoperability Reference Models. In *Proceedings of the 2007 Winter Simulation Conference*, ed. S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 594-602. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Verbraeck, A. 2004. Component-Based Distributed Simulations. The Way Forward? In *Proceedings of the Eighteenth Workshop on Parallel and Distributed Simulation*. Kufstein, Austria. 141-148

Villa, A. 2002. Emerging Trends in Large-Scale Supply Chain Management. In *ICPR 16th International Conference on Production Research*. 40: 3487-3498.

Von Mevius, M., and R. Pibernik, 2004. Process Management in Supply Chain – A New Petri Net Based Approach. In *Proceedings of the 37th Hawaii International Conference on System Sciences*.

Zeigler, B., G. Ball, H. Cho, and H. Sarjoughian. September 1998, The DEVS/HLA Distributed Simulation Environment and Its Support for Predictive Filtering. Advance Simulation Technology Thrust (ASTT) DARPA Contract N6133997K-007.

Zeigler, B., H. Praehofer and T. G. Kim. 2000, *Theory of Modelling and Simulation. Integrating Discrete Event and Continuous Complex Dynamic System*. Second Edition. California: Academic Press.