Proceedings of the 2003 European Simulation and Modelling Conference (ESM2003)
Pp:277-282. October 2003. Naples, Italy. ISBN: 90-77381-04-X

MODEL INTEGRITY AND DISTRIBUTED PROCESS INTERACTION SIMULATION

Hans P.M. Veeke, Jaap A. Ottjes
Faculty of Mechanical Engineering and Marine Technology, OCP
Delft University of Technology
Mekelweg 2, 2628 CD Delft, the Netherlands
E-mail: H.P.M.Veeke@wbmt.tudelft.nl, J.A.Ottjes@wbmt.tudelft.nl

INTRODUCTION

Simulation has become a common tool in research projects and large logistic and manufacturing design projects.
Because of the growing scale, the need for distributed simulation grows rapidly also. Usually it is developed from a general information system approach e.g. by means of HLA (Kuhl et al., 1999] and CORBA [Adamski, Hiller, 1998].
Technically these approaches focus on 'data-integrity', but no provisions are made for model-integrity. Above that, complete knowledge about the final simulation structure is required. The iterative course of a design project however, requires a smooth (and preferably transparent) transition from stand-alone to distributed simulation.
The approach as described in this paper, preserves model integrity for process interaction simulation and is implemented in the free source language TOMAS as can be found on TOMASWEB.COM
First the basic elements and process interaction methods for stand-alone simulation are described and then a short description of the concepts used in TOMAS for distributed simulation are explained. The concept is restricted to conservative distributed simulation.

STAND ALONE SIMULATION

According to Zeigler[2000] the process interaction world view identifies the elements of the model and describes the sequence of actions for each element in a structured way. In the process interaction world view, the simulation is considered a collection of interactions among processes. This world view focuses on a sequence of events and/or activities that are logically connected. There are two different views to connect events and activities. Each view corresponds to a different assumption as to what are the active and passive elements in building a model. In the first view the active elements are taken to be the elements that do the processing, in the second view the active elements are the flow elements. The first one is called pure "process interaction" and the second one the "transaction" world view. From a modeler's point of view events have no explicit meaning. From a processor's point of view the system is still based on event scheduling and an event scheduling mechanism is still required.
The "pure" process interaction world view has been implemented in an object oriented way in the simulation language TOMAS [Veeke, Ottjes, 2000]. The approach matches with a formal systems approach to model logistic and manufacturing systems [Veeke, 2003]. In order to connect them in an interdisciplinary way to behavior descriptions of these systems, an intermediate Process Description Language PDL has been developed [Ottjes, Veeke, 2002].

TOMAS implements two basic object classes for stand alone simulation: TomasElement and TomasQueue. The TomasQueue class supports fast and easy methods to collect, select, sort and register TomasElements during the course of a simulation run. Table 1 shows the basic methods of the TomasQueue class.

Method	**Description**
AddToTail(Elm) AddToHead(Elm)	To add element Elm to the tail or head of the queue
AddSorted(Elm,Sort)	To add element Elm to the queue sorted by procedure Sort
PutBefore(E1, E2) PutAfter(E1,E2)	To put E1 before or after E2 into the queue
FirstElement LastElement	To retrieve the first or last element in the queue
Successor(Elm) Predecessor(Elm)	To retrieve the successor or predecessor of Elm
Length MeanLength MinLength MaxLength	To retrieve the actual, mean, minimum or maximum length of the queue
MeanWT MinWT MaxWT	To retrieve the mean, minimum or maximum length of stay

Table 1. Basic methods of TomasQueue

With the TomasElement class both the active and passive model elements can be represented. Passive TomasElements are the model elements being handled or transformed by the active TomasElements. The latter have their own Process description method. All TomasElements have an Identity attribute, an Arrival Time, an Event Time and a Status.
The Status attribute is directly related to the process interaction approach. In TOMAS, 4 values for the Status are defined (see fig.1.)

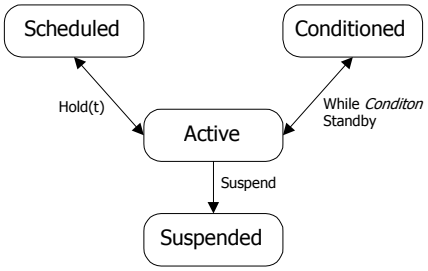

Fig. 1. State values of a Process

A process status can be:

- *Suspended*: the TomasElement "sleeps" and no future event has been defined.
- *Active*: the TomasElement is the "current" element and executing it's event.
- *Scheduled*: the TomasElement is waiting for the simulation clock to arrive at a predefined event time to become "active".
- *Conditioned*: the TomasElement is waiting for a condition to become false and then to be come "active".

As shown in figure 1, a TomasElement must be in the state "Active" to be able to change its state. The sequencing mechanism of TOMAS offers three methods for this purpose:

- Hold(t): The execution of the process description is interrupted and will be resumed at the current simulation clock time + t.
- While *Condition* Standby: the execution of the process description is interrupted and will be suspended as long as *Condition* holds.
- Suspend: the execution of the process description is interrupted indefinitely.

All states except the "Active" state, represent time periods; the Active state represents an event.

The 'Suspended' state is the initial state of every TomasElement. Figure 1 shows that an "external" cause is required to change the process to another state. Here we reach the important issue of "process interaction".

STAND ALONE PROCESS INTERACTION

The only way for a TomasElement to become the active element is to become Scheduled by means of another TomasElement. If the simulation clock time reaches the moment where the element is Scheduled, the sequencing mechanism causes it to become Active. The process description uses the above mentioned method to change to the other states autonomously during the course of its process. However, there can be circumstances where the normal process course – and thus the states Scheduled, Conditioned and Suspended, should be interrupted e.g. because of a machine disturbance, or the arrival of a new order etc. For these situations the TomasElement class offers a number of methods to intervene with the processes of other elements. They are shown in figure 2.

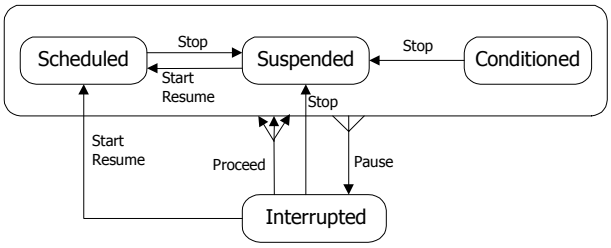

Fig. 2. Process interactions

In the upper part of figure 2 the three time consuming states of figure 1 are shown. For example, a TomasElement can force another element to change from the state Suspended to Scheduled by means of the Start or Resume method. A short description of each method is given in the table below.

Method	Description
Start(t)	Start the process of the element at simulation time t, beginning with the first statement of description.
Resume	Resume the process of the element, with the statement immediately following the last executed statement, which can be Hold, Suspend or StandBy
Stop	Make the process of the element Suspended
Pause	Pause the execution of the process, but remember the state it is in and the time left for its next event
Proceed	Resume the execution of the process as remembered by the Pause method

Table 2. Process interaction methods

The Pause method introduces a new process state: "Interrupted". Interrupted can be compared with Suspended with the difference that the state of the process is completely remembered. From the Interrupted state one may decide to continue the process by means of Proceed, to start it from the beginning by means of Start or to Resume the process, skipping the state it was in.

Figure 2 shows clearly that the methods can only be invoked in certain situations, depending on the state of the process. Starting a process with Start(t) is only allowed if the process is in a Suspended or Interrupted state. The sequencing mechanism of TOMAS preserves this type of *"process integrity"*.

DISTRIBUTED SIMULATION

In research situations and large industrial projects, it often happens that a simulation model grows in size and detail and more than one modeler is constructing separate parts of the model. In these cases it will be very profitable if the model can be split up easily into different stand-alone models that will be synchronized to the same simulation clock. Until now the client-server concept of TOMAS facilitates distributed simulation for both a fast conservative approach and real-time situations. However, the transition from a stand-alone model to a set of distributed models was quite labor-intensive.
The conservative approach can best be described by distributing the processes over several processors, but allowing only one processor to be really 'processing' at any one moment. It is just as "quasi-parallel" as the stand-alone situation. However, instead of one sequencing mechanism, there are several sequencing mechanisms now. Because of this (and of course because of distributing "data") the integrity of the model is not automatically preserved. To find a solution it is necessary to define the term integrity precisely.
Model integrity is defined as the preservation of correct values and correct process states at any moment at any model during a simulation run. The preservation of correct values in a stand-alone situation is automatically preserved by TOMAS, because it is a single processor application where no threads or agents are being used. There is also only one sequencing mechanism present, which takes care of the process states. From the above definition it becomes clear that we should distinguish data-integrity and process-integrity.

DATA-INTEGRITY

Data-integrity could be achieved by using HLA or CORBA for example. However, the application of HLA and CORBA is not straightforward and again very labor-intensive. TOMAS uses a simple, but fast TCP/IP-based message concept that can be made transparent for the modeler, where the data-integrity of TomasElements and TomasQueues is concerned.

The basic assumptions to achieve this type of integrity are:

- Only one processor is actually processing
- Only one process is actually 'Active' during the simulation run
- The "same" TomasElement may be present in different models, but not necessarily with the same attributes.

The last assumption needs some explanation. Usually data-integrity leads to the requirement that each object may only be present once and only once in the whole model structure. There are two reasons to drop this requirement:

- in each participating model, the same object can play a different role with different attributes. For example, a container in a transportation model only needs a size and weight, while the same container needs other attributes like "empty/not empty" and "reefer/non-reefer" in a storage model.
- By restricting this approach to the conservative distributed concept, only one model is running at any moment. So there is only one source, which can alter the attribute values.

Data-integrity can now be preserved by updating the common and generally defined attribute values at the moment that the active model returns control to the server in order to advance the simulation clock. The implementation of data-integrity can thus be achieved by defining one object (class) definition in one model to be the "home element" and the other ones in the other models to be "remote elements". It is up to the modeler to decide on which class definition is to be considered the home definition. As soon as a model containing remote elements returns control to the server, the sequencing mechanism communicates all changed objects to the server. The server distributes these updates to the "home" models involved. To realize this distinction TOMAS now recognizes a "TRemoteElement" class. The home element can still be defined as a TomasElement class.

Example

Suppose a distributed model of container transfer contains a.o three submodels: a generator model, an

AGV transportation model (Automatic Guide Vehicles) and a Storage model. The modeler decides to define the container class in the generator model as the home element class. The class definitions in the model could look like the ones shown in table 3 (in Delphi Pascal).

Generator model Tcontainer = class(TomasElement) Size: Integer; Weight: Integer; Reefer: Boolean; Empty: Boolean; Stored: Boolean; End;
AGV Transportation model TContainer = class(TRemoteElement) Size: Integer; Weight: Integer; OnAGV: AGVReference; End;
Storage Model TContainer = class(TRemoteElement) Size: Integer; Reefer: Boolean; Empty: Boolean; StackPosition: XYZ_Position; Stored: Boolean. End;

Table 3. Home and remote elements

Size, Weight, Reefer and Empty are considered the common attributes here. At the first time after receiving control from the server, a model refers to a container, the attribute values of the home element are "automatically" communicated through the distributed sequencing mechanism of TOMAS. If other models need to know the StackPosition, the modeler is free to add this attribute to the definition in the Generator model. If the container is stored in the Storage model, the "Stored" attribute is set to TRUE and when this model returns control to the Server again, this attribute value is automatically updated by the distributed sequencing mechanism.

PROCESS-INTEGRITY

Process-integrity (as defined by figure 2) should be preserved by and between the different sequencing mechanisms. In this way it can be made (almost) transparent to the modeler.
Two situations should be distinguished:

1. the process description of an element is completely contained in only one participating model.
2. The process description is split into parts, where each part is contained in a different model.

In situation 1 the process interaction methods of figure 2 can be called from any model. Suppose the home element is defined in the model containing its process description. The other models contain the remote element definition, but remote elements also have process interaction methods. In this case, a call to "Start" the process of a remote element is interpreted by the distributed sequencing mechanism and translated into a "Start message" for the home element. The sequencing mechanism in the home model receives the message and performs the required action if it is allowed (just as it did in the stand alone case).
Situation 2 however is more complicated. The distinction between home and remote element definitions is not sufficient to guarantee process integrity. It should be guaranteed that the element can be active, scheduled or conditioned in only one model at a time. To achieve this an extra process state "Disabled" is introduced. At the moment of creation the home element initializes to the state "Suspended". Above that, all remote elements are considered "Disabled" initially. Process interaction statements are considered to apply to the one element which is not "Disabled". Now the method to perform the state transitions to and from the state "Disabled" must be defined. For this purpose the element method "SwitchToModel(*modelname*)" is introduced. The method is defined for both home and remote elements.
First of all this method can be called by the element itself whenever it is in the Active state (see figure 3).

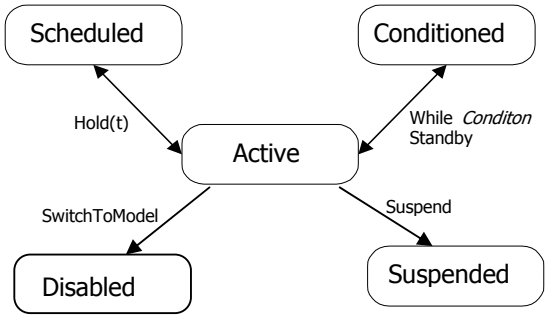

Figure 3. State values of a distributed process

By calling "SwitchToModel" the element itself will become "Disabled" and the distributed sequencing mechanism sends a message to the server to enable the corresponding (home or remote) element in the model mentioned. If the sequencing mechanism in

this model receives this "enable"-message it changes the state of the element concerned to "Scheduled" at the current simulation clock time. The process of this element will resume its course from the statement following a SwitchToModel or –if it is the first time of enabling- will start its process description.

Secondly, if the SwitchToModel is called when the element is not Active, the process is required to be in a Suspended state. It is physically impossible to be scheduled, conditioned or interrupted in two models at a time. Therefore SwitchToModel will only be accepted by the sequencing mechanism if the element is Suspended or Active. As described above, when the element is "enabled" it changes from a Disabled to a Scheduled state. Figure 4 now shows the process interactions in a distributed environment.

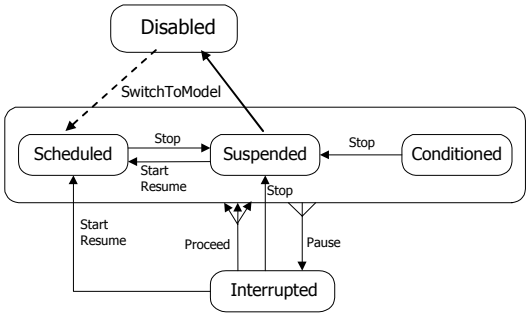

Figure 4. Distributed Process Interactions.

Example

Suppose a transportation system with Automatic Guided Vehicles (AGV) provides the transport of containers between quay cranes and stacking cranes. Arriving at a quay crane AGV's line up in a QuaycraneQueue and drive forward until they reach the transfer position. There an AGV is loaded and drives to a stacking crane. At a stacking crane it searches a free position and waits until the container is picked. Then the AGV drives to the next quay crane.
Initially the process of an AGV was completely described in one stand alone model. The process description of an AGV looked like below (assuming an AGV starts at a transfer position near a stacking crane).

Process of an AGV

```
While TRUE Do
Begin
  While JobList.Length = 0 Do
    StandBy;
  Hold(Drivetime);
  EnterQueue(QuaycraneQueue);
  Suspend;
  Hold(DriveTime) ;
  EnterQueue(TransferPositions) ;
  Suspend;
End;
```

The model is now split up in three different models: a Quay crane Model, an AGV model and a Stacking model. In order to study the behavior and control at quay cranes and stacking cranes further, the process of an AGV is also split up. This is now easily accomplished as shown below.

AGV_model

Process of an AGV

```
While TRUE Do
Begin
  While JobList.Length = 0 Do
    StandBy;
  Hold(Drivetime);
  SwitchToModel(Quaycrane_model);
  Hold(DriveTime) ;
  SwitchToModel(StackingModel)
End;
```

Quaycrane_model

Process of an AGV

```
While TRUE Do
Begin
  EnterQueue(QuaycraneQueue);
  Suspend;
  SwitchToModel(AGV_model) ;
End;
```

Stacking_model

Process of an AGV

```
While TRUE Do
Begin
  EnterQueue(TransferPositions);
  Suspend;
  SwitchToModel(AGV_model) ;
End;
```

Now in each of the models the descriptions can be further detailed independently and concurrently.

CONCLUSIONS AND FUTURE RESEARCH

It has been shown that the use of distributed simulation can be logically connected to a stand alone approach. In this article distributed simulation was restricted to the conservative (i.e. quasi-parallel) concept. In this case the requirements of model integrity should be preserved as they already are in the stand alone environment. In TOMAS this has been implemented by distinguishing home and remote elements, by introducing an extra process

state “Disabled” and by adding an extra process interaction method “SwitchToModel”. In the near future the concept will be expanded for real-time distributed simulation in order to support prototyping of real equipment and control software.

The model integrity concept will be used in a starting research project VISITOR, which aims to construct a virtual industrial system for education and research purposes.

REFERENCES

Zeigler, B.P., Praehofer, H., Kim, T.G., 2000, “*Theory of Modeling and Simulation*”, Academic Press, San Diego, 2nd Edition.

Ottjes, J.A., Veeke, H.P.M., 2002, “*Prototyping inProcess Oriented Modeling And Simulation*”, Proceedings of the 16th European Simulation Multiconference ESM 2002, pp. 20-26, Darmstadt, Germany

Kuhl,F., Dahmann,J. Weatherly,R.,1999*,”Creating Computer Simulation Systems: An Introduction to the High Level Architecture”,* Prentice Hall, ISBN 0130225118

Veeke, H.P.M., Ottjes, J.A., 2000, “*TOMAS: Tool for Object oriented Modeling And Simulation*”,Proceedings of Advanced Simulation Technology ASTC 2000, pp. 76-81, WashingtonDC, SCS International

Adamski, D., Hiller, M., 1998, “*CORBA in Simulation Tasks*”, Proceedings of EUROSIM 1998, Helsinki, Finland

Veeke, H.P.M., 20003, “*Simulation Integrated Design for Logistics*”, Dissertation Delft University of Technology, ISBN 90-407-2417-2