Using Distributed Simulation for Analyzing Enterprise Models

Ma. De los Milagros Gutierrez [1], Horacio P. Leone [1,2]

[1] Universidad Tecnológica Nacional. Facultad Regional Santa Fe. CIDISI, Lavaise 610 3000 Santa Fe, Argentina
[2] Instituto de Desarrollo y Diseño INGAR. Conicet
mmgutier@frsf.utn.edu.ar hleone@santafe-conicet.gov.ar

Abstract. Nowadays the incremental use of distributed simulation presents new challenges to overcome. In the Enterprise model analysis, this simulation paradigm is valuable since it allows us to represent the model in a modular way. In this sense, this paradigm makes it easier to represent enterprises with different structures such as the virtual enterprise, the distributed enterprise and the networked enterprise. This paper presents both an environment for developing an enterprise model using a business-oriented language - called Coordinates - and its automatic transformation in a simulation model that is based on DEVS formalism. In order to execute the simulation model in a distributed environment we have used HLA specification as middleware.

Keywords: Enterprise Model, DEVS, Distributed simulation, HLA.

1 Introduction

Enterprise Model (EM) can be used to describe the organization "as-is", to evaluate its processes, to define a new or desired organization, to conceptualize workflows as well as to specify the information requirements of the organization. An EM is usually composed of entity relationship diagrams, state diagrams and activity diagrams.

Nowadays, the enterprises are changing their structures in order to overcome the increasingly and strict demand of the market. In this sense, the companies are making alliances with others, giving as a result new structures like virtual, network and extended enterprises. This represents a new challenge in the development and analysis of the EM. Mainly, because enterprises are disaggregated in branches and nodes dispersed geographically, and sometimes independently.

Simulation of an EM is valuable since it allows us to analyze the organization behavior in different scenarios, real or virtual ones. That is to say, simulation in this context provides a powerful tool to analyze how the system will perform in its “as-is” configuration and under a myriad of possible “to-be” alternatives. Then one can confidently choose the best way to run their business. But to generate a simulation model (SM) a detailed understanding of the organization is required as regards the processes sequences, the resource availability, the constrains, the resources needed by tasks, among others. Most of this information is contained in the EM and a replication

in a SM not only represents a loss of time but can also lead to inconsistency. Then an automatic generation of the simulation model from EM is required. Many proposals overcome this problematic but they do not take into account the possible organization structure. When the organization has a distributed structure as for instance a supply chain or decentralize, the inherent complexity of the EM and the multiplicity of actors involved, make the design and development of simulation model very difficult. This problematic is overcome using distributed simulation. In this way, the EM is built modularly and independently, even when the participants use different languages and platforms. According with Strassburger [1], the requisites for a successful application of distributed simulation are: (i) the existence of complex simulation models, and (ii) the existence of a problem or question, which can only be solved if the models are combined. We think that the simulation of an Enterprise model meets these requirements.

Our proposal presents an architecture to develop a distributed and executable enterprise model (DE^2M), which introduces a solution to develop and run EM that takes into account possible company structures. This architecture allows the user to develop an EM using a business-oriented language and to analyze it using discrete event simulation. The distributed execution of the model reflects the distributed structure of an organization. The main goal is to integrate the simulation models from different members (such as enterprise branches) into one larger model. This is known as "distributed simulation" [2]. In this context model behavior is exposed to other simulation software via network simulation protocol. In this sense, the environment admits the analysis of Enterprise model using distributed simulation.

Then, the EM can be run in a local environment or in a distributed one depending on the enterprise structure. The execution in a local environment helps to analyze and to improve the innermost processes before analyzing the relation with other members in a distributed environment. The distributed simulation uses HLA [3] as middleware, in this particular case, we have used poRTIco [4], an implementation of RTI. The model is developed once and reused in both environments. This is possible since we have used components-based simulation (DEVS formalism) where the models are separated from the simulation engine.

This work is organized as follows. Firstly we depict the architecture and its objectives; next we describe the mechanism to obtain the federate which represents the EM in a distributed environment. After that, we present an example. Finally, conclusions are provided.

2 DE^2M Architecture

Figure 1 shows the architecture of the environment called DE^2M - Distributed and Executable Enterprise Model.

In our proposal, we have decided to choose a two-layer architecture in order to hide the simulation process. The top layer – the Conceptual Model Layer - represents the knowledge about an organization in terms of processes, tasks, resources, and objectives. The bottom layer – the Simulation Layer - represents the behavior of the

organization in terms 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 [5]. The Document component is called Coordinates Model and has two components: Domain and Implementation. The functions that are offered in this layer are related to the development of the EM using a business-oriented language, in this particular case Coordinates language [6]. 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, because it represents 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 proposed in [7]. It is responsible for translating the conceptual model in a simulation model without the user intervention. So, this environment is oriented to business experts more than simulation experts.

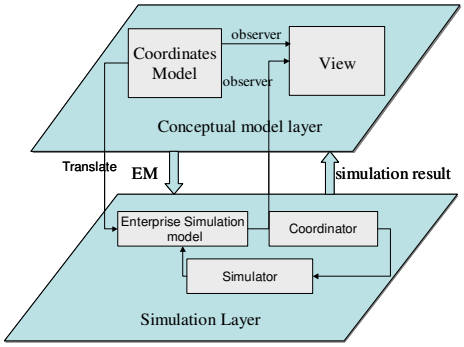


Fig. 1. DE^{2}M architecture.

As regards Simulation layer, it is based on DEVS formalism [8]. DEVS is a simulation formalism that uses discrete event models and offers an open approach based on general concepts about system dynamics. It allows for model decomposition with a hierarchical and modular approach. There are two reasons to choose this formalism in this layer. The first is the hierarchical construction of the model and the second is the fact that the models are different from the simulation engine, this characteristic allows us to generate the SM only once, and use it in both, local and distributed environment with different associated engine. The components of this layer are: Enterprise Simulation Model, Coordinator and Simulator.

The Enterprise Simulation Model component defines the building-blocks used in the development of simulation model. These building-blocks are in correlation with the concepts defined in the conceptual model and represent the behavior of them. As an example, let's consider a Task (see figure 2). In the conceptual model, a task represents an activity carry out in the enterprise that uses resources to accomplish it objective. The tasks can be decomposed into other tasks, through Task versions. We will call this type of task composite task. Examples of tasks can be: fill customer

order, check payment, check goods availability, among others. Then, in Enterprise Simulation Model, the building-block that encapsulates the behavior of an atomic task (tasks that do not have task version) is TaskAtomicDevs and TaskVersionDevs encapsulates the behavior of task version. TaskAtomicDevs is a Devs model with a special definition of states, transition functions, output function and time advance function. TaskVersionDevs is a digraph composed of other Devs models.

In the simulation layer there is a Translator process that takes the conceptual model and generates the simulation model. Figure 2 shows the relation set between building-blocks used in conceptual model (gray classes) and the building-blocks used in simulation model with some context information.

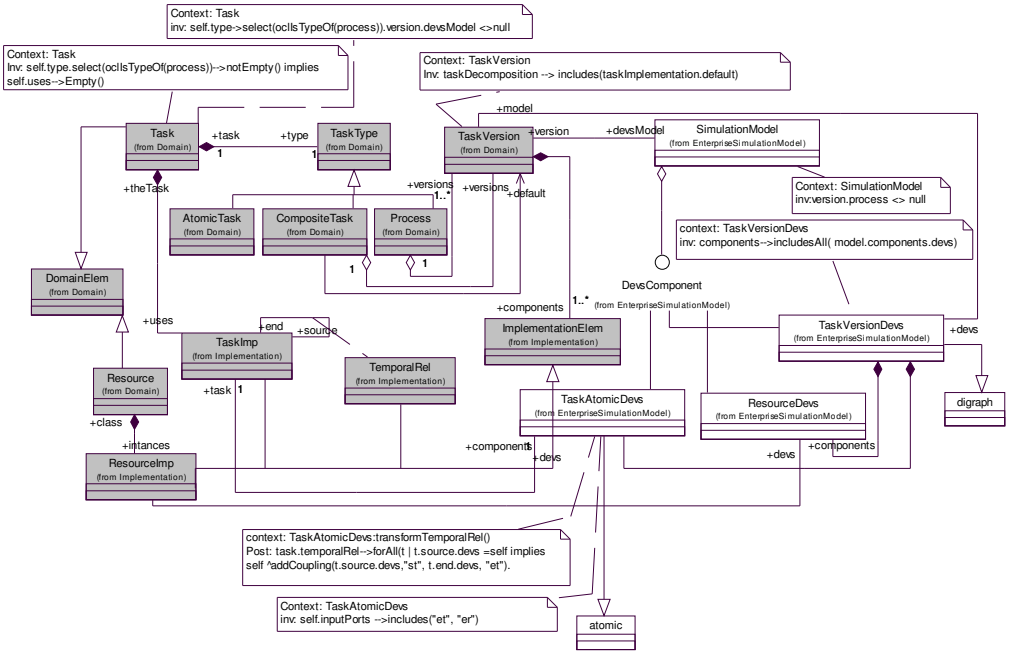


Fig. 2. Class diagram relating conceptual model Building-blocks and Simulation model building-blocks.

The simulation model building-blocks extend from classes belonging to DEVSJAVA 3.1 framework [9].

The Coordinator component is the engine of simulation model. It covers the root coordinators. There are two root-coordinators: one in a local and other in a distributed environment. This component is responsible for controlling the simulation execution. This architecture admits two different executions: local and distributed. In this way,

there are two root-coordinators directing the simulation in various environments: coordinator, provided by DEVSJAVA framework and CoordinatorE2M.

In local environment, the coordinator is in charge of directing simulation run. But in distributed environment, the CoordinatorE2M does it.

CoordinatorE2M overwrites the simulate method in order to solve problems with time management. There are two generic approaches to time management in distributed simulation: conservative and optimistic [10]. 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 (i.e. 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 [11].

On the other hand, HLA allows federates to 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 considered at the time of generating the HLA federate.

Since DEVS does not implement rollback, and since time management is centralized in root coordinator, the way to communicate and to interact with the RTI is using a conservative scheme.

Table 1. Comparing 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 coord.	Root coordinator	RTI
Message sent	Regulated by Coordinator.	Schema publish/subscribed.

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 Simulator component covers the engine associated with each building-block participating in the SM. According to DEVS, each atomic model has a simulator associated and each coupled model has a coordinator associated. The coupleCoordinator and coupledSimulator are defined in DEVSJAVA framework. The first one is associated with coupled models whereas the second one is associated with atomic models.

Finally, the View component is made up of entities that appear in the user interface showing the concepts (task, resources, states, etc), graphics and results. In order to update the view, we have used the observer design pattern [12].

3 Federate Structure

In this section we will 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 [3], 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 3 shows the FederateE2M structure according to the federate definition. The federate has a Federated Ambassador, a simulation code (SimulationModel in the picture), and an RTI-Ambassador.

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 in order 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 federate, it must be changed into a message that can be understood by the federate simulation code. In the same way, when the federate sends interactions towards other federates, the messages from SimulationModel must be translated into interactions, which in term, will be sent to the RTI in order to be published.

From this general abstraction, the class diagram shown in figure 4 defines the class FederateE2M. It is composed of a SimulationModel and implements the IODevs interface. That it to say, this class is a DEVS model and a federate too.

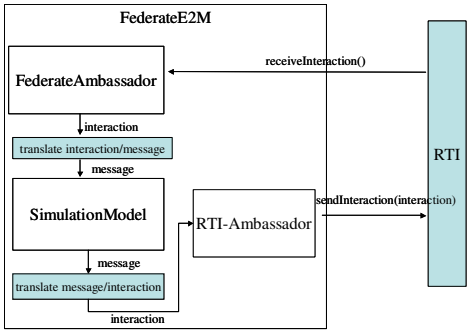


Fig. 3. FederateE2M structure.

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. The federate FederateE2M is conservative, therefore when it joins to a federation, it defines itself as time Constrained and time Regulated.

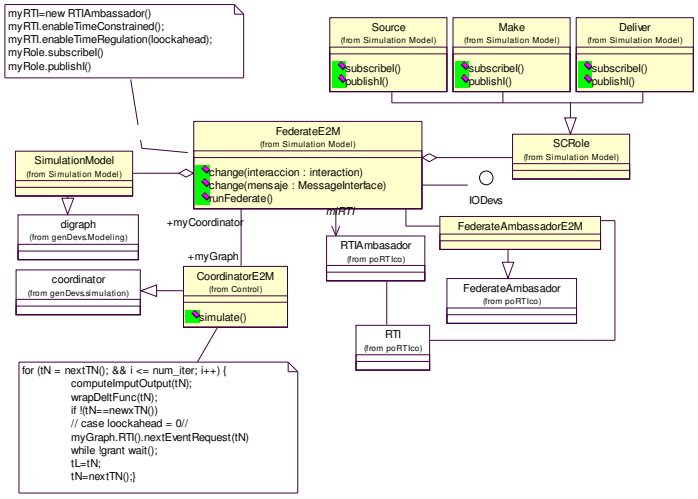


Fig. 4. FederateE2M class diagram.

Figure 5 shows a collaboration diagram depicting the interaction among models, engines and controllers in a distributed environment in the particular 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 FederateE2M is responsible for interacting with others federates and it works with RTIAmbassador and FederateAmbassador in order to achieve its objective. The SimulationModel is the enterprise behavior and has a coupledCoordinator associated.

CoordinateE2M asks for next event time to the coupled coordinator associated, this message is propagated in simulation engine hierarchy.

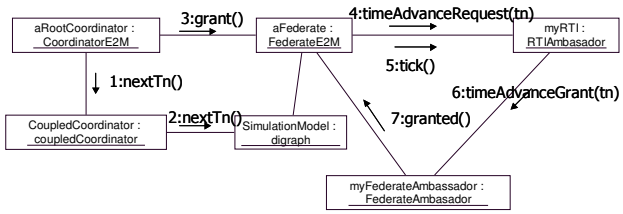
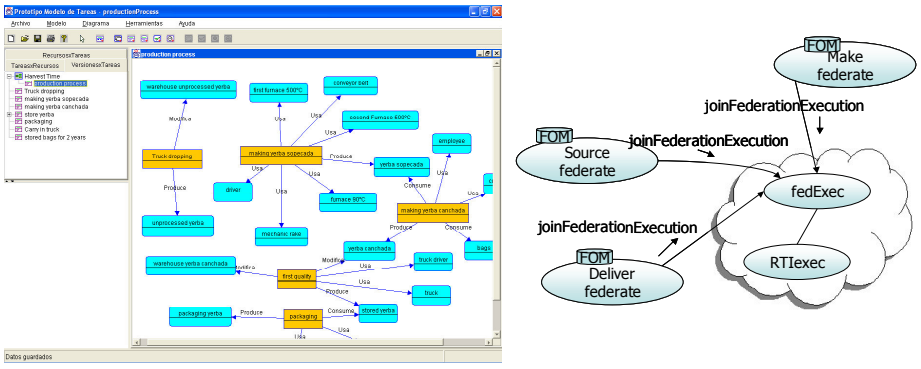


Fig. 5. Time management in distributed environment.

When the coordinatorE2M has the Tn value, before advancing the simulation time, it asks for grant to the FederateE2M, next FederateE2M invokes the services timeAdvanceRequest and waits for grant. After a few seconds, RTIAmbassador sends the grant invoking the call back function timeAdvanceGrant. Then, FederateAmbassador informs about grant to FederateE2M, which in turn, return the grant to the coordinatorE2M. Finally, the root coordinator follows the simulation cycle.

4 Example

In this example we will analyze a supply chain made up of a factory, a wholesaler and a retailer. 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" [13]. 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. Due to lack of space; we will only show how the factory develops the EM using the proposed architecture. As an example, figure 6(a) 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 others processes are defined giving as a result the EM, which can be analyzed and its results can contribute in the processes improvement. Then, a distributed simulation can be executed and the FederateE2M is developed using the EM. In this way, we consider the following scenario: the RTI is running in a machine, a federation has been created and federates have been joined to federation. Here, our federate is ready to join the federation. Then, when it receives the run command, it will try to join a federation. Figure 6(b) shows the federates joining the federation. Federates are named with the role that each enterprise plays in the supply chain. That is to say, the factory appears as the make federate, the wholesaler is the source federate and the retailer is the deliver federate.



(a) (b)

Fig. 6. (a) task diagram Snapshot. (b) a representation of a supply chain federation.

The results obtained from this simulation are shown in figure 7. With this example we propose to analyze the customer satisfaction as the total customer orders satisfied in full and in time. The simulation run was fed with 1.500 customer orders. Figure 7 shows that the 73% of the customer orders were satisfied in time and in full. The 18% suffered from delay and 9% were not satisfied at all. This reveals that 21% of time customers are not happy with the services offered by this supply chain.

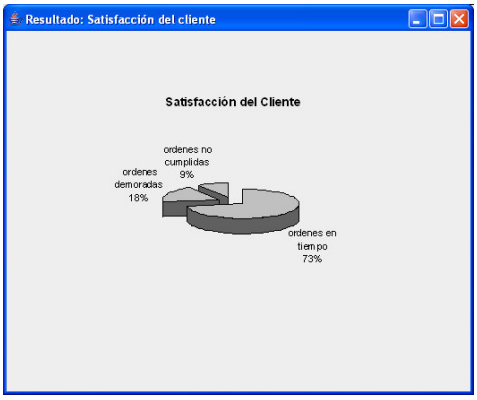


Fig. 7. Simulation results.

5 Conclusion

This work presents an integrated environment that allows modeling and simulation enterprise models. 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 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 depict a different aspects of the Company, without losing sight of the integrated model that these views integrated all together.

The simulation model is built from the integration of existing knowledge in different views. This helps 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 automatic, the user does not need to know about how to use the tool.

The Simulation layer has been designed with the MVSC design pattern. This allows us to reuse the SM in distributed environment.

Since the simulation model is developed using DEVS, in order to be run under HLA specification, we have defined the structure of a particular federate that uses DEVS in the simulation code. Then, it was showed how DEVS simulation cycle changed in order 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 Enterprise simulation model in a distributed environment. So, the

existing simulators were reused. On the other hand, this proposal is important when used in a networked company and supply chain. Thus, the construction of an executable enterprise model in a modular way has important advantages. 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 needed to develop DE^2M, and simultaneously it facilitates the EM modification and test.

Acknowledgments. The author wants to acknowledge the "programa de becas de doctorado para docentes de la UTN" that makes this project possible.

References

1. Lendermann, McGinnis, McLean, Strabbuger, Heinicke, Taylor. Panel: Distributed Simulation In industry – a real-world necessity or Ivory Tower Fancy? Proceeding of the 2007 Winter Simulation Conference.
2. Verbraeck A.: Component-based Distributed Simulations. The way forward? In Proceeding of the eighteenth workshop on parallel and distributed simulation. Kufstein, Austria (2004). ISBN-ISSN: 1087-4097, 0-7695-211-8.
3. IEEE 1516-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules (2000)
4. poRTIco version 0.7 October 2007 URL: http://www.porticoproject.org access date: 25/10/07
5. Buschmann F., Meunier R., Rohnert H., Sommerlad P. and Stal M. In: Pattern-oriented software architecture, A system of patterns. (1996).
6. Mannarino G., Henning G. and Leone H., , J.J. Mills and F. Kimura (Eds.). Coordinates: A Framework for Enterprise Modeling. In: Information Infrastructure Systems for Manufacturing. IFIP-Kluwer Academic Publishers. (1999)
7. James Nutaro, Phil Hammonds. Combining the Model/view/Control design pattern with the DEVS formalism to achieve rigor and reusability in distributed simulation. In: The society for modeling and simulation international. Issue 1. (2004)
8. Zeigler, B. Praehofer and Kim. Theory of Modeling and simulation. Integrating Discrete event and continuous complex dynamic system. Second edition. Academic press (2000)
9. DEVSJAVA 3.1 URL: http://www.acims.arizona.edu/SOFTWARE/software.shtml#DEVSJAVA
10. Fujimoto R. Distributed Simulation Systems. In: Proceeding of the 2003 winter Simulation conference (2003)
11. Zeigler, B. Ball, G. Cho H. and Sarjoughian H. The DEVS/HLA distributed simulation environment and its support for predictive filtering. In: Advance simulation technology Thrust (ASTT) DARPA Contract N6133997K-007 (1998)
12. Gamma E., Helm R., Johnson R.and Vlissides J. Design Patterns. Elements of Reusable Object-Oriented Software, Addison-Wesley. (1994)
13. Villa, Emerging trends in large-scale supply chain management. In: ICPR 16th international conference on production research. (2001).