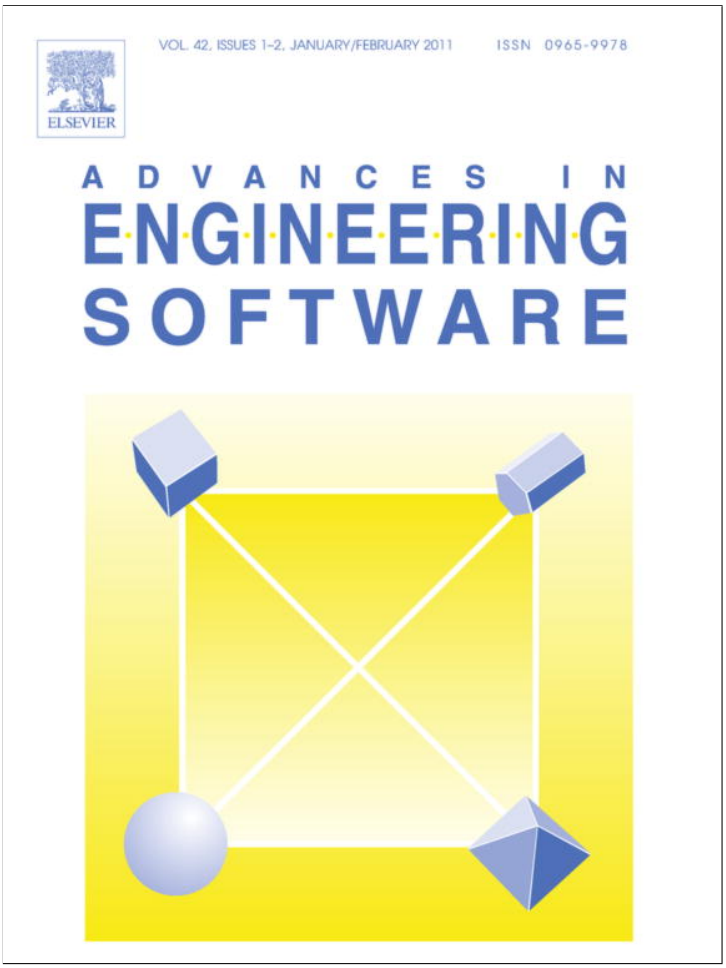


(This is a sample cover image for this issue. The actual cover is not yet available at this time.)

Advances in Engineering Software 47 (2012) 80–103

Contents lists available at SciVerse ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

DE^{2}M: An environment for developing distributed and executable enterprise models

María de los M. Gutierrez [a,*], Horacio P. Leone [b]

[a] Universidad Tecnológica Nacional, Fac. Reg. Santa Fe CIDISI, Lavaise 610, 3000 Santa Fe, Argentina
[b] Universidad Tecnológica Nacional, Fac. Reg. Santa Fe CIDISI, Instituto de Desarrollo y Diseño, INGAR (CONICET), Lavaise 610, 3000 Santa Fe, Argentina

ARTICLE INFO

Article history:
Received 12 June 2008
Received in revised form 25 November 2011
Accepted 5 December 2011

Keywords:
HLA
DEVS
Discrete events simulation
Enterprise model
MDA
Simulation model

ABSTRACT

The distributed and executable enterprise models are one of the most important sources of an organization's information requirements where the business expert has not only an appropriate representation of the organization in terms of processes, information flows and user roles, but also a simulation capability for the interpretation of the dynamic behavior. We present an environment to support the development of such a model. It uses a MDA approach to acquire the simulation model from conceptual model. The simulation model can run both distributed and local environment.

1. Introduction

Business process (re)engineering, benchmarking, supply chain management, and several other techniques and activities, use the enterprise model (EM) to get a general view of the organization which will enable the evaluation of processes from different perspectives depending on the organization's objectives. Enterprise models represent the organization knowledge in terms of its processes, its material and information flows, the available and required resources, functional areas interactions, available information technology and time constraints. EM can be employed to describe the organization "as-is", to evaluate its processes, to define a new or desired organization as well as to conceptualize workflows and to specify the information requirements of the organization [3]. An EM is usually composed of entity relationship diagrams, state diagrams and activity diagrams.

The simulation of the processes of an EM is very valuable since 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. Usually, an organization develops its EM with other purposes than simulation. Therefore, it is necessary to construct a simulation enterprise model (SM). The SM includes knowledge of the EM but also it needs more data and information.

Also, in the context of the new agile and dynamic structure organizations like virtual enterprises [2,7,8,9,18,49], network enterprise [28,30] and extended enterprises [10,5,8], the complexity of the business processes implies to share information among departments, areas, nodes and branches of an organization, some time they are distributed geographically having different cultures and speaking different languages. As consequence, SM development is a difficult, expensive and time consuming task. SM is usually relegated to evaluate some aspect of an organization, as for example logistic performance, bullwhip effect, production process and some workflows.

Organizations need tools that help them to construct SM from their EM. The Model Driven Arquitecture (MDA) [38,33] presents a very promising approach to reduce the difficulties and cost of SM construction. Currently the use of MDA to translate from a platform independent model (PIM) to a platform specific model (PSM) is widely used in different areas, such as to align IT infrastructure of an enterprise with its business needs and requirements [27]; also to solve Web services composition [31] and to validate models as workflows [51], among others.

In this work, we present an environment to develop EM with a business-oriented language and a MDA approach to construct SM from EM. In the building of SM we have used DEVS formalism [52], because it enables the modular assembly and helps to reuse components. The MDA approach proposed focuses on the predominant challenges encountered in simulation business process development:

* Corresponding author. Tel.: +54 342 4608585x258103.
E-mail addresses: mmgutier@frsf.utn.edu.ar (M.d.l.M. Gutierrez), hleone@santafe-conicet.gov.ar (H.P. Leone).

0965-9978/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.advengsoft.2011.12.002

(i) an enterprise model that has been designed with business requirements and goals in mind is not necessarily a simulation model, with events, time and metrics,
(ii) a simulation model derived from an enterprise model may not be so easily reused and integrated in a distributed simulation environment,
(iii) a simulation model derived from an enterprise model must be semantically equivalent

This work is organized as follows. Section 2 presents an overview of the related work in the enterprise simulation domain. Section 3 introduces the main concepts, like DEVS and HLA, used in simulation model construction and execution. Section 4 presents the architecture of the environment. The language used in development of EM is presented in Section 5. Section 6 describes the construction of SM. Section 7 illustrates the way in which to execute the SM. Section 8 shows an example and finally in Section 9 conclusions are drawn.

2. Literature review

In some approaches, it is proposed to define processes using a business-oriented language as SIMPROCES [39], or an English-like language as SIMSCRIPT [37]; but the model that is defined is a SM. In this case, the user must realize that s/he should represent actions such as "*count number of orders released*" when in daily actions, this task is not done, that is, it is not really a task. Then we can say that the business process knowledge is mixed with simulation knowledge. This simple fact can complicate the development of the SM.

In literature, supply chain modeling and simulation are considered independently [12,26,36] without taking into account that a supply chain is defined as a virtual enterprise and, according to that, it can be modeled using enterprise modeling techniques. That is to say, there is no connection between supply chain analysis and enterprise model and simulation. As regards supply chain simulation, we have observed that there are two mainly approaches to construct the model and execute it: global central model and distributed model. The global central model uses a single global model running in a single machine [12,27,6,4,13,36,46]. This model reproduces the nodes of the supply chain, and relations among them. The distributed model employs several modular models running in a distributed environment [25,50,48]. The first approaches have not interoperability problems, but they cannot reuse the existing local models of the supply chain nodes. The second approaches are better for supply chain simulation; however there are few contributions describing supply chain simulation in distributed environments [42]. Perhaps, one of the main reasons is the difficulties in providing interoperability among modular simulation 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) [40,41]. 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.

DESSCOM [4] presents an architecture of the decision making tool for supply chains based on sound modeling and problem solving approaches. XML-nets (Mevius and Pivernik; 2004) presents an approach to SCPM (supply chain process management) based on a type of high-level Petri-nets, which can manipulate XML document representing the objects (customer order, bills of material, materials, finished goods, etc.). The XML-net can be executed and analyzed by a workflow engine. Vieira and Guilherme [45] proposes an environment that uses ARENA [1] to develop and simulate a supply chain model. This work has a specific objective: to analyze the bullwhip effect and the benefits of collaborative planning, forecasting and replenishment. ONE [17] is a tool to support decision makers for the assessment, design and improvement of a supply chain. The supply chain is represented using arcs and nodes, where each node represents a member and the arcs represent connections between them. This tool has an optimization module offering a set of optimization methods such as mathematical programming and genetic algorithm. This module allows user to obtain new optimized networks depending on the goals set. EASY-SC [26]) is a tool that represents the supply chain as a graph with nodes and arcs. The environment is developed using Java language.

These tools, offer model and simulation process capability, using a graphic representation based on some formalism like petri net [34], DEVS models [52], or in ad hoc representation [26,17]. These models are then, interpreted by a machine obtaining data used in the analysis process. But this representation oriented to a specific simulation objective and with a simulation language is not reconciled with the enterprise expert knowledge. Heretofore, tools and methods proposed fail in the integration of the enterprise knowledge and present the simulation model oriented to analyze some characteristic. In any case, the tools propose to reuse modes that represent some aspect of the enterprise.

3. Preliminary concepts

3.1. DEVS formalism

DEVS is a simulation formalism that uses discrete event models and offers an open approach based on general concepts about system dynamics. It makes possible model decomposition with a hierarchical and modular approach.

In this work we have used parallel DEVS [11]. In agreement with the literature on DEVS, the specification of an atomic discrete event model is defined as structure:

$$DEVS = \langle X, Y, S, \delta_{int}, \delta_{\text{ext}}, \delta_{\text{con}}, \lambda, \text{ta} \rangle$$

where

$X = \{(p, v) | p \in \text{InPort} \wedge v \in Xp\}$ is the set of input ports and values.
$Y = \{(p, v) | p \in \text{OutPort} \wedge v \in Yp\}$ is the set of output ports and values.
S is the set of states.
δ_{int}: $S \to S$ is the internal transition function, it defines the state changes caused by internal events.
δ_{ext}: $Q \times X^b \to S$: is the external transition function, it determines the state changes produced by external events.
δ_{con}: $S \times X^b \to S$: is the confluent transition function.
λ: $S \to Y^b$: is the output function, it specifies the output values caused by the internal transitions.
ta: $S \to R_0^+ \cup \infty$: is the time advance function. Special cases 0 and ∞ identify transitory states or passive states. In the first case, the stay in state s is so short that no external events can intervene. In the second case, the system will stay in such a state forever unless an external event interrupts its slumber.
$Q = \{(s, e) | s \in S, 0 \leqslant e \leqslant \text{ta}(s)\}$ is the set of total state and e is the elapsed time since the last transition.
X^b denotes the collection of bags over X (sets that some elements may occur more than once).
Y^b denotes the collection of bags over Y.

The specification of a DEVS coupled model is defined as structure:

$$COUPLE = \{X, Y, D, \{Mi | i \in D\}, \text{EIC}, \text{EOC}, \text{IC}\}$$

where

$X = \{(p, v) | p \in \text{InPort} \wedge v \in Xp\}$ is the set of input ports and values.
$Y = \{(p, v) | p \in \text{OutPort} \wedge v \in Yp\}$ is the set of output ports and values.
D is the set of the components' names.

$\{Mi | i \in D\}$ is the set of the component. Components are DEVS models for each $i \in D$:

$$Mi = \{X_M, Y_M, S_M, \delta_{\text{int\_}M}, \delta_{\text{ext\_}M}, \delta_{\text{con\_}M}, \lambda_M, \text{ta}_M\}$$

$\text{EIC} \subseteq \{((M, ipN)(d, ipd))\}$ $ipN \in \text{InPort} \wedge d \in D \wedge ipd \in \text{InPort}_d$
$\text{EOC} \subseteq \{((d, opd)(M, opM))\}$ $opM \in \text{outPort} \wedge d \in D \wedge opd \in \text{OutPort}_d$
$\text{IC} \subseteq \{((a, opa)(b, ipb))\}$ $a,\ b \in D \wedge opa \in \text{outPort}_a \wedge ipb \in \text{InPort}_b \wedge a \neq b$

3.1.1. DEVS simulation cycle

DEVS proposes separate models of their simulators, thus, each atomic model has a simulator associated and each coupled model has a coordinator associated. A special coordinator called root coordinator, directs the simulation. The DEVS simulation mechanism consists of four stages: (i) advance the function time until the time of the next event, (ii) calculate the inputs and outputs, (iii) send message and (iv) execute the internal and external transition functions. Fig. 1 shows the DEVS simulation cycle implemented in the root coordinator.

As we can see in the cycle, the coordinator asks its components to compute the time of the next event (TN). Then, the smallest of them is called TN*. The simulation time will be advanced to TN*. We represent this as follow:

$$\text{TN}^* = \min\{\text{tn}_d | d \in D\}$$

where

D is the set of components and tn_d is the next event time of the d component.

With this value, the coordinator groups in set I all the components whose next event time is equal to TN*, known as imminent components:

$$I = \{d | \text{tn}_d = \text{TN}^*\}$$

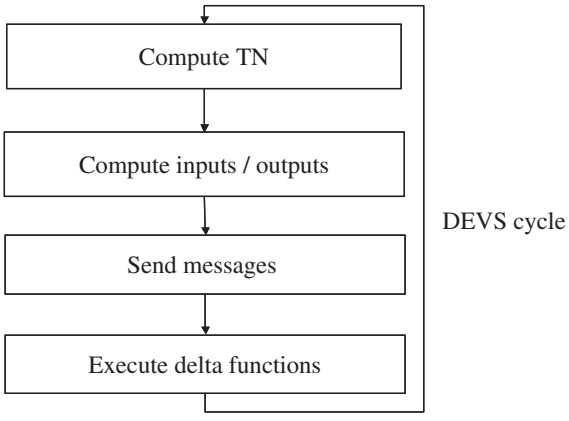


Fig. 1. DEVS cycle.

Next, it asks to calculate inputs and outputs. That is, the imminent components execute their output function generating messages that will be the inputs to other components.

$$\forall\ d \in I\ \text{send}(\text{computeInputOutput}, d)$$

When the coordinator receives these outputs, using coupling information, it sends the messages to the corresponding input ports. Here the coordinator makes a set M of all the components that have a set of inputs different from empty.

$$M = \{d | \exists p \in \text{Inport}_d\ \text{messageOnPort}(p) \neq \phi\}$$

In the following step, the coordinator sends the messages to the components, which will receive them in their input ports. Finally, it arranges the components to execute their transition functions in the following way: all the components that are in the intersection of sets I and M execute the confluence transition function, all the remaining components of I execute the internal transition function and the remaining components of M execute the external transition function:

$$\forall\ d \in I \cap M, \text{send}(\delta\text{con}, d)$$
$$\forall\ d \in M - (I \cap M), \text{send}(\delta\text{ext}, d)$$
$$\forall\ d \in I - (I \cap M), \text{send}(\delta\text{int}, d)$$

This cycle is repeated until there are no imminent components or a given number of iterations have been fulfilled.

3.2. HLA (High Level Architecture) specification

The High Level Architecture (HLA) is a simulation framework developed by the Defense Modeling and Simulation Office (DMSO) [15] whose more important objectives are reusability and interoperability in complex simulation systems. This framework was standardized by the IEEE, giving rise to the standard 1516-2010 [22]. HLA provides a general framework within which simulation developers can structure and describe their simulation applications. When a simulation is implemented as part of an HLA-compliant simulation, it is referred to as a federate. HLA simulations are made up of a number of HLA federates and are called federations. The HLA definitions involve three main elements: rules, interface specification [23] and object model template (OMT) [24]. The rules are a set of ten items that describe the relationship and responsibility among a federation's components. The interface specification defines the functional interface between HLA federates and the run time infrastructure (RTI). Finally the OMT describe a common format for representing the object model in HLA. Simulations, which use the HLA, are modular in nature allowing federates to join and resign from the federation as the simulation executes. Each federate has a Simulation Object Model (SOM) that describes the data that the federate can produce or consume.

Each federation has a Federation Object Model (FOM) that describes the common parts of the participating federates' SOM to be used in the federation.

The RTI is software that is used in the federation execution. It is an implementation of the interface specification and consists of a set of services. The RTI has services to start and end simulation execution, transfer data between federates, coordinate the simulation time, etc. This software is out of the specification and specific software suppliers provide it.

4. DE^{2}M environment architecture

In our proposal, we have decided to choose a two-layer architecture (Fig. 2), in order to hide the simulation process.

The first layer – conceptual model layer – is the one the user works with. The functions that are offering are related to develop

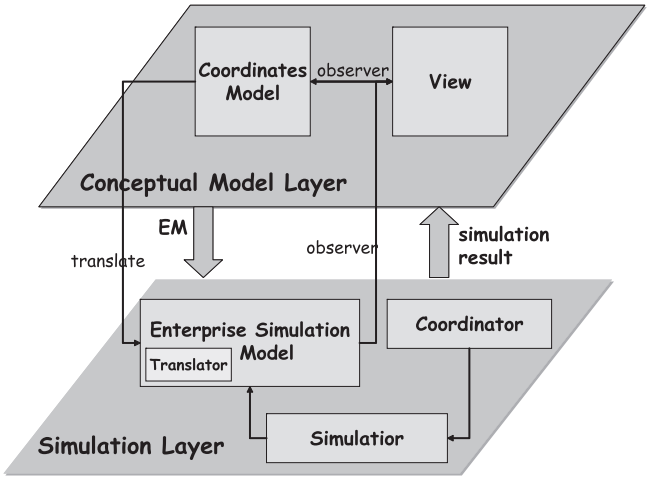


Fig. 2. DE^{2}M architecture.

the conceptual EM using a business-oriented language, in this particular case Coordinates language. Then, the user can develop the model in an easy and friendly way using a known vocabulary (like task, resources, processes, states) and a graphical interface. As result, the obtained EM is called conceptual model (CM), because it represents the conceptualization of the organization. This model does not have simulation information.

The simulation layer provides functionality to acquire SM, execute it either locally or in a distributed environment; and compute metrics. This layer implements the MSVC design pattern proposed by Nutaro and Hammonds [32], which is oriented to develop component-base simulation. It proposes a clear separation between models, simulation and distributed computing. The component-base simulation paradigm [43] emphasizes the development of simulation model in term of components that interacts each other. A component can be reused and/or substituted in the simulation model and interpreted both isolated and in cooperation with others components. A component, also know as building-block, is defined by BETADE-research group [44] as *self-defined unit, interoperable, reusable and replaceable that encapsulate its internal structure given services and useful functionality to its environment through its interface defined previously*.

5. Conceptual enterprise model

DE^{2}M environment gives support to the business process modeling and simulation. It is the starting point for analyzing, evaluating, describing, improving, reorganizing and managing an organization.

We have adopted the Coordinates language, which has met the requirements of a language for enterprise modeling [29,17]. This language proposes the EM development from three views: (i) **Task view**, representing the functional view of an organization (ii) **Dynamic view**, representing the resource life cycle as a consequence to participate in task execution, and (iii) **Domain view**, representing the resources relationship such as composition, association and inheritance [20]. As they are related to one another, they make possible to obtain a general view of the enterprise. Fig. 3 shows the perspectives that compound the conceptual EM.

The **Task view** is represented by task model. It describes an organization as a set of Tasks, which make use of different Resources in order to achieve their goals. Different semantic links (Task-Resource Link) relate Tasks with Resources: *uses*, *modifies*, *processes*, *creates* and *eliminates* are some of the primitives. That is to say, a Task *modifies* a Resource when the Resource changes its state because of the Task execution. The *uses* relationship means the Resource is seen as a tool during the Task execution. The *creates* and *eliminates* relationships indicate a Resource is generated/eliminated in the domain as the result of the Task execution. A task *processes* a resource when the resource waits in a queue. Any other relationship can be expressed through the combination or specialization of the previous links. The Coordinates language supports Task description at multiple levels of detail and it is composed by Tasks. A *Process* must only be the root in a 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 *Task Version*, each one encapsulating a particular way of accomplishing the task. *Task Version*, 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. A *Task Version* is described in terms of a particular set of *Tasks*, *Resources*, *Temporal Links* and *Task-Resource Links*. One or more preconditions, each one identifying a particular *Resource state*, constrain the structure that a *Task* may assume. Fig. 4 shows the class diagram implementing Task model in the Conceptual model layer of the architecture. This model focuses on Task Version concept.

Domain view is depicted by *Resource* model. This model focuses on the structural characteristics of the resources. *Specialization*, *Composition* and *Association* relationships are the basic links used to represent the structure of *Resources*. Fig. 5 shows the class diagram use to implement this view. The class *Resource* identifies the concept resource. The class *RelResource* represents the relation among resources. It has three subclasses: *Association*, *Inheritance* and *Composition*.

The **Dynamic view** is depicted by a Resource life cycle model. This model is a Resource centered view as it emphasizes the way a particular *Resource* evolves because of participating in different *Tasks*. This view is based on the statechart formalism proposed by Harel [21]. The evolution of a resource is represented in terms of states and transitions. A transition abstracts a change of state and is produced because of the starting or ending of a *Task*. Fig. 6 shows the class diagram to implement this view. Note that the *StateTransition* class has two relationships with task: *startTrigger* and *endTrigger*. They indicate that the transition takes effect when task begins or finishes its execution respectively. These relations are the nexus between dynamic view and task view. That is, on the one hand task execution makes resources change states and on the other hand the resource state is precondition of tasks execution.

6. Simulation model construction

This section explained the transformation process from CM to SM. It is based on the schema shows in Fig. 7. So, a source model (CM) writing in Coordinates language is automatically translated by transformation tool in an output model (SM) writing in DEVSJAVA language [16]. The implementation of the transformation tool has been based on transformation definition writing in OCL language [47], this definition consist on 7 rules that are described in Appendix A.

The CM comprises of three models related to each others depicting the different EM views. Although the translation begins when a task version from Task view is selected, translation tool takes into account the three views. Then, the SM acquired represents the organization as a whole and not simply the functional view. Although the translation process is complex, it has the advantage to represent the behavior of the organization in all its aspect, having better data analysis on which to take decisions.

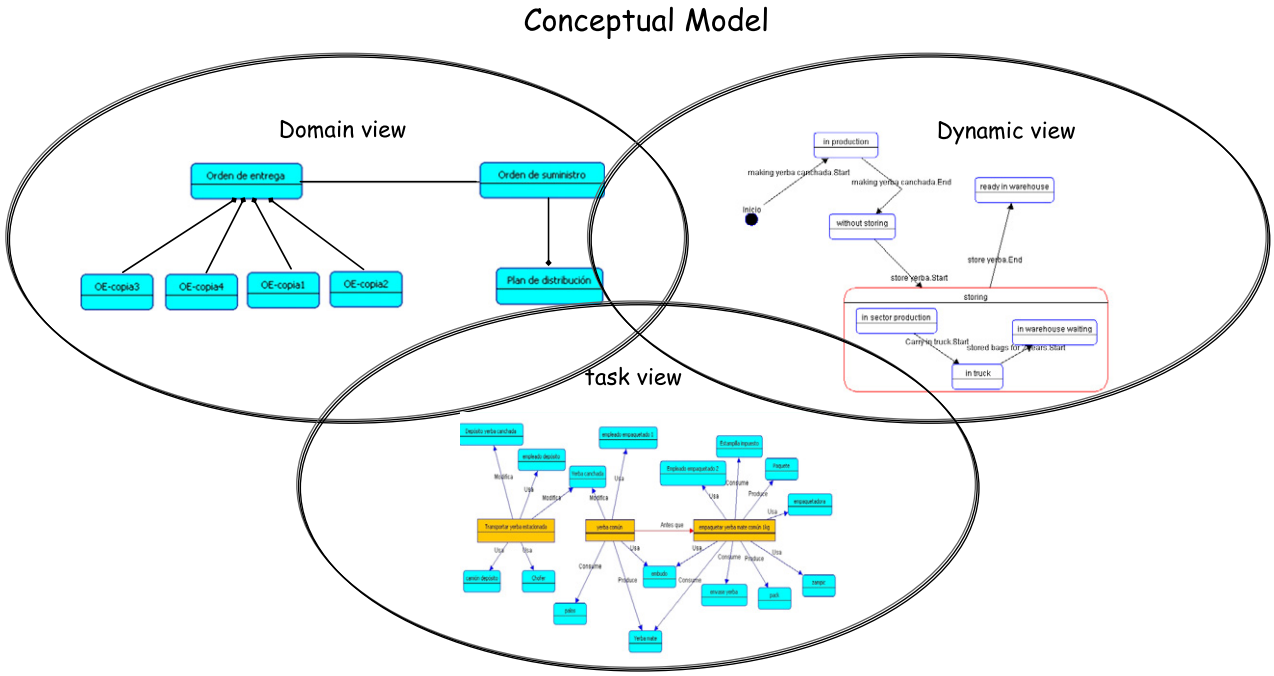


Fig. 3. Conceptual enterprise model.

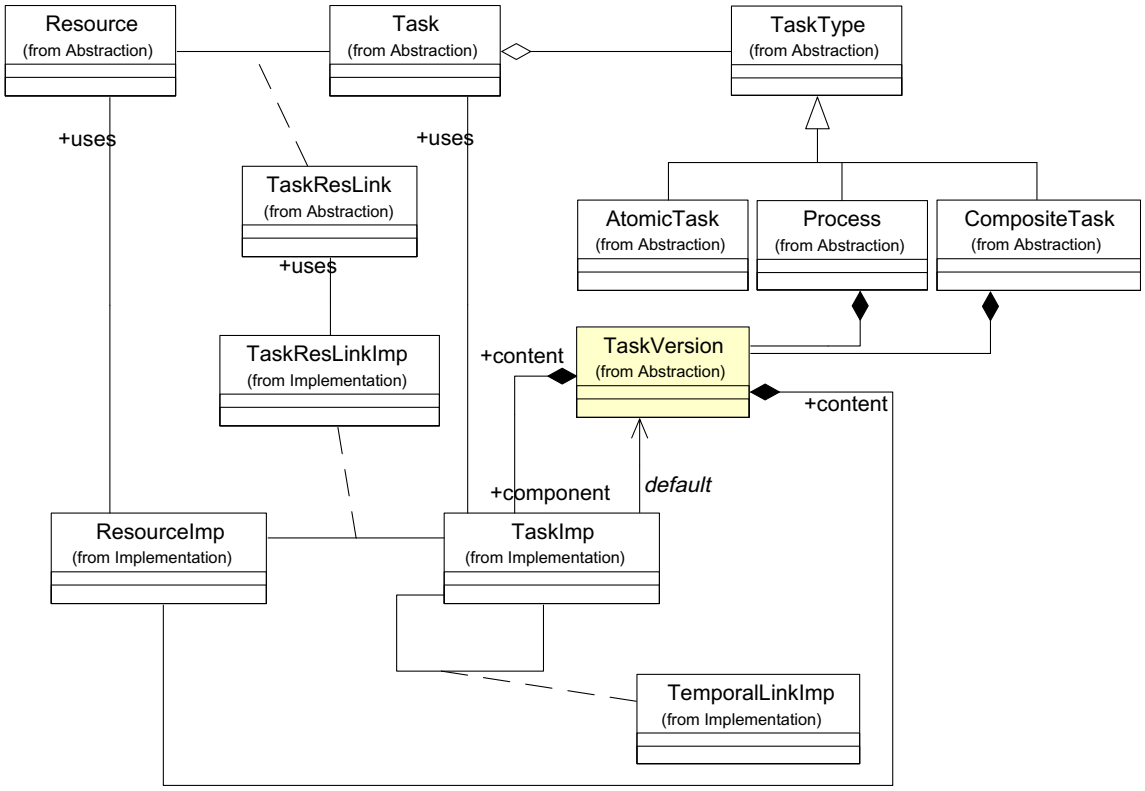


Fig. 4. Task model class diagram.

To achieve the automatic transformation of CM in SM was necessary to taking into account several aspects. On the one hand, we have defined and implemented the components of SM necessary to represent the concepts found in CM and the rules that govern the transformation. On the other hand, we have to solve how to run SM in a way that is as transparent as possible. For this reason, assumptions have been made and set in the *Experimental-Frame* component as for example the metrics defined in *Resume* model, which could be obtain as result of simulation run; the ending of simulation setting in *Acceptor* component, and resources that feed the simulation model defining in *Generator* component.

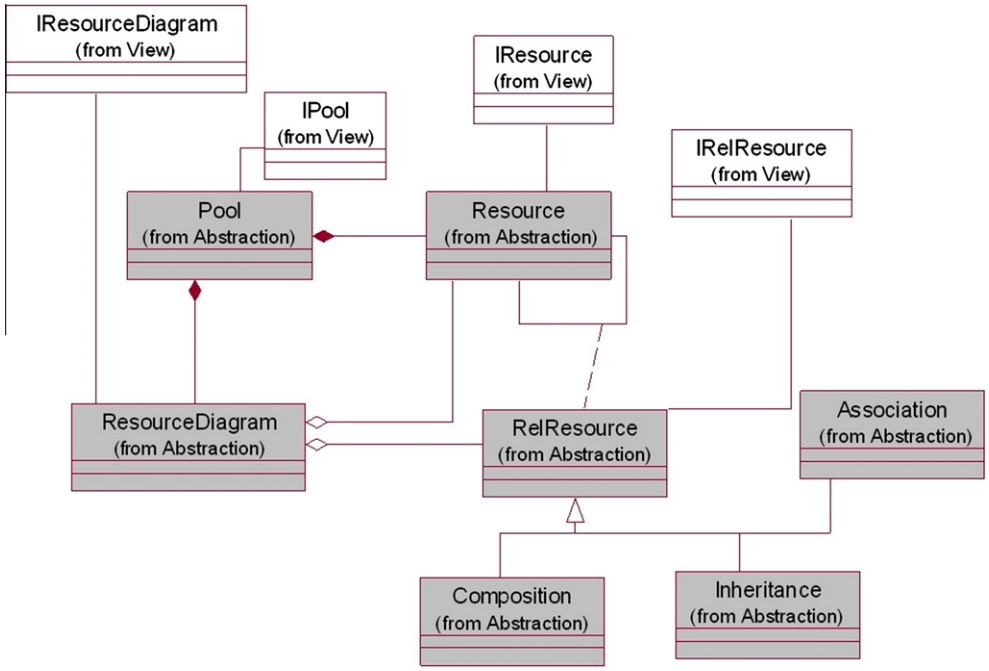


Fig. 5. Domain view class diagram.

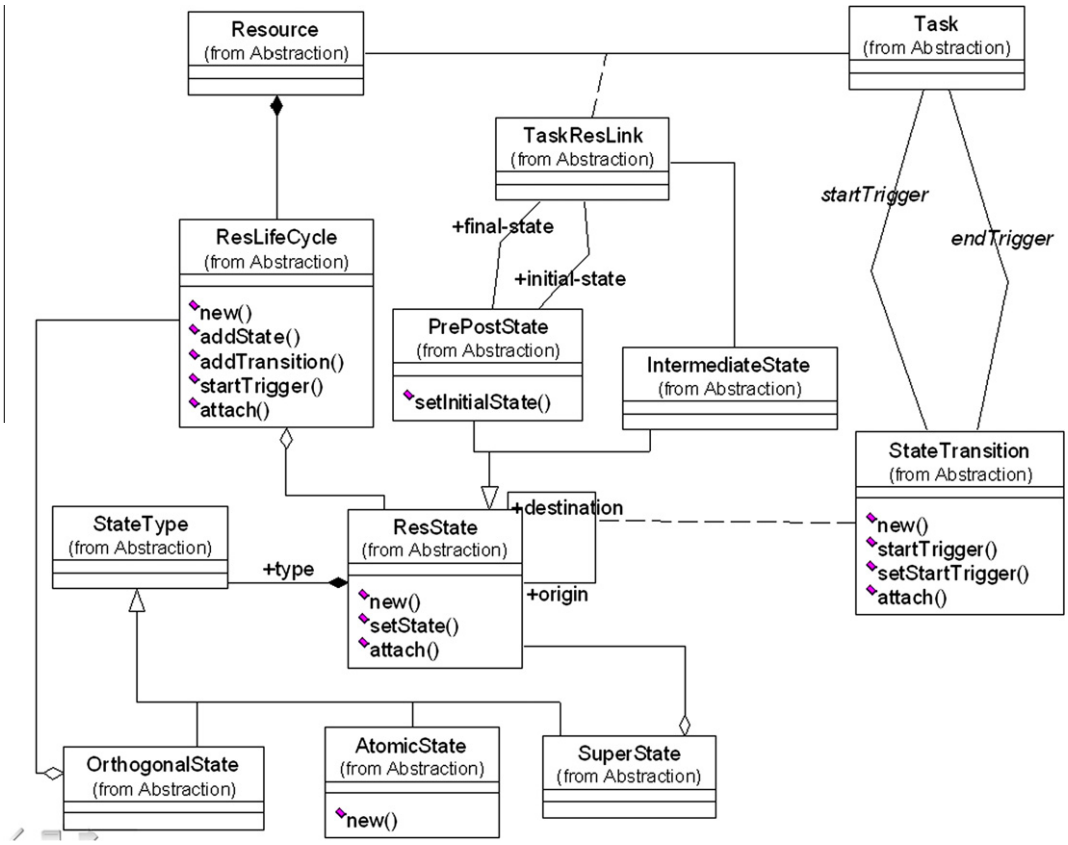


Fig. 6. Dynamic view class diagram.

6.1. Simulation model components

Components are DEVS model. Each one has a well-defined interface through its input and output ports and the values in such ports. These blocks define the vocabulary needed to construct SM. They are: *AtomicTaskDevs*, *ResourceDevs*, *TaskVersionDevs*, *SimulationModel* and *ExperimentalFrame* which are described below. As these components have been defining with a special purpose, it has determined a special message structure that is interchanged among them, which is explained at the end of this section.

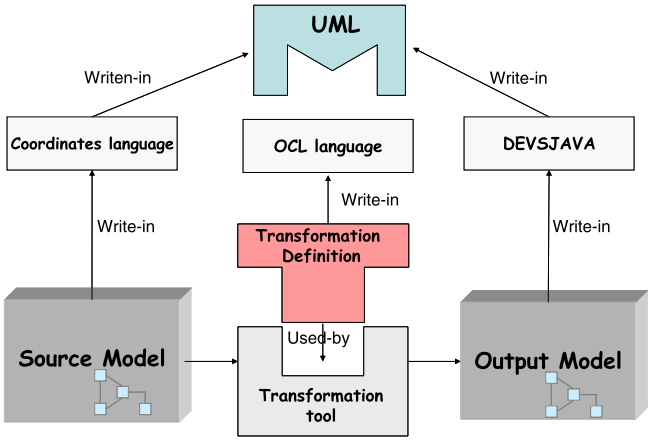


Fig. 7. Transformation schema.

6.1.1. AtomicTaskDevs

An *AtomicTaskDevs* is an atomic DEVS model that encapsulates the behavior of a task in the CM. Fig. 8 is a representation of an *AtomicTaskDevs* model.

An atomic Task *T* belonging CM could be part of two diagrams: Task version diagram and resource state transition diagram (STD).

The dynamic interpretation of *T* must realize when *T* can execute and what are the *T* execution effects. *T* can execute if both the resources are available and the time correspond to the execution of *T* according with the arrangement. In business process the order in which tasks are executed is important and must be taking into account. Then, when the task executes, it causes: (i) changes in resources states and (ii) execution of following tasks. In DEVS model the way to represent this behavior is through inputs and outputs ports and events in that ports. The *er* port and the values in that port *ready*, *no-ready* are interpreted as the resources that are ready or not to participates in task execution. The *ed* port and the values in that port *process*, means that a job arrives and will be put in queue. The *eta/ets* ports and the values in that ports *end, start, ready, no-read,y* when they are received are interpreted as synchronization message of previous tasks. The *stop* port and the value in that port *stop* is interpreted as simulation finalization message. Once the task is executed, it generates output events that will cause changes in models that receive such events. The output events *start/end* means the task start/end execution. The output events *ready, no-ready* are sent by task when it needs to synchronize. The output event *process* sends a job. Finally events *tSusp* and *longCola* stand for time in suspended state and average queue length, respectively.

The states transition diagram in Fig. 9a shows partially the dynamic of *AtomicTaskDevs* model due to message in input ports.

In state *init* the task is ready to execute, immediately the task enters in the state *executing* where it remained until the *execTime* has elapsed. Before the transition from *init* to *executing* state take effect, the task sends out the *start* and *tSusp* events.

Then, when the task finishes, which is task's execution time has elapsed, the task enters in the state *suspended*, before this transition, the task send out the *end* and *longCola* events. The internal transition that occurred from the state *informing* to *waiting* has the purpose to send the *ready* output event in order to synchronize the beginning of a set of tasks.

As regard external transition function, there is a transition from *suspended* to *init* state when there are: (i) events *ready* in *er* input port, and (ii) *end* events in *sta* input port. In some cases, the task needs to synchronize with other tasks to begin to execute, in this case, when the above conditions are met, there is a transition from *suspended* to *informing* state. The task will be in *waiting* state until receiving *ready* synchronization messages. When this happened, the task goes from *waiting* to *init* state. It is possible that, while the task is waiting for synchronization, the resources became unavailable sending *no-ready* event; in this case, the transition from *waiting* to *withoutRes* state is performed. Finally, the transition from *suspWithoutRes* to *informing* is performed when task receive the events *ready* in *er* port, meaning resources become available again for the task.

Diagram in Fig. 9b shows messages in *stop* and *ed* ports because they have special usage: when the *stop* event is present in *stop* port, the task will be in the *suspended* state whatever state it is (show in the figure as *xx* state). When an event is presented in *ed* port, it is placed in the queue and the task remains in the same state.

6.1.2. ResourceDevs

A *ResourceDevs* is an atomic DEVS model that represents the behavior of a resource from the conceptual model. Fig. 10 shows a graphical representation of a *ResourceDevs*. Then, for each resource R in CM, there is a *ResourceDevs* in SM typifying R. This model has input and output port well defined but the internal and external transition function is not determined until the translation process is executed. When a *ResourceDevs* is generated, the states and transition from STD belonging CM are copied and adapted in order to represent the behavior of the *ResourceDevs*. In this way, the behavior of this model and the way in which the process generates the resources model will be explain in Section 6.2.

6.1.3. TaskVersionDevs

TaskVersionDevs represents a particular decomposition of a task with respect to other simpler tasks. It is a coupled DEVS model where its components are *AtomicTaskDevs* and *TaskVersionDevs*

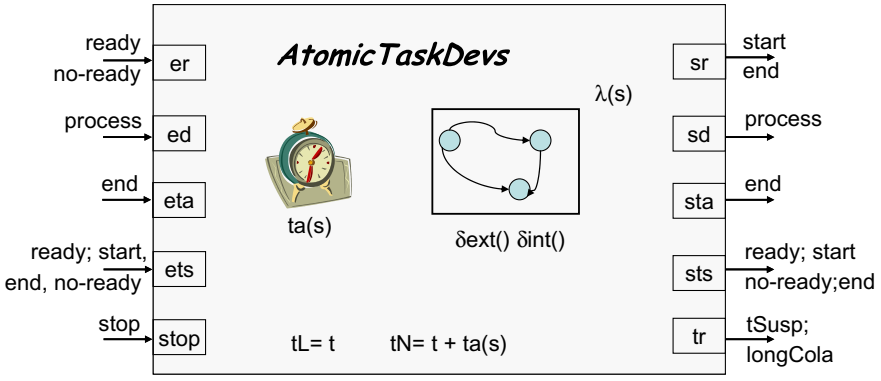


Fig. 8. *AtomicTaskDevs* component.

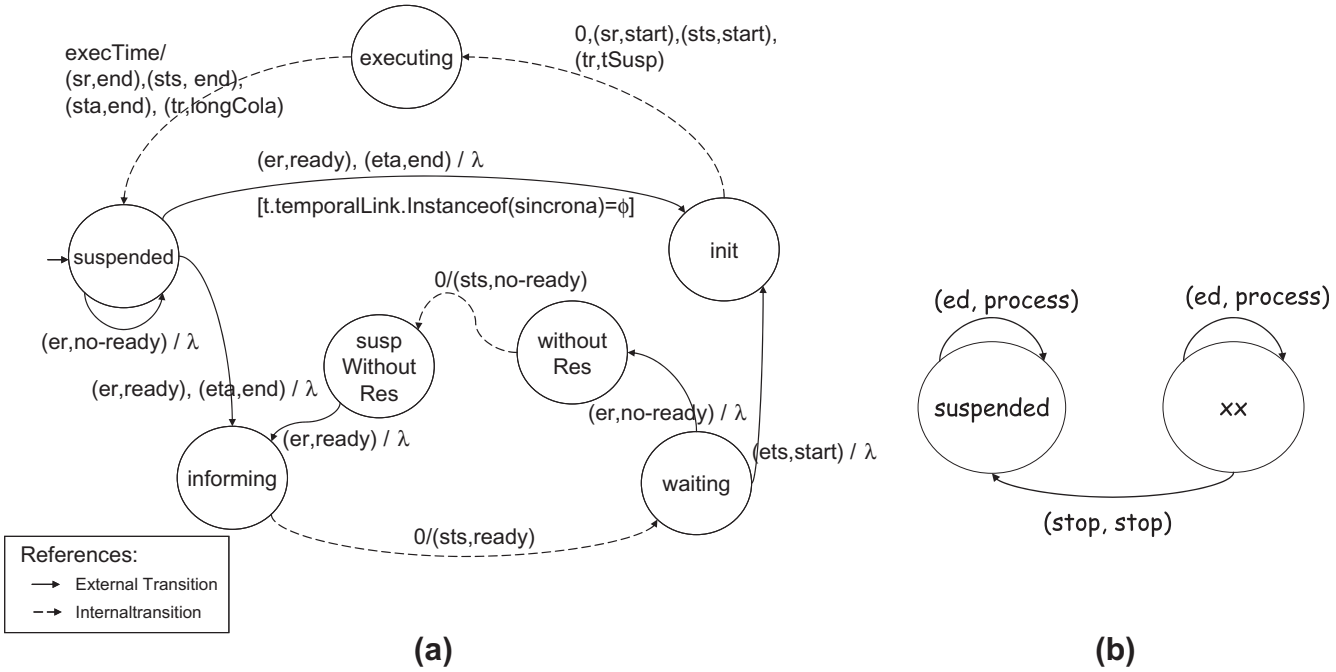


Fig. 9. Internal and external transition function.

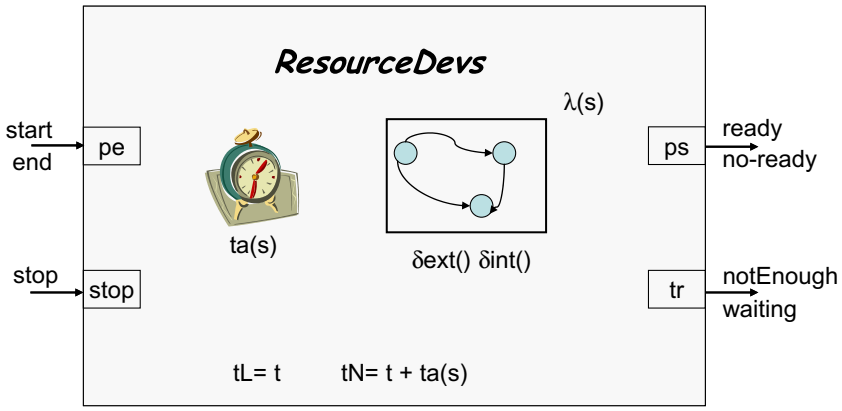


Fig. 10. *ResourceDevs* component.

models, linking by coupled relationship. This model typifies a task version associated to a composite task in CM. The model *TaskVersionDevs* has the same *X* and *Y* set as *AtomicTaskDevs*, it is to say, both are consider having the same interface.

The couplings among component models vary according with the temporal relationship in CM and they are explained in the translation process.

6.1.4. SimulationModel

SimulationModel represents the business process behavior to be simulated. It is a DEVS coupled model. It is possible to connect to this model an experimental frame in order to execute the simulation in a given scenario. Fig. 11 depicts the *SimulationModel* DEVS model.

The input events are those that feed the simulation model. The event *process* has a resource as an argument that represents the resources generated outside the enterprise process simulated. These types of resources are for example customer order, raw material, among others. In CM this type of resources can be distinguished through the *processes* relationship with tasks. As regards *stop* event, it sets the end of simulation.

The outputs events from *tr* port are those generated by simulation process. These events represent statistical value that will be interpreted by *ExperimentalFrame*. As we can note, these values are outputs event of *AtomicTaskDevs*, *TaskVersionDevs* and *ResourceDevs* building blocks explain previously. The *process* events in *out* port will be taking into account only if the simulation model is part of a federate. That is, these events are those generated by the internal business process that will be the input events to external processes.

6.1.5. ExperimentalFrame

ExperimentalFrame model represents the experiment with which to prove the simulation model. The experimental frame is defined by Zeigler in [52], it can adopt different forms according with the designer goals. In this architecture, the *ExperimentalFrame* model has been defined with three components: *Acceptor*, *Generator* and *Resume* models. Fig. 12 shows the structure of *ExperimentalFrame* model. Their components are described follow.

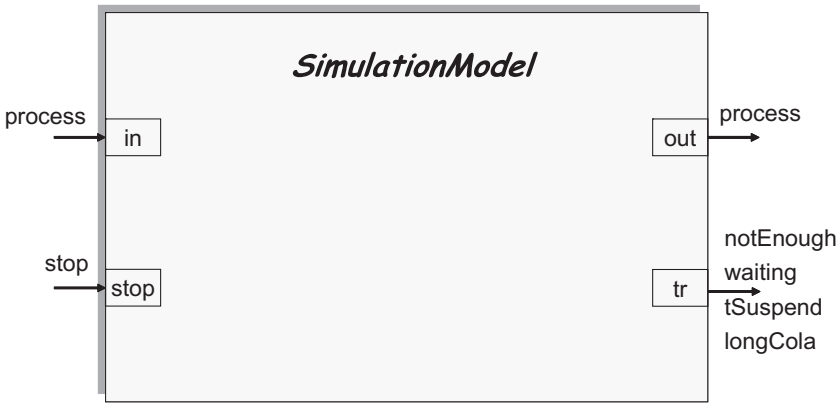


Fig. 11. *SimulationModel* component.

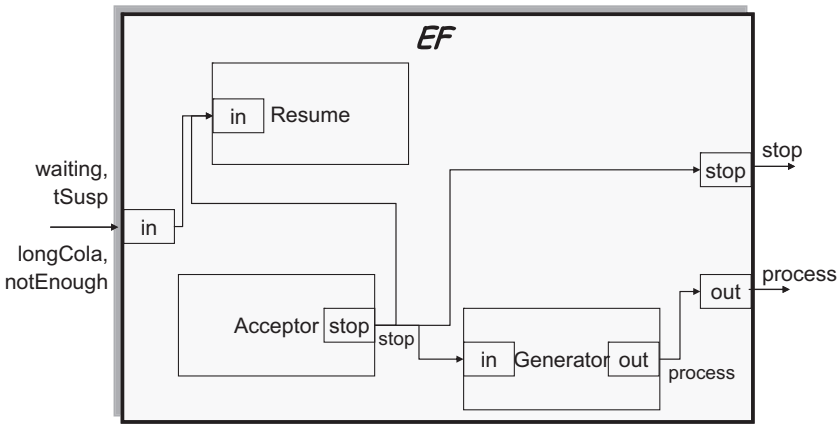


Fig. 12. *ExperimentalFrame* component.

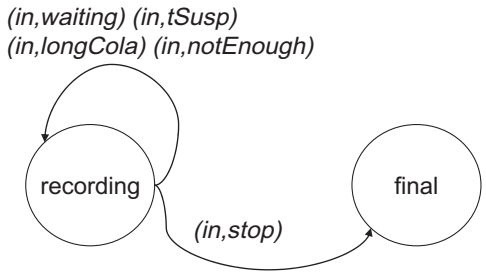


Fig. 13. Resume model state transition.

6.1.6. Resume

Resume is an atomic DEVS model, which is in charge to collect the statistical values, such as the time in which a resource has waiting in a queue (*waiting* event), time in which a task has been suspended (*tSusp* event), among others. This model has only external transition function. We can summarize the behavior of *Resume* model with the state transition diagram on Fig. 13.

Resume model is in state *recording* for an infinite time, when receives an event *waiting*, *tSusp*, *longCola* or *notEnough* in *in* port, then record the event and remands in state *recording* waiting the new event. When receive the event *stop*, go to the state *final*. This model has not output and generates the following metrics from recorded values during simulation run:

TI_i is the average of task i inactivity time (Eq. (6.1)) and LC_i is the average of queue length (Eq. (6.2)).

$$TI_i = \frac{\sum_{j=0}^{n} tSusp_{i,j}}{TotalSimulationTime} \tag{6.1}$$

$$LC_i = \frac{\sum_{i=0}^{n} longCola_i}{n} \tag{6.2}$$

TIP is the process inactivity time (Eq. (6.3)):

$$TIP = \sum_{i=1}^{n} TI_i \tag{6.3}$$

For each resource i:

TC_i is the average of the time in queue (Eq. (6.4)) and TF_i is the average time in which the resource i was not enough for a given task (Eq. (6.5)).

$$TC_i = \frac{\sum_{i=0}^{n} waiting_i}{totalSimulationTime} \tag{6.4}$$

$$TF_i = \frac{\sum_{i=0}^{n} notEnough_i}{totalSimulationTime} \tag{6.5}$$

6.1.7. Generator

Generator is an atomic DEVS model, which is in charge to generate events that feed simulation model. These events represent resources generated externally to the process that is simulated. For instance, *customer orders*, *production plan* and *payment* can be considered as events that make processes react. These resources are identifying in the CM as the resources that are processes by tasks, but there is not task that generate those. *Generator* makes events

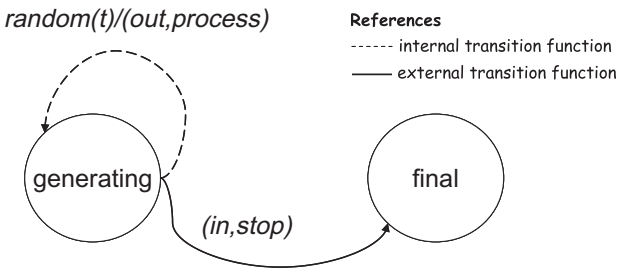


Fig. 14. Generator state transition diagram.

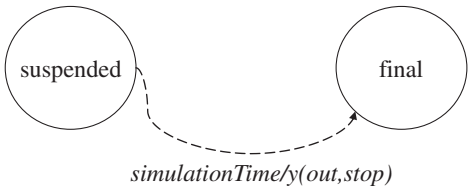


Fig. 15. Acceptor state transition diagram.

until receives the event *stop* in its *in* input port. We can summarize its behavior with the state transition diagram shown in Fig. 14.

There is an internal transition from the state *generating* to the same state, which makes the output event *process* appear in *out* output port. This transition takes effect when the elapsed time is equal to *random (t)*, this value is a random number that determine the time in which the events are generated.

The external transition function represents the presence of the event *stop* in *in* input port. When this happened, the model goes to the *final* state.

6.1.8. Acceptor

Acceptor is an atomic DEVS model in charge to determine the end of the simulation run. It has one output port but not input port. Initially it is in the state *suspended*, remaining in this state during the simulation time. When this time elapses, it goes to the *final* state generating the event *stop* on the *out* output port.

Fig. 15 shows the state transition diagram that represents the behavior of this model.

As we can see in Fig. 15 it has only one internal transition and one output. It has not external transition.

These components have been implementing as it shown in the class diagram of Fig. 16. As we can note in the previous diagram, the building blocks have been implementing as subclasses of classes belonging DEVSJAVA framework (filled boxes).

6.1.9. Events

In this approach, the events interchanged among components are implemented in the class *MessageCont*. It has two components: *event* that represents the event interchanged as for example *process*, *end*, *start* among others; and *list* that is a set of components representing the event argument. For instance, the *process* event has a *ResourceDevs* instance as argument; the *ready* event from *ResourceDevs* component has a list of *AtomicTaskDevs* instances, which represents the tasks for which the resource is ready. All events from *AtomicTaskDevs* have itself references, meaning which is the task that sends the event.

6.2. Transformation process

Rules that govern transformation process (see Appendix A) determine in which way a CM element is translated in a component of SM but they do not specify the order in which they must execute. In MDA approach implemented in this environment, there is an internal scheduling defining the order in which individual rules are applied [14]. Because CM has a hierarchical organization, transformation process is executed in top-down way from the highest level CM component to the inner most one. Fig. 17a shows the sequences of rules performed by translator. Rule 1 and 2 are in charge to generate the framework of simulation model (*topModel* and its components: *SM* and *ef*) and the other rules are in charge to generate its components iterating on CM components,

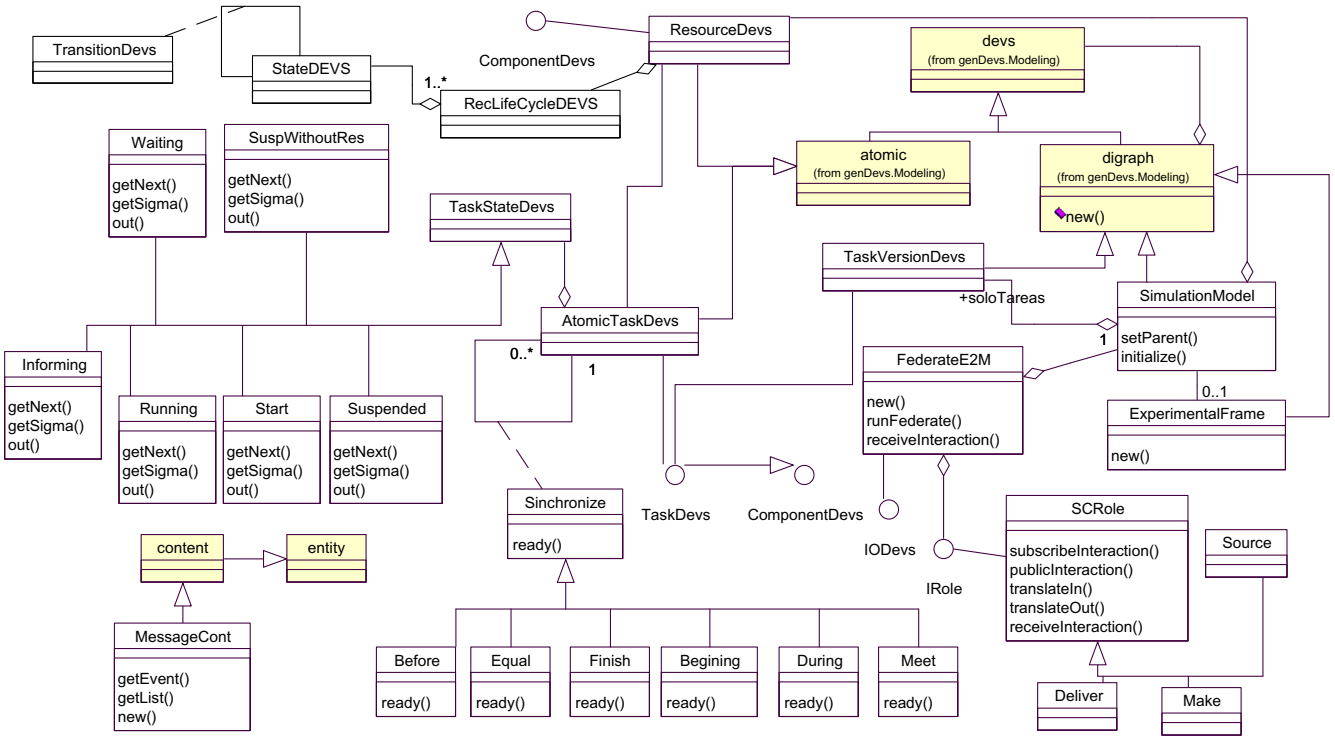


Fig. 16. Class diagram of SM components implementation.

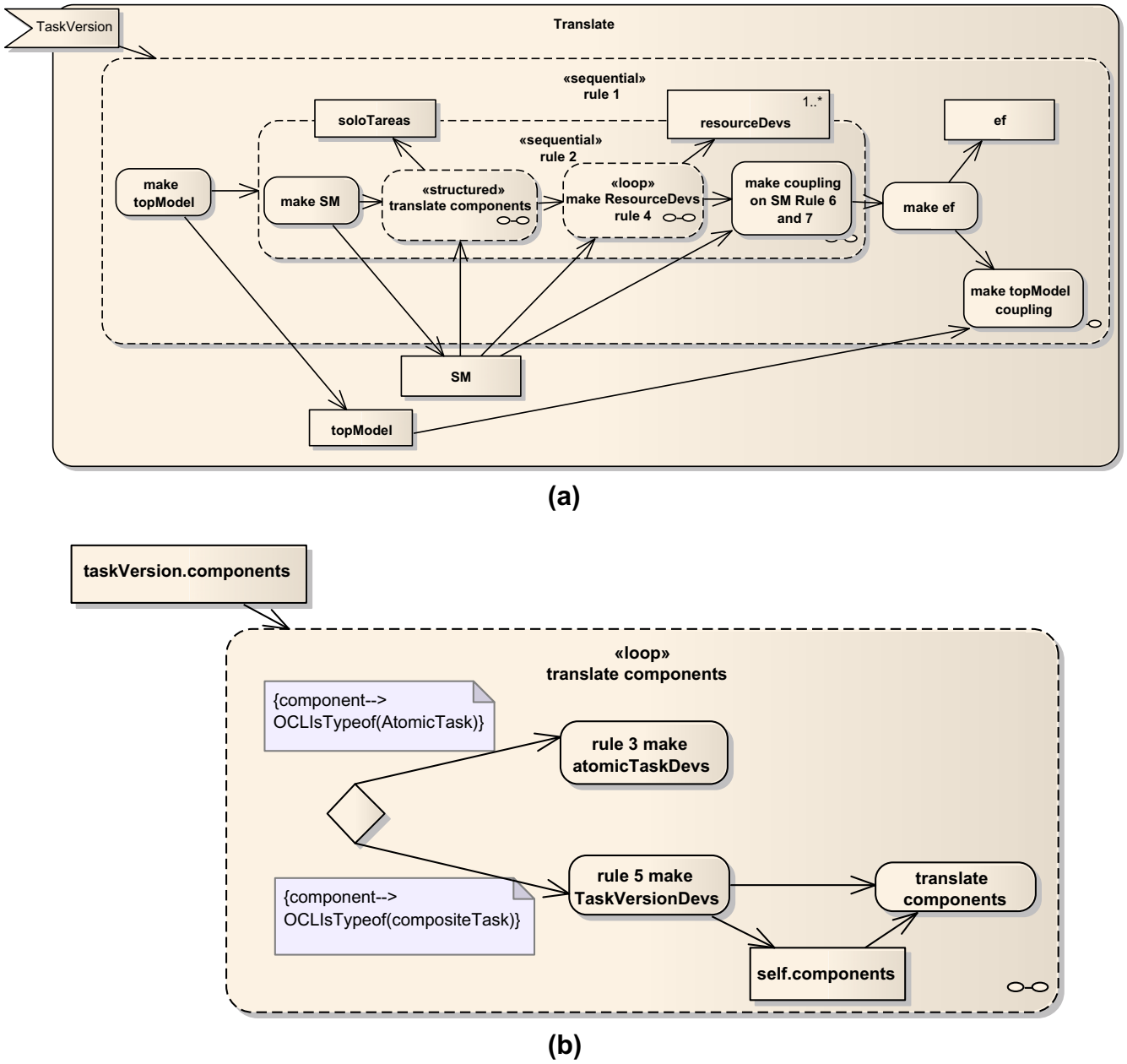


Fig. 17. Translation process sequence.

preserving the hierarchy. The activity *Translate components* is a complex activity, that iterates on task version components. It is spread out in Fig. 17b and involve calls to rules 3 and 5. Task Version components are both tasks and resources, but this activity only considers task to translate. If the task is an atomic task the translation is simple, but if the task is a composite task, the *Translate components* activity is recursively called with its components as shown in Fig. 17b. The activity *make ResourceDevs* is a loop that iterates on resources instances, which are found in different depth levels, in CM. Once the models were generated, coupling among them are carried out. Rules 6 and 7 define coupling due to task-resource relationship and temporal relationship.

Fig. 18 shows the implementation of the translation process in translator tool to construct SM from a task version in CM. The process begin when *Translator* receive *translate (selectedProcess)* message, whose argument is a *TaskVersion* from task view. Sequence diagram shows the way in which models are generated and compounds each other. Tasks can use the same instance of resources and they can compete for these. Then, in the construction of SM, on the one hand it has represented the decomposition of tasks in subtasks (*soloTareas* instance) and on the other hand it has represented the different instances of resources that participate in the selected process (*generateResDevs*() method).

This description is not fully detailed, but represents the steps necessary to get higher level model and correspond to the implementation of rule 1 and 2. Fig. 19 shows the schema of the *topModel* digraph generated with this sequence. We can see the two models components *SM* and *ef* and how they are coupling.

Every time that the transformation process takes effect, the structure of *topModel* and *ef* will be the same, but not so with the *SM* model, its structure depends on the CM structure. One of the *SM* components is *soloTareas*, which is an instance of *TaskVersionDevs* class and represents the higher level task decomposition.

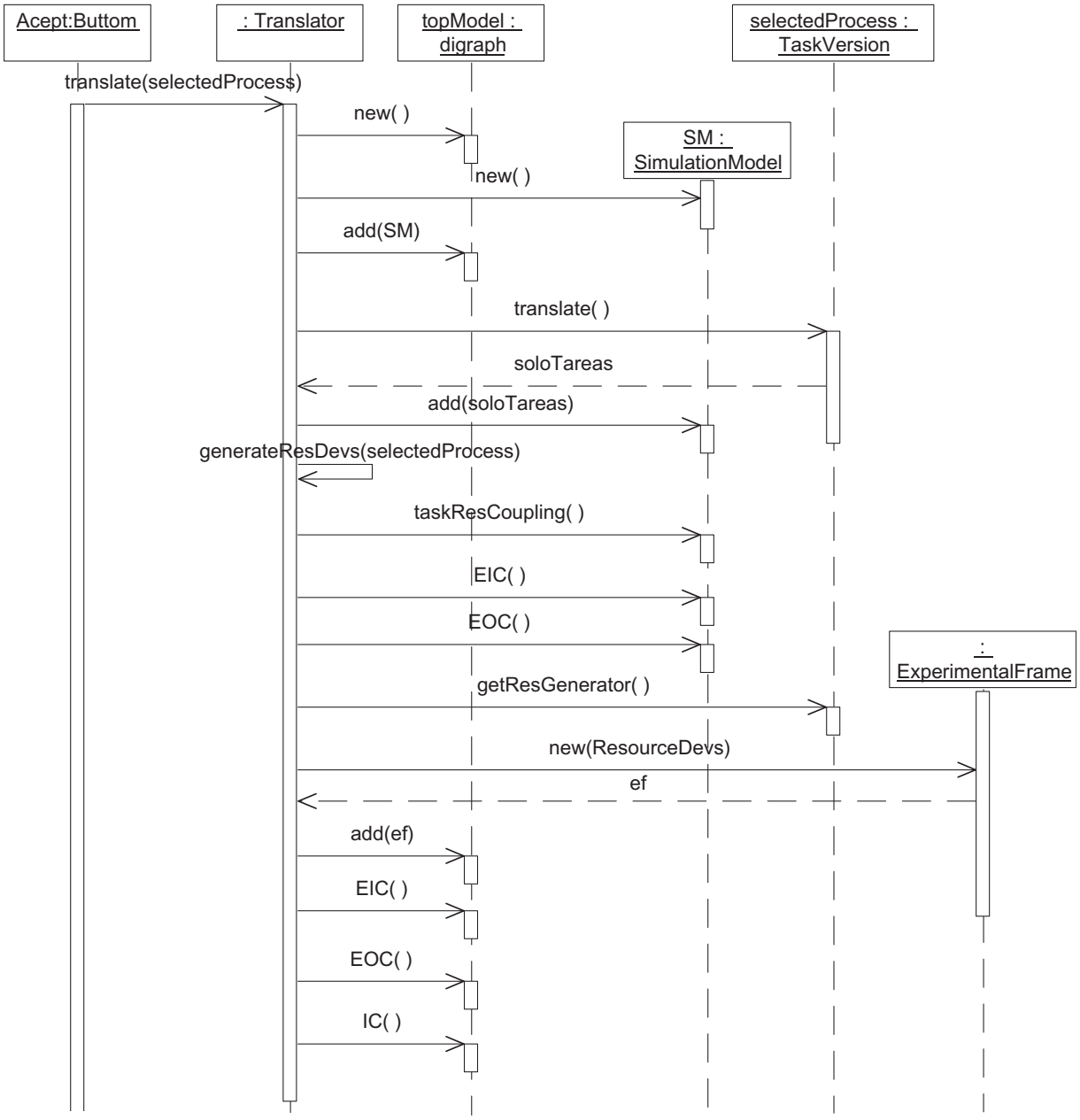


Fig. 18. *Translate* sequence diagram.

Fig. 20 shows the sequence diagram to translate the task hierarchy, and corresponds to the implementation of rule 3 (for atomic task) and 5 (for composite task).

A task that it is translated can have associated both, an *AtomicTask*, in which case it is instantiated *AtomicTaskDevs*, or a *CompositeTask*, in which case it is instantiated *TaskVersionDevs*. In the latter case, its components will be translated calling this method recursively.

Let's consider the task version shows in Fig. 21a, which represents the decomposition of a task called "*making yerba sapecada*". The transformation process generates an instance of *TaskVersionDevs* coupled model that represent the higher level task "*making yerba sapecada*", refer this DEVS model as *V*. Then, for each task that participate in task decomposition, it is generated the corresponding DEVS model. In this example the three tasks, which are part of the decomposition, are atomic tasks. Then each one has an instance of *AtomicTaskDevs* associated, which are part of *V* as we can see in Fig. 21b. Couplings within *TaskVersionDevs* model is complex enough because it realize the arrangement of task in CM through temporal relationship.

Rule 6 describes the *TaskVersionDevs* IC base on temporal relationship. Then, whichever two task associated with an asynchronous temporal relationship (*before*, *during)* it will be generated the coupling between origin task *sta* port and destination task *eta* port. In Fig. 21a we can note that there is a temporal relationship *before* (in the graph appeared as "antes que") between *"dry yerba mate"* and "*burn yerba*" tasks, this relation makes the coupling between *sta* port of *"dry yerba mate"* DEVS model and *eta* port of "*burn yerba*" DEVS model as shown in Fig. 21b.

In the same way, whichever two tasks associated with a synchronous temporal relationship (*meet, equals, begin*), it will be generated the coupling between origin task *sts* port and destination task *ets* port.

The coupling between *ed* port and *sd* port is defined in rule 7 and is associated with the occurrence of *processes* relationship between a *task* and a *resource*.

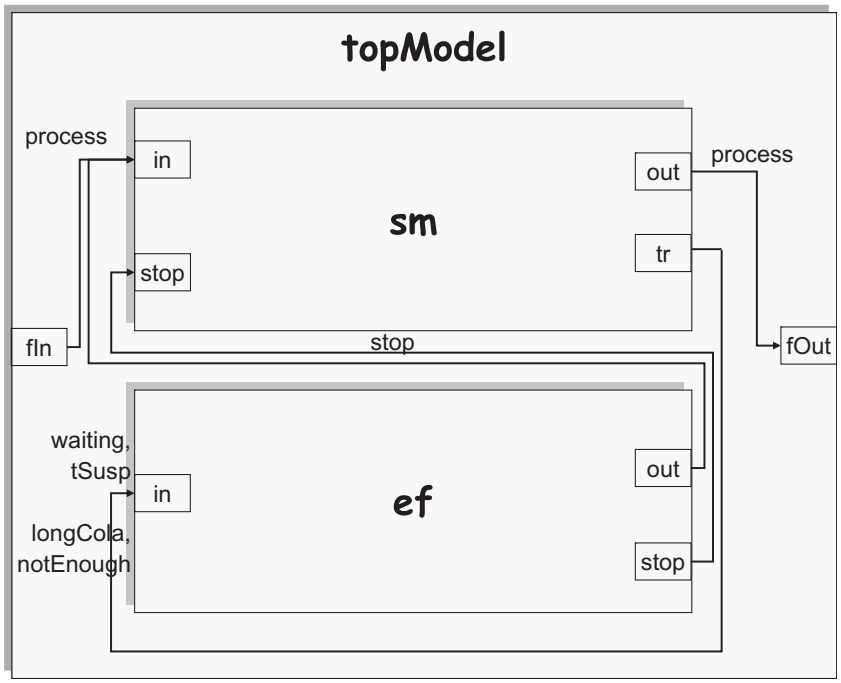


Fig. 19. Structure of obtained simulation model.

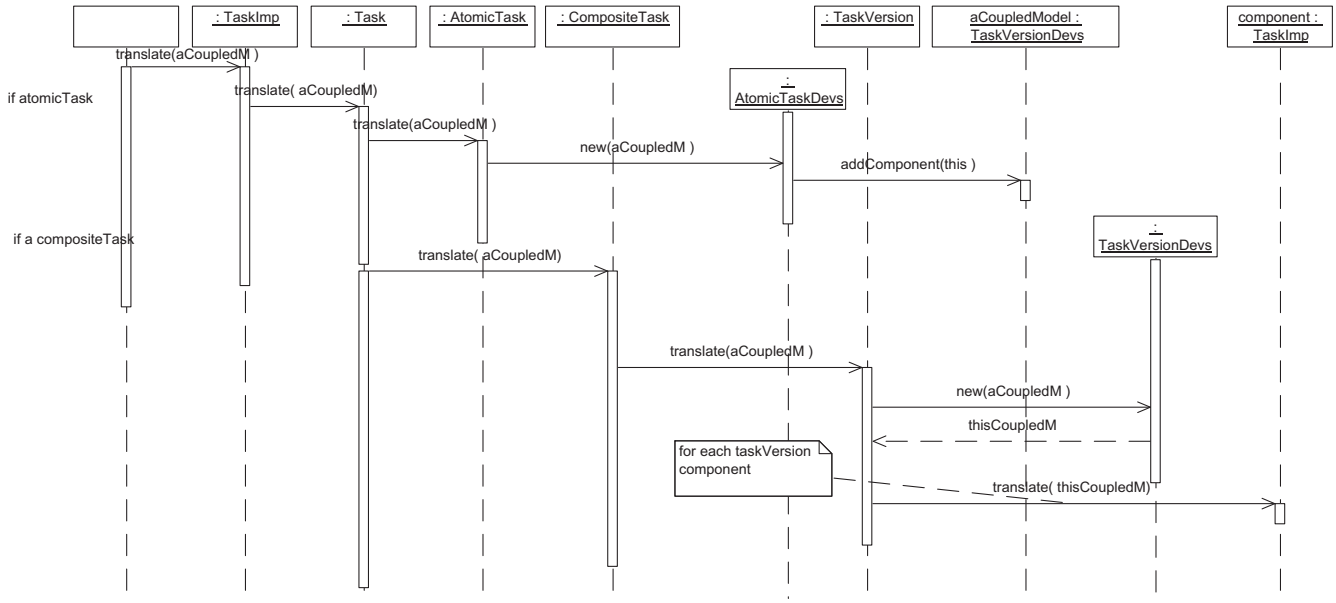


Fig. 20. Task translation.

As regard EIC, it is necessary determine which component model is the first to execute. A model t, which is a component of a coupled model, is the first to execute if there is not a pair in internal coupling with value equal to (t,eta). Let's $\{M_i\}_V$ be the V models components, then the set *firstTask* $\subseteq \{M_i\}_V$ is defined as follow:

$$\mathit{firstTask} = \{t/t \in \{M_i\}_V, \forall t_j \in \{M_i\}_V, ((t_j, \mathrm{sta}), (t, \mathrm{eta})) \notin \mathrm{IC}_V\} \tag{6.6}$$

Since the temporal relationship between tasks in the conceptual model, determine couplings between DEVS models, the set *firstTask* depict those DEVS models that typify a task t that have not predecessor in the conceptual model, that is, there are not tasks with temporal link *before* with task t.

With this definition, it is possible determine same pair in the EIC of the V coupled model as follow:

$$\forall t \in \mathit{firstTask},\ \{((V, \mathrm{ets}); (t, \mathrm{ets})), ((V, \mathrm{eta}), (t, \mathrm{eta}))\} \subseteq \mathrm{EIC}_V \tag{6.7}$$

With the expression (6.7) the EIC_V is not completely defined. The *stop* input port of model V is connected with the *stop* input port of each component t, then:

$$\forall t \in \{M_i\}_V, ((V, \mathrm{stop}); (t, \mathrm{stop})) \in \mathrm{EIC}_V \tag{6.8}$$

We can note in Fig. 21a that the first task is *Dry yerba mate* task, in Fig. 21b this is reflected in coupling ((*V, ets); (dry yerba mate, ets)*)and ((*V, eta); (dry yerba mate, eta)*). As well in Fig. 21b the *stop* port of V is connected with *stop* port of all its components.

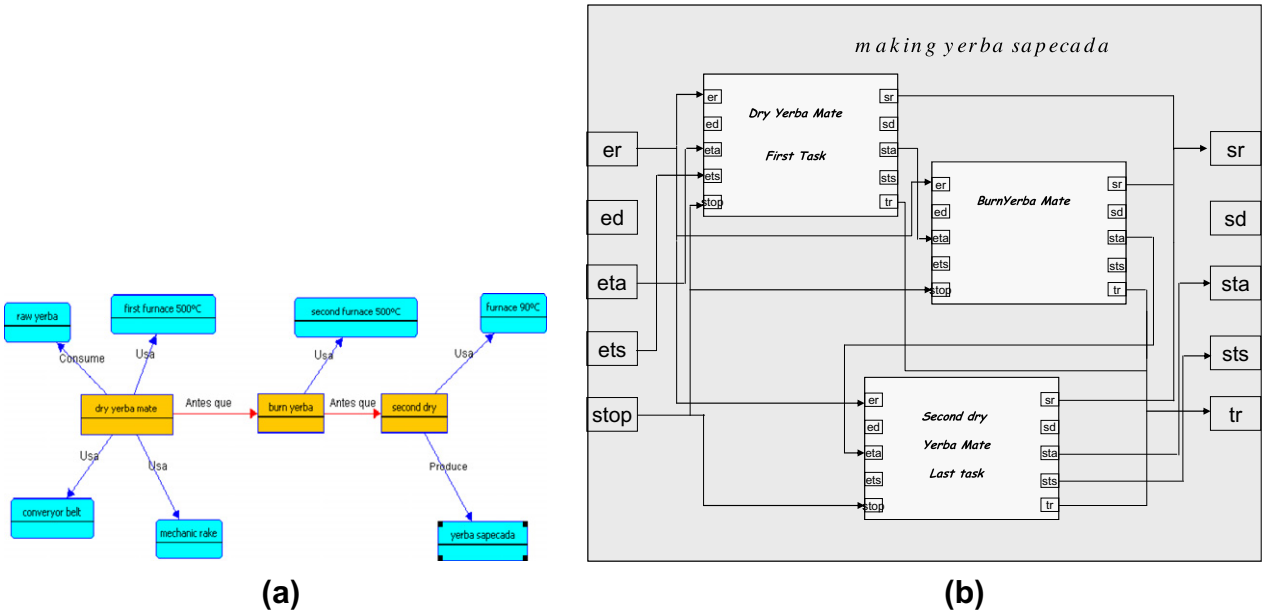
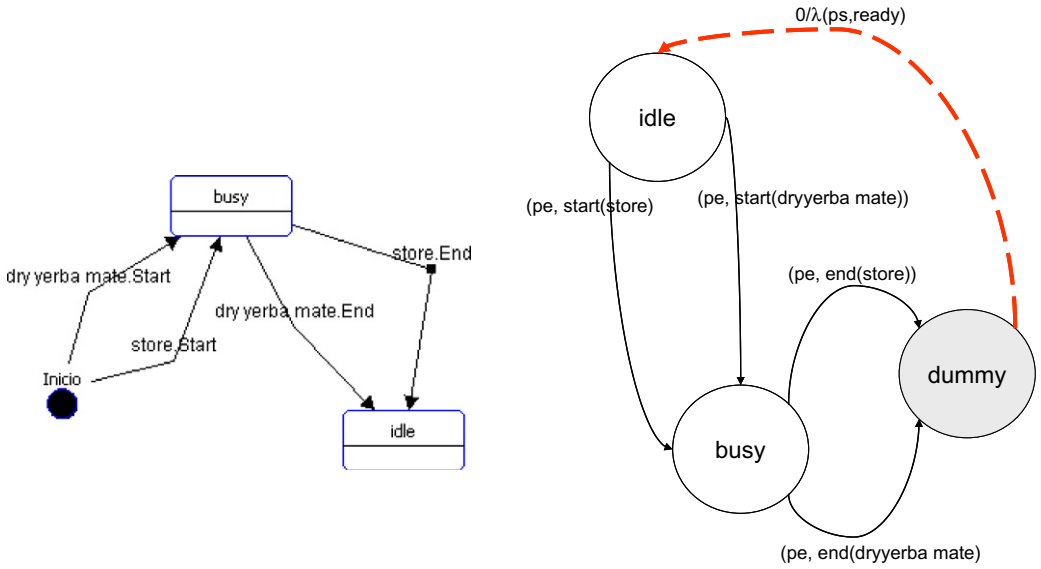


Fig. 21. Transformation of *TaskVersion* to *TaskVersionDevs*.



(a) *resource STD from CM* **(b)** *ResourceDevs transition functions*

Fig. 22. Mechanic rake STD.

In order to define EOC_V coupling, it is necessary determine which model is the latest to execute. A model $t \in \{M_i\}$ is the latest to execute if there is not a pair in internal coupling with value equal to (t, sta_t). We can define the set *finalTask* of models that meet this condition with the expression 6.9:

$$finalTask = \{t/t \in \{M_i\}_V \wedge \forall t_j \in \{M_i\}_V, ((t, \text{sta}), (t_j, \text{eta})) \notin IC_V\} \tag{6.9}$$

As for the *firstTask*, we can reason in the same way thinking that the model t typifies a task T in conceptual model, is the latest model to execute in a version V if t is not the origin of a temporal relationship *before* with other task.

Then, the expression 6.10 defines the pairs that are included in EOC_V:

$$\forall t \in finalTask,\ \{((t, \text{sts}), (V, \text{sts})), ((t, \text{sta}), (V, \text{sta}))\} \subseteq EOC_V \tag{6.10}$$

Other pairs that are part of EOC_V are couplings between ports *tr*. The expression (6.11) defines these couplings.

$$\forall t \in \{M_i\}_V,\ ((t, \text{tr}), (V, \text{tr})) \in EOC_V \tag{6.11}$$

Note that in the transformation of the version in Fig. 21, the resources are not taking into account. The *TaskVersionDevs* generated has only *TaskDevs* model as components, either *atomicTaskDevs or TaskVersionDevs*. The resources translation will generate instances of *ResourceDevs*, which are part of *SM* components.

In the translation of a resource (described by rule 4), it is taking into account the STD associated of each resource. STD shows in which way the tasks make a resource goes from one state to

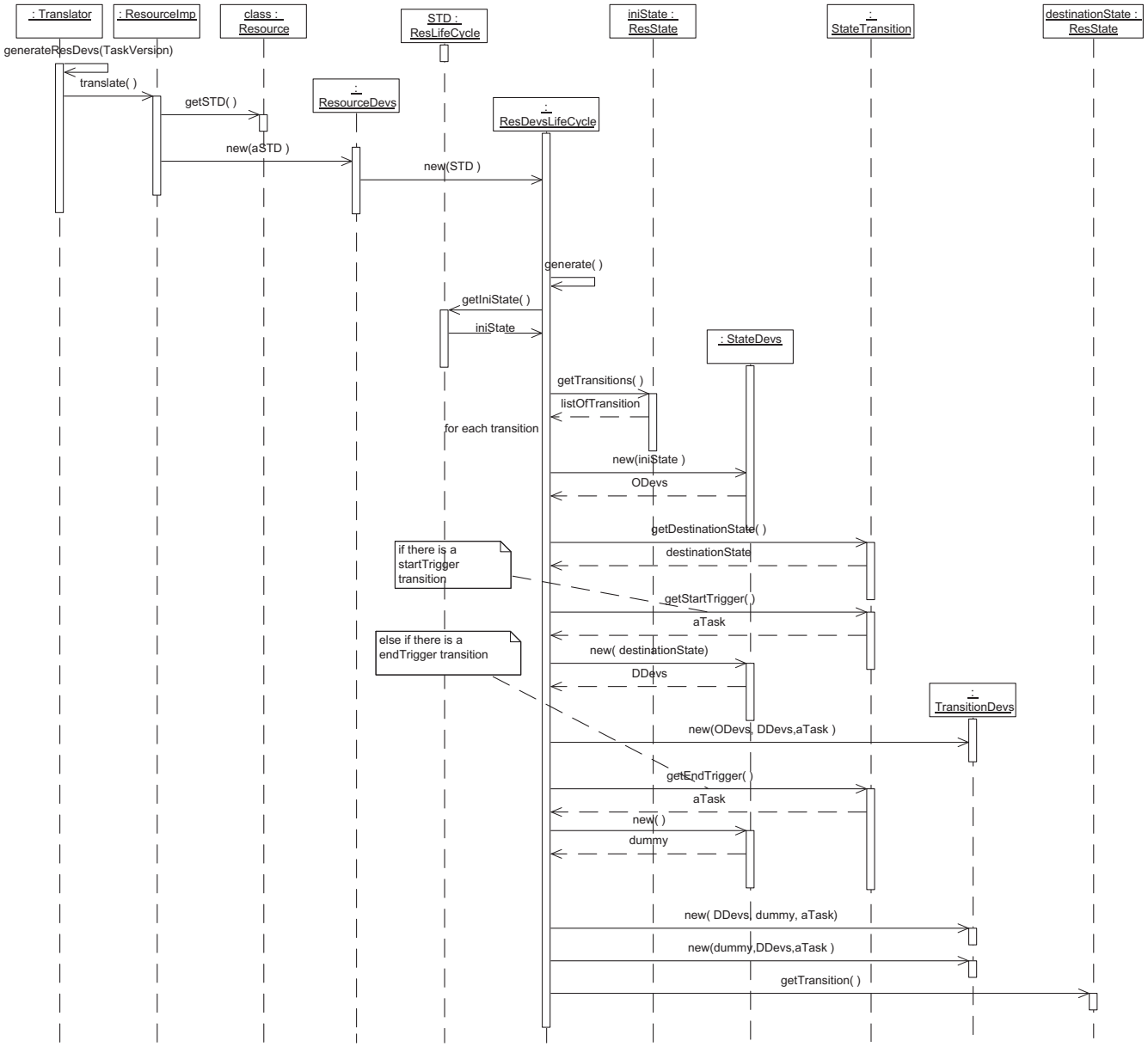


Fig. 23. Sequence diagram to translate resource.

another. Coordinate language sets to represent the evolution of a resource giving a task execution is necessary to represent three states: initial state, representing the pre-condition; intermediate state, representing the state in which the resource is during task execution; and final state, representing the post-condition. Two events makes resources go from one state to another: start trigger and end trigger of the task. Fig. 22a shows an example of mechanic rake STD.

Idle state is both a pre and a post-condition state and *busy* is an intermediate state. This resource is use by two task: *dry yerba mate* and *store*. To translate a resource, the sequence diagram of Fig. 23 shows the steps to follow.

First an instance of *ResourceDevs* is generated but this model does not have transition functions defined. The *ResDevsLifeCycle* instance describes the internal and external transition function and is generated based on resource's STD in CM. This sequence makes the STD shows in Fig. 22a will be translated in the transition function of *ResourceDevs* model shown in Fig. 22b. As we can note, the *startTrigger* and *endTrigger* events are translated as *start* and *end* event in *ResourceDevs* input port, the arguments shown as the event *end* and *start* refer to the special usage in this approach of the *messageCont* class. A new state *dummy* and an internal transition are added in order to send the event *ready* meaning the resource is ready for a given task. Note that the post-condition state of a task could be the pre-condition state for another task.

Here, we have explained the coupling within *topModel* and *ef* coupled models. Still have to explain coupling within *SM* and *soloTareas* coupled models. Fig. 24 shows the structure of *SM* and the IC, EOC and EIC. It shows only two resource models but could be more than two. The internal coupling implements the relationship between task and resources. The resources *ps* output ports will be connected with *soloTareas er* input port. The *soloTareas sr* output port will be connected with resources *pe* input ports. Through these couplings, the events generated by resources will be propagated to *soloTareas* components. The *SM in* input port is connected with *soloTareas ed* input port, thus the events from the *ef* model are propagated within *soloTareas* model. As regards

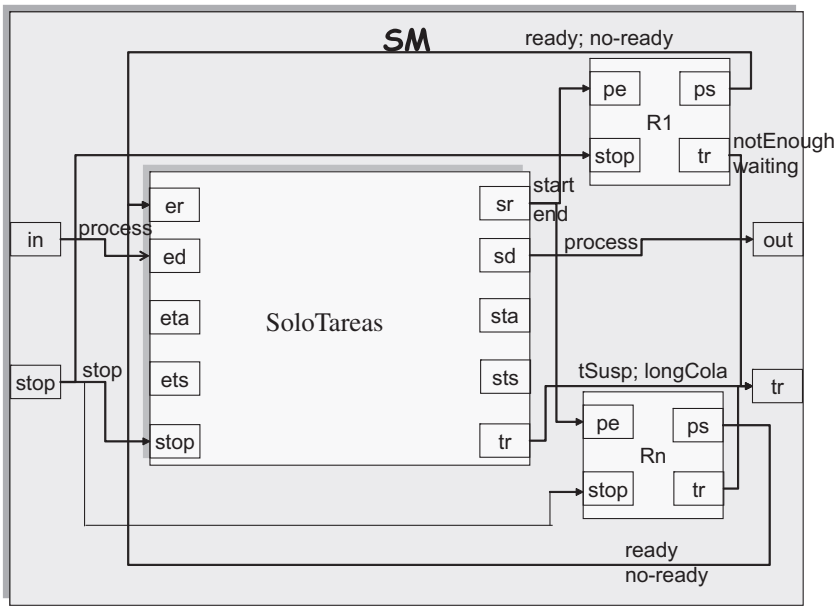


Fig. 24. *SM* coupled model.

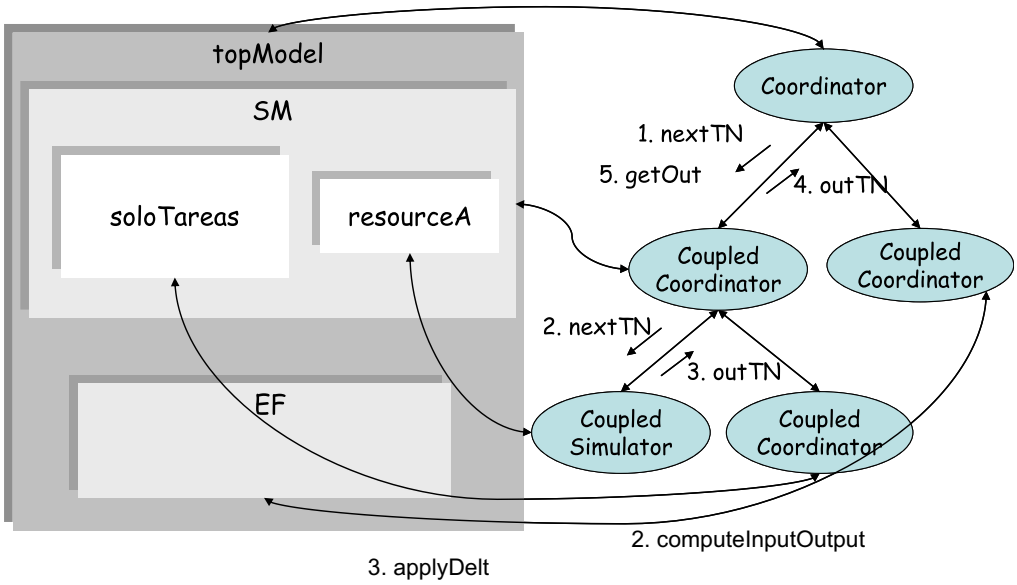


Fig. 25. Execution of SM in local environment.

SM stop port, it is connected with all the components *stop* ports, in this way, when the *stop* event is generated it is propagated within *SM*.

7. Executing SM

Once the translation process finish and SM is acquired, it is possible to execute it both locally and distributed. In local environment, the *coordinator* is in charge of directing the simulation run. And in distributed environment, the *CoordinatorE2M* does it.

Fig. 25 shows the SM structure and the engines associated in order to execute in local environment. This represents a normal execution of a DEVS models.

In order to execute SM in distributed environment, we have defined the model *FederateE2M* that represents the HLA-compliant federate. It can run under HLA and interact with RTI.

The class diagram shown in Fig. 26 defines the class *FederateE2M*. It is made up 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 redefines the *simulate* method in order to incorporate modifications in the DEVS simulation cycle and makes root coordinator been a conservative one. 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 [19]. Using this scheme, the RTI and the federate can guarantee no send an event in the past.

Fig. 27 shows a collaboration diagram depicting the interaction carry out among models, engines and controllers to time management.

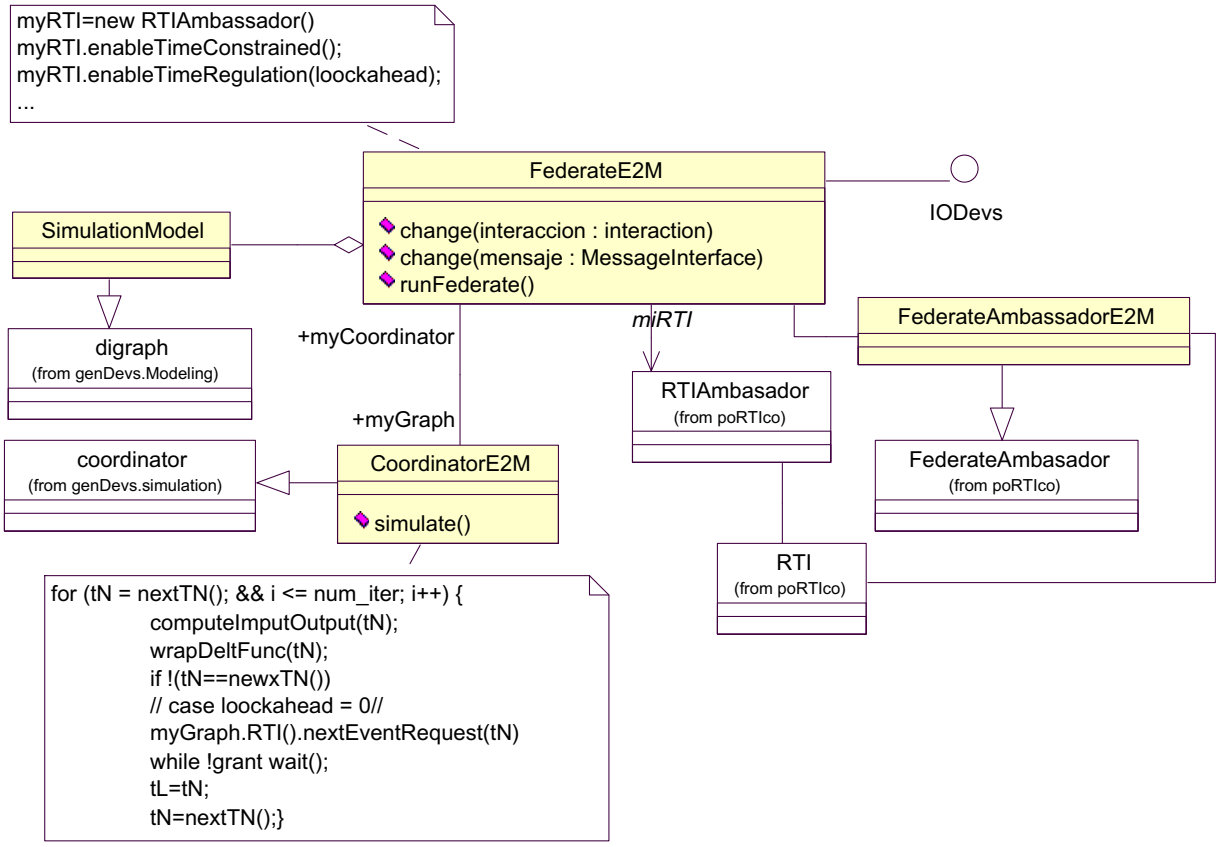


Fig. 26. FederateE2M class diagram.

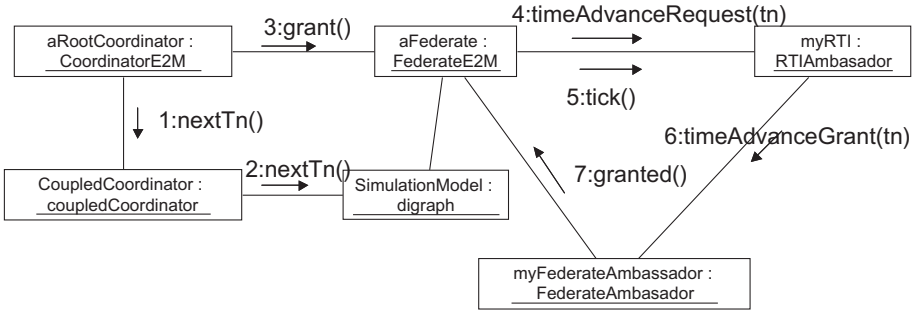


Fig. 27. Time management in distributed environment.

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 other federates and it works with *RTIAmbassador* and *FederateAmbassador* [35] 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. 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, returns the grant to the *CoordinatorE2M*. Finally, the root coordinator follows the simulation cycle.

8. Case study

Let's us consider a *yerba mate* factory. *Yerba mate* is a product used in Argentina to prepare an herbal infusion known as "mate". The factory sells about 8 millions kilos of this product annually in the country and it wants to improve its production process with the aim of having a better performance and, at the same time, improving the sales mechanism. This factory participates in a supply chain with retailers and wholesalers.

To use the environment, the first step is to develop the CM of the factory. Note that the environment has its user interface in Spanish language. The diagram in Fig. 28 shows the *yerba mate* production process with a set of tasks, resources and links. It is important to note that in this example we have only described the production process but in the conceptual model there are others processes defined such as supply process, sales process and logistic process.

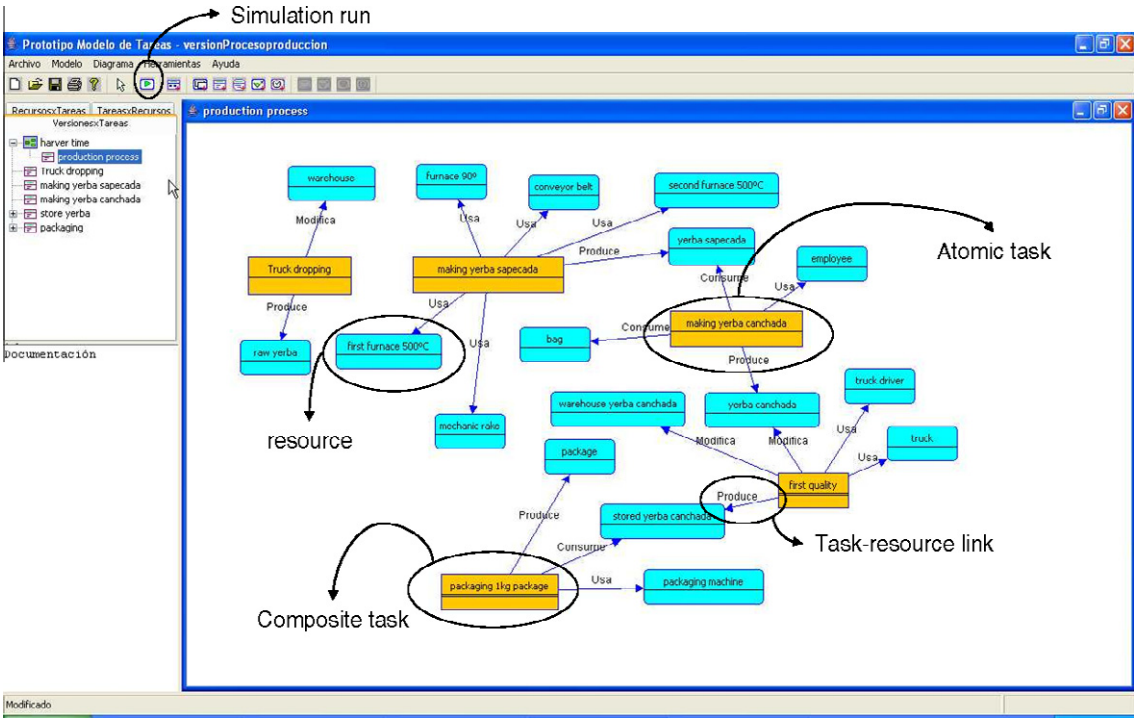


Fig. 28. Yerba mate production process snapshot.

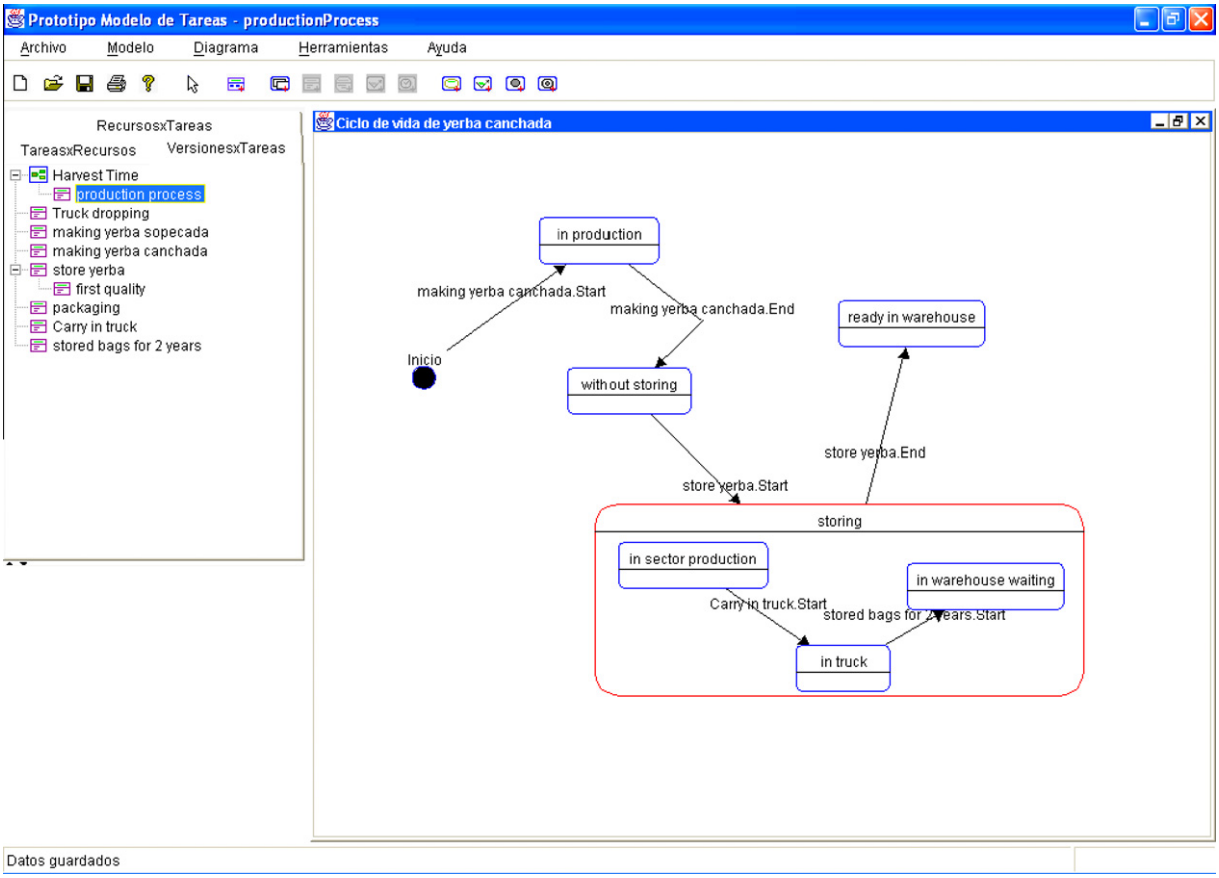


Fig. 29. *Yerba Canchada* STD.

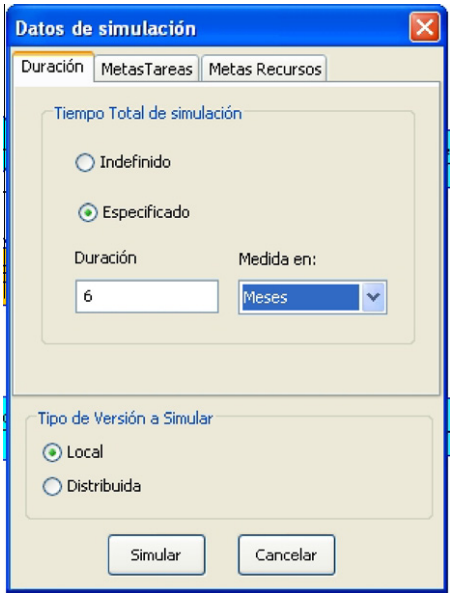


Fig. 30. Snapshot to set the simulation parameters.

The production process has been defined as five tasks: *truck dropping, making yerba sapecada, making yerba canchada, storing yerba* and *packaging*. In this process, the tasks can be executed in parallel, where the only restriction is the availability of resources. Therefore, there are not temporal links between tasks. Each task can be described in a more detailed way using a task version where the task is broken down into subtasks, given a low level view of the process. For instance, there are two ways of accomplishing the *storing yerba task*: (i) to store yerba for two years in warehouses labeled from 1 to 10, or (ii) to store yerba for one year in warehouses labeled 11 to 20. Two task versions are attached to the *storing yerba* task, each one depicting a different way of performing it. In Fig. 28 this task appeared with the first version call *first quality* because this version produces the best quality of *yerba*. In diagram it is possible distinguish an atomic task (rectangles with a single line) from a composite task (rectangles with a double line).

A dynamic view of the conceptual enterprise model is depicted by state transition diagram, one for each resource, which represents the behavior of it. This view shows another process perspective. A *yerba canchada* STD is shown in Fig. 29.

In Fig. 29, is possible to see the states of a resource and the transitions cause by tasks execution. For example when the *making yerba canchada* task starts, the *yerba canchada* resource stays in the *inProduction* state and when this task finishes, the resource goes to the *withoutStoring* state (waiting to be stored). In this way, we must define the STD for each resource participating in some task.

When the conceptual EM is obtained, it can run selecting the bottom corresponded to simulation run in the tool bar. Then, the snapshot in Fig. 30 appeared. It is necessary to set the simulation time and the type of simulation (local or distributed). In Fig. 30, the time is set to 6 month (for the harvest time) and it was selected the option *Local* running. Once the parameter was defined, the *simular* (meaning *simulate*) bottom is selected, the translation process take effect and the simulation begin execute.

When the simulation finishes, the results are shown like the one in Fig. 31. In local environment the results correspond to the metrics identified in resume model. Fig. 31a shows a bar graph representing task execution time (gray bars) and task suspended time (black bars). Fig. 31b shows a bar graph of resources. This graph represents the time in which resources were insufficient for tasks (black bars) and the time in which resources were sufficient for tasks (gray bars).

Graphics show that the "*making yerba canchada*" task has a long inactivity time, and the "*yerba sapecada*" resource has a long time under the stock and it is an input to the "*making yerba canchada*" task. Then, the business analyst can analyze possible improvements before making a decision. A proposal can be to replace the furnaces used in the "*making yerba sapecada*" task, which are controlled by employees, by furnaces mechanically controlled. Thus, the loss of "*yerba sapecada*" will be smaller, from 15% presently to 2% with the improvement. This change will cause the amount of "*yerba sapecada*" to be greater and the "*making Yerba Canchada*"

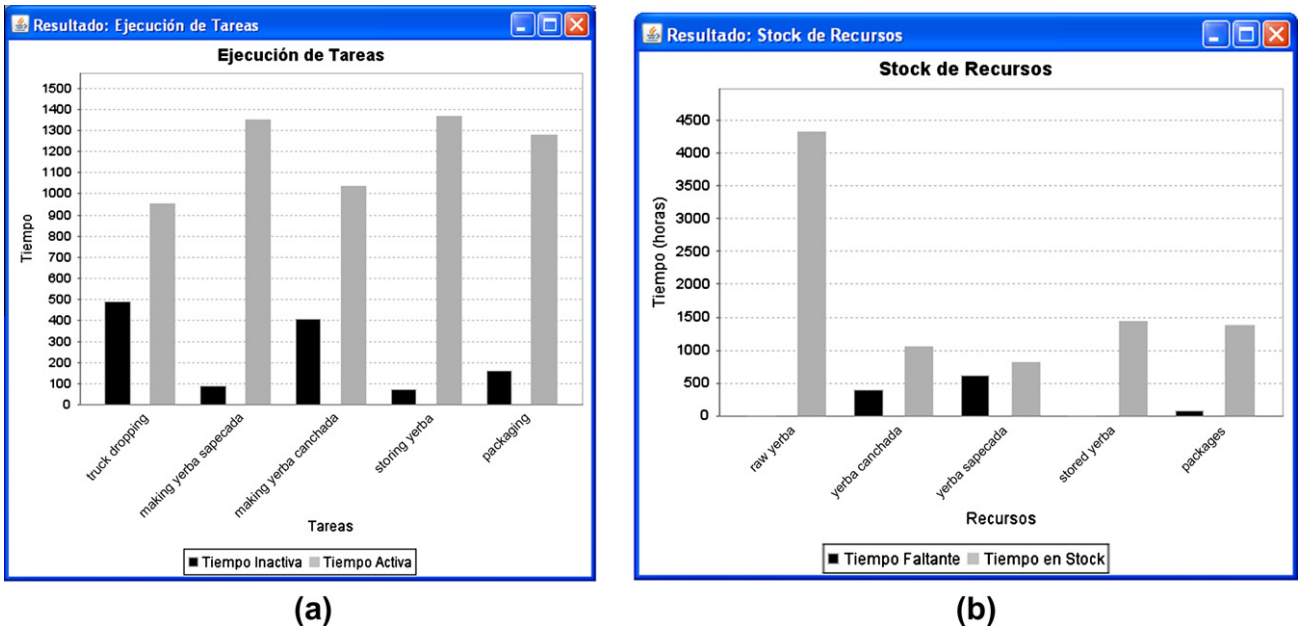


Fig. 31. simulation result.

task will have enough resources to process, so the time in the state suspended will be smaller. Although we were not able to modify the time needed to execute "*make yerba sapecada*" task, we were able to improve the process by incorporating new resources, so the task will be carried out more efficiently with the new resource.

9. Conclusions

This work presents an integrated environment that supports the modeling and simulation of business processes. It gives services to develop a conceptual model of an organization and to analyze its behavior using simulation. In this way, it helps the business expert to introduce his/her knowledge in an easy and friendly way and then analyze the dynamic behavior in both local and distributed environment.

The development of an EM is a complex task, not only for the number of components but also for the multiplicity of actors involved in this activity. Then, in this approach, we have proposed the use of different views to represent the EM, representing each one a different aspect or perspective of the company, without losing the integrated vision of the EM.

We can mention the advantages to obtain the SM from EM through the use of MDA approach, where both static and dynamic aspects of the enterprise are represented:

- SM easy to acquire and execute;
- the results of analysis can be used in decision making, as it takes into account all aspects of the Company;
- the execution of the SM represents interacting business processes behavior;
- avoid the replication of information.

We have chosen the DEVS formalism in the development of simulation layer because:

- it makes easier the modular construction of the SM,
- it facilitates the reuse of SM in different environments, since it propose to differentiate model from the engine that interpret it. In this way, it is possible to develop the SM once and run it with different associated engine, giving as result the satisfactory performance of the model both in local and distributed environments.

The HLA compliant-federate generated with the environment can interact with a variety of other simulators because HLA is widely use in distributed simulation. The use of HLA standard guarantees the interoperability with other simulators. This federate encapsulate the private data and information of an Organization and exposes its behavior through its interface making easier the behavior composition of it.

It is possible extend these concepts in order to model other type of enterprise with different structures such as networked enterprise, distributed enterprise, as well as, whichever company that can develop its model with different granularity levels.

Acknowledgment

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

Appendix A

Rule 1 TaskVersionToFramework

From the top most *TaskVersion* model is obtained a digraph *f* which has two components: *ExperimentalFrame* (ef) and *SimulationModel* (sm).

Transformation *TaskVersionToFramework* (*Coordinates, DEVS*)

Source: v1: Coordinates :: TaskVersion (A.1)

Target: f: DEVS :: digraph
sm: DEVS :: SimulationModel
ef: DEVS :: ExperimentalFrame (A.2)

Source condition

Target condition (A.3)

sm.version = v1

f.components → includes (ef, sm) (A.4)

f.inputports.name → includes ("fIn") (A.5)

f.outports.name → includes ("fOut") (A.6)

sm.inputports.name → includes ("in", "stop") (A.7)

sm.outputport.name → includes ("tr") (A.8)

ef.inputports.name → includes ("in") (A.9)

ef.outputports.name → includes ("out", "stop") (A.10)

- - IC between sm and ef - -

f.cp → exists (t|t.desv1 = ef^t.port1.name = "out"^
t.devs2 = sm^t.port2.name = "in") (A.11)

f.cp → exists (t|t.desv1 = ef^t.port1.name = "stop"^
t.devs2 = sm^t.port2.name = "stop") (A.12)

f.cp → exists (t|t.desv1 = sm^t.port1.name = "tr"^
t.devs2 = ef^t.port2.name = "in") (A.13)

- - EIC

f.cp → exists (t|t.desv1 = f^t.port1.name = "fIn"^t.devs2 = ef^t.port2.name = "in") (A.14)

(*continued on next page*)

- - EOC
f.cp → exists (t|t.desv1 = sm^t.port1.name = "out"^
t.devs2 = f^t.port2.name = "fOut") (A.15)

Mapping
try TaskVersionToSimulationModel **on**
v1.components < ∼ > f.sm (A.16)

Rule 2: TaskVersionToSimulationModel
The *TaskVersion* components are translated in simulation model components.

Transformation *TaskVersionToSimulationModel* (*Coordinates, DEVS*)
Source
v1: Coordinates :: TaskVersion (A.17)
Target
sm: DEVS :: SimulationModel
SoloTareas: DEVS :: TaskVersionDevs (A.18)
Source condition
Target condition SoloTareas.inputports.name→ includes ("er", "eta", "ets", "ed", "stop") (A.19)
SoloTareas.outputport.name→ includes ("sr", "sts", "sta", "sd", "tr") (A.20)
- - The sm components are: SoloTareas and ResourceDevs models:
sm.components→ includes (SoloTareas) (A.21)
sm.components→ includes (v.components → select (OclIsT ypeof (ResourceImp)).devs) (A.22)
- - EIC between sm and SoloTareas:
sm.cp → exists (t|t.desv1 = sm^t.port1.name = "in"^t.devs2 = SoloTareas^ t.port2.name = "ed") (A.23)
sm.cp→ exists (t|t.desv1 = sm^ t.port1.name = "stop"^t.devs2 = SoloTareas^
t.port2.name = "stop") (A.24)
- - EOC between sm and SoloTareas:
sm.cp→ exists (t|t.desv1 = SoloTareas^ t.port1.name = "tr"^
t.devs2 = sm^t.port2.name = "tr") (A.25)
sm.cp→ exists (t|t.desv1 = SoloTareas^ t.port1.name = "sd"^
t.devs2 = sm^t.port2.name = "out") (A.26)
- - IC:
sm.cp→ exists (c|c.desv1 = SoloTareas^ c.port1.name = "sr"^c.devs2 = r^
c.port2.name = "pe"^r.isOclType (ResourceDevs)) (A.27)
sm.cp→ exists (c|c.desv1 = r^ c.port1.name = "ps"^c.devs2 = SoloTareas^
c.port2.name = "er"^r.isOclType (ResourceDevs)) (A.28)

Mapping
try TaskToTaskDevs **on**
v1.taskImp < ∼ > SoloTareas.components (A.29)
try ResourceToResourceDevs **on**
v1.resourceImp < ∼ > sm.resourceDevs (A.30)
try VersionToVersionDevs **on**
v1.taskImp.default < ∼ > SoloTareas.components (A.31)
try TemporalLinkToCoupling **on**
v1.taskImp.temporalLink < ∼ > SoloTareas.cp (A.32)
try TaskResLinkToCoupling **on**
v1.taskImp.taskResLinkImp < ∼ > SoloTareas.cp (A.33)

Rule 3: TaskToTaskDevs
Each *TaskImp* whose default link is empty, is translated in a *AtomicTaskDevs*.

Transformation*TaskToTaskDevs* (*Coordinate, DEVS*)
Source
comp: Coordinates :: TaskImp (A.35)
Target
modelD: DEVS::AtomicTaskDevs
SoloTareas: DEV S :: TaskVersionDevs (A.36)
Source condition:
TaskImp.default.isEmpty () (A.37)
Target condition:
modelD.ta (executing) = comp.execTime (A.38)
SoloTareas.components → includes (modelD) (A.39)
modelD.name = comp.name (A.40)
modelD.model = comp (A.41)

modelD.inputports.name→ includes (“er”, “eta”, “ets”, “ed”, “stop”) (A.42)
modelD.outputport.name→ includes (“sr”, “sts”, “sta”, “sd”, “tr”) (A.43)
SoloTareas.inputports.name→ includes (“er”, “eta”, “ets”, “ed”, “stop”) (A.44)
SoloTareas.outputport.name→ includes (“sr”, “sts”, “sta”, “sd”, “tr”) (A.45)

Mapping

Rule 4: ResourceToResourceDevs
Each *ResourceImp* is translated in a *ResourceDevs* model.

Transformation ResourceToResourceDevs (*Coordinate, DEVS*)
Source
cvr: Coordinates :: ResLifeCycle
comp: Coordinates :: ResourceImp (A.46)
Target
modelD: DEV S :: ResourceDevs
sm: DEV S :: SimulationModel (A.47)
Source condition:
cvr = comp.class.theResLifeCycle (A.48)
Target condition:
sm.components → includes (modelD) (A.49)
modelD.name = comp.name (A.50)
modelD.resource = comp (A.51)
modelD.inputports.name → includes (“pe”, “stop”) (A.52)
modelD.outputport.name → includes (“tr”, “ps”) (A.53)
modelD.state → includes (cvr.theResState. type → oclIsTypeOf (AtomicState).name) (A.54)
modelD.deltext (e, “start”, s) = s’:
cvr.theResState.theStateTransition → exists (t|t.origin = s^t.destination = s’^
t.startTrigger → notempty ()) (A.55)
modeloD.deltext (e, “end”, s’) = dummy:
cvr.theResState.theStateTransition → exists (t|t.origin = s^dummy ∈ DUMMY^
t.endT rigger → notEmpty () (A.56)
modeloD.deltint (dummy) = s”
cvr.theResState.theStateTransition → select (t |t.endTrigger.notEmpty ()^
t.origin = s“^t.destination = s”) (A.57)
Mapping

Rule 5: VersionToVersionDevs
A *TaskVersion* associated a composite task is translated in a *TaskVersionDevs* model.

Transformation *VersionToVersionDevs* (*Coordinate, DEVS*)
Source
comp: Coordinates :: TaskVersion (A.58)
Target
modelD: DEVS :: TaskVersionDevs
firstTask: DEVS :: TaskDevs
finalTask: DEVS :: TaskDevs (A.59)
Source condition:
Target condition:
modelD.name = comp.name (A.60)
modelD.inputports.name → includes (“ed”, “eta”, “ets”, “stop”, “er”) (A.61)
modelD.outports.name → includes (“sd”, “sta”, “sts”, “sr”) (A.62)
modelD.components → includes (comp.components →
select (t|t.isOclTypeOf (TaskImp)).devs) (A.63)
firstTask = comp.firstTask.devs (A.64)
finalTask = comp.finalTask.devs (A.65)
– – EIC:
modelD.cp → exists (t|t.devs1 = modelD^ t.port1.name = “ets”^
t.devs2 = firstTask^t.port2.name = “ets”) (A.67)
modelD.cp → exists (t|t.devs1 = modelD^ t.port1.name = “eta”^
t.devs2 = firstTask^t.port2.name = “eta”) (A.68)
modelD.cp → exists (t|t.devs1 = modelD^ t.port1.name = “er”^
t.devs2 = firstTask^t.port2.name = “er”) (A.69)
modelD.cp → exists (t|t.devs1 = modelD^ t.port1.name = “stop”^

(*continued on next page*)

modelD.components → (p| t.devs2 = p^ t.port2.name = "stop") (A.70)

- - EOC

modelD.cp → exists (t|t.devs1 = finalTask^ t.port1.name = "sd"^
t.devs2 = modelD^t.port2.name = "sd") (A.71)

modelD.cp → exists (t|t.devs1 = finalTask^ t.port1.name = "sta"^
t.devs2 = modelD^t.port2.name = "sta") (A.72)

modelD.cp→ exists (t|t.devs1 = finalTask^ t.port1.name = "sts"^
t.devs2 = modelD^t.port2.name = "sts") (A.73)

modelD.cp → exists (t|t.devs1 = finalTask^ t.port1.name = "sr"^
t.devs2 = modelD^t.port2.name = "sr") (A.74)

Mapping

try TaskToTaskDevs **on**
comp.components < ∼ > modelD.components (A.75)

try VersionToVersionDevs **on**
comp.components < ∼ > modelD.components (A.76)

try TemporalLinkToCoupling **on**
comp.components < ∼ > modelD.cp (A.77)

try TaskResLinkToCoupling **on**
comp.components < ∼ > modelD.cp (A.78)

Rule 6: TemporalLinkToCoupling

A temporal relationship between two tasks is translated in internal coupling between the DEVS models associated with these tasks.

Transformation *TemporalLinkToCoupling*(*Coordinates*, *DEVS*)

Source

tl: Coordinates :: TemporalLink (A.79)

Target

coup: DEVS :: couprel
taskDsourc: DEVS :: TaskDevs
taskDend: DEVS :: TaskDevs (A.80)

Source condition: Target condition: taskDsourc = tl.source.devs (A.81)

taskDend = tl.end.devs (A.82)

tl → oclIsTypeOf (Before, During) implies
taskDsourc.container.cp→ exists (p |p.devs1 = taskDsourc^p.port1.name = "sta"^
p.devs2 = taskDend^p.port2.name = "eta") (A.83)

tl→ oclIsTypeOf (Meet, Equal, Begin) implies
taskDsourc.container.cp→ exists (p |p.devs1 = taskDsourc^p.port1.name = "sts"^
p.devs2 = taskDend^p.port2.name = "ets") (A.84)

Mapping

Rule 7: TaskResLinkToCoupling

A task-resource relationship is translated into external input coupling and external input coupling between TaskVersion models and its components.

Transformation *TaskResLinkToCoupling* (*Coordinates*, *DEVS*)

Source

relTR: Coordinates :: TaskResLinkImp (A.85)

Target

cp: DEV S :: couprel (A.86)

Source condition:

Target condition: taskDsource = relTR.source.devs (A.87)

taskDen = relTR.end.devs (A.88)

taskDsourc.container.cp →exists (p |p.devs1 = taskDsource^p.port1.name = "sr"^
p.devs2 = taskDsourc.container^p.port2.name = "sr") (A.89)

taskDsource.container.cp → exists (p |p.devs1 = taskDsource.container^p.port1.name = "er"^
p.devs2 = taskDsource^p.port2.name = "er") (A.90)

relTR → oclIsTypeOf (processes) implies
taskDsourc.container.cp →exists (p|p.devs1 = taskDsourc.container^p.port1.name = "ed"^
p.devs2 = taskDsourc^p.port2.name = "ed") (A.91)

Mapping

References

[1] Arena <http://www.arenasimulation.com/> [accessed 20.11.07].
[2] Ballou R. Business Logistics Management. Planning, Organizing and Controlling the Supply Chain. Fourth ed. Upper Saddle River, New Jersey: Prentice Hall; 1999. 07458. ISBN 0-13-795659-2.
[3] Bakos JY. From integrated enterprise to regional clusters: the changing basis of competition. Computer in Industry 2000;42:289–98.
[4] Biswas S, Narahari Y. Object Oriented modelling and decision support for supply chains. European Journal of Operational Research. Elsevier; 2004.
[5] Browne J, Zhang J. Extended and virtual enterprise. Similarities and differences. International Journal of Agile Management System 1999;1:30–6.
[6] Byrne P, Heavey C. Simulation, a framework for analysing SME supply chain. In: Proceedings of the winter simulation conference; 2004.
[7] Camarinha-Matos L. Virtual organization in manufacturing: trends and challenges. In: 12th International conference of flexible automation and intelligent manufacturing; 2002. p. 1036–54.
[8] Camarinha-Matos L, Afsarmanesh H. The virtual enterprise concept. In: PRO-VE '99, Proceedings of the IFIP TC5 WG5.3 / PRODNET working conference on infrastructures for virtual enterprises: networking industrial enterprises, vol. 153; 1999. p. 181–99.
[9] Chalmeta R, Grangel R. ARDIN extension for virtual enterprise integration. Journal of Systems and Software 2003;67:141–52.
[10] Childe S. The extended concept of co-operation. Production Planning and Control 1998;9(4):320–7.
[11] Chung Hen Chow A, Zeigler B. Parallel DEVS: a parallel, hierarchical, modular, modeling formalism. In: Winter simulation conference; 1994. p. 716–22.
[12] CIMTOOL. RCGB. <http://cimosa.de/index.html> [accessed 26.10.07].
[13] Cope D, Fayes, Mllaghasemi, Kaylani. Supply chain simulation modeling made easy: an innovative approache. In: Proceeding of the winter simulation conference; 2007.
[14] Czarnecki K, Helsen S. Classification of model transformation approaches. Workship on generative techniques in the context of model-driven architecture; 2003.
[15] DOD. High level architecture interface specification. Version 1.0, defence modelling and simulation organization; 1996. <http://msis.dmso.mil>.
[16] DEVSJAVA 3.1; 2004. <http://www.acims.arizona.edu/SOFTWARE/software.shtml#DEVSJAVA>.
[17] Ding H, Benyoucef L, Xie X, Hans C, Schumacher J. ONE a new tool for supply chain network optimization and simulation. INRIA-Lorraine, MACSI proyect ISGMP, Bat. A, Ile du Saulcy, BIBA. In: Proceedings of the winter simulation conference; 2004.
[18] Fischer M, Jahan H, Teich T. Optimizing the selection of partner in production networks. Robotics and Computer-Integrated Manufacturing 2004;20(6):593–601.
[19] Fujimoto R. Distributed simulation systems. In: Proceeding of the winter simulation conference; 2003.
[20] Gutiérrez M, Mannarino G, Leone H. Coordinates workbench: an object-oriented architecture of a tool for conceptualizing an organization. In: International conference of the Chilean Computer Science Society. Los Alamitos (USA): IEEE Computer Society; 2001, ISBN 0-7695-1396-4. p. 133–42. ISSN: 1522-4902.
[21] Harel D. Statecharts: A visual formalism for complex systems. Science of Computer Programming 1987;8:231–74.
[22] IEEE 1516-2010. IEEE standard for modeling and simulation (M&S) high level architecture (HLA) – framework and rules; 2010.
[23] IEEE 1516.1-2010. IEEE standard for modelling and simulation (M&S) high level architecture (HLA) – federate interface specification; 2010.
[24] IEEE 1516-2.2010. IEEE standard for modelling and simulation (M&S) high level architecture (HLA) – object model template (OMT) specification; 2010.
[25] Jian J, Zhang H, Guo B, Wang K, Chen D. HLA-based collaborative simulation platform for complex product design. In: The 8th international conference on computer supported cooperative work in design proceedings. IEEE; 2003.
[26] Liu Juqi, Wang Wei, Chai Yueting, Liu Yi. EASY-SC: a supply chain simulation tool. In: Proceedings of the winter simulation conference; 2004.
[27] Koehler J, Hauser R, Kuster J, Ryndina K, Vanhatalo J, Wahler M. The role of visual modeling and model transformations in business-driven development; 2008.
[28] Mallidi K, Praskevopoulos A, Paganelli P. Process modeling in small-medium enterprise networks. Computer in Industry 1999;38:149–58.
[29] Mannarino G, Henning G, Leone H. Coordinates: a framework for enterprise modeling. In: Mills JJ, Kimura F, editors. Information infrastructure systems for manufacturing. IFIP-Kluwer Academic Publishers; 1999. p. 379–90.
[30] Mezgar I, Kovacs G, Paganelli P. Cooperative production planning for small and medium-sized enterprises. International Journal of Production Economics 2000;64:37–48.
[31] Ni Y, Fan Y. Model transformation and formal verification for semantic Web Services composition. Advances in Engineering Software. Elsevier; 2010.
[32] Nutaro N, Hammonds P. 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. 2004:19–28.
[33] OMG Object Management Group. In: Miller Joaquin, Mukerji Jishnu, editors. MDA guide version 1.0.1; 2003.
[34] Petri CA. Petri net, communication whit automata. Technical report, PDTIC AD0630125; 1962.
[35] Pokorny T, Fraser M, Burns L, Lim K. poRTIco version 1.0.1; 2009. <http://www.porticoproject.org> [accessed 05.10.09].
[36] Provision. Metastorm. <http://www.metastorm.com> [accessed 06.05.11].
[37] Rice S, Marjanski, Markowitz, Bailey. The SIMSCRIPT III programming language for modular object-oriented simulation. In: Proceeding of the winter simulation conference; 2005.
[38] Siegel J. Developing in OMF's model drive architecture. Technical report, Object Management Group; 2001.
[39] Simprocess CACI Products Company; 2006. <http://www.simscript.com/> [accessed 20.11.07].
[40] SISO. Draft standard for commercial-off-the-shelf simulation package interoperability reference models; 2007. <http://www.sisostds.org/>.
[41] Taylor S, Mustafee N, Turner, Low, Strassburger, Ladbrook. The SISO CSPI PDG standard for commercial off-the-shelf simulation package interoperability reference models. In: Proceeding of the winter simulation conference; 2007.
[42] Terzi S, Cavalieri S. Simulation in the supply chain context: a survey. Computer in Industry, vol. 53. Elsevier; 2004. pag 3 -16.
[43] Verbraeck A. Component-based distributed simulations. The way forward? In: Proceeding of the eighteenth workshop on parallel and distributed simulation. Kufstein, Austria; 2004. ISBN: 1087-4097, ISSN: 0-7695-211-8.
[44] Verbraeck A, Saanen Y, Stojanovic Z, Shishkov B, Meijer A, Valentin E, et al. What are building blocks? Building blocks for effective telematics application development and evaluation. TU Delf Press; 2002 [chapter 2].
[45] Vieria, Guilherme. Ideas for modeling and simulation of supply chains with ARENA. In: Proceedings of the winter simulation conference; 2004.
[46] Von Mevius M, Pibernik R. Process management in supply chain – a new petri net based approach. In: Proceedings of the 37th Hawaii international conference on system sciences; 2004. ISBN: 0-7695-2056-1.
[47] Warmer J, Kleppe A. The Object Constraint Language. Getting your models ready for MDA. Second ed. Addison-wesley; 2003. ISBN: 0321179366.
[48] Wenbo T, Haibin L, Wei Z. Distributed supply chain simulation for decision support in daily operation of the coal enterprise. In: The 2nd IEEE international conference on information management and engineering; 2010.
[49] Wu N, Su P. Selection of partners in virtual enterprise paradigm. Robotic and Computer-Integrated Manufacturing 2005;21(2):119–31.
[50] Yoo T, Kim K, Song S, Cho H, Yücesan E. Applying Web services technology to implement distributed simulation for supply chain modeling and analysis. In: Proceeding of the winter simulation conference; 2009. p. 863–73.
[51] Zha H, Wil van der Aalst, Wang J, Wen L, Sun J. Verifying workflow processes: a transformation-based approach. Software and Systems Modelling 2010;253–64.
[52] Zeigler B, Praehofer, Kim. Theory of modelling and simulation: integrating discrete event and continuous complex dynamic system. 2nd ed. Academic Press; 2000.

María De los Milagros Gutiérrez is a Professor in the Systems Department (Universidad Tecnológica Nacional, Facultad Regional Santa Fe). She is undertaking research into the use of simulation technologies within Business Processes and Engineering Design Environment. She obtained her Phd in Information System engineering degree from the Universidad Tecnológica Nacional in 2009. She is promoter member of the Research Center for Information Systems (CIDISI). Her current research interests are distributed simulation and intelligent agents.

Horacio P. Leone is currently Professor in the Systems Department (Facultad Regional Santa Fe, Universidad Tecnológica Nacional) and Independent Researcher in the Instituto de Desarrollo y Diseño (CONICET-UTN). He is undertaking research into the use of simulation technologies within Business Processes and Engineering Design Environment. He obtained his PhD in Chemical Engineering from the Universidad Nacional del Litoral in 1986. Other research interests are semantic web applications to supply chain information systems and enterprise modeling and organizational dimension of IT. He has supervised a number of successful PhD candidates.