Proceedings of the 2003 European Simulation and Modelling Conference (ESM2003)
Pp:283-288. October 2003. Naples, Italy. ISBN: 90-77381-04-X

TRANSPARENT DISTRIBUTED DISCRETE EVENT MODELING

Jaap A. Ottjes
Delft University of Technology, Sub Faculty of Mechanical Engineering and Marine Technology
Transport and Logistic Technology Group
Mekelweg 2, 2628 CD Delft
The Netherlands
J.A.Ottjes@wbmt.tudelft.nl

Hans P.M. Veeke
Delft University of Technology, Sub Faculty of Mechanical Engineering and Marine Technology,
Industrial Engineering Group
Mekelweg 2, 2628 CD Delft
The Netherlands
H.P.M.Veeke@wbmt.tudelft.nl

ABSTRACT

Stand-alone modeling and distributed modeling show considerable differences in methodology and amount of modeling effort needed. An approach is discussed to design and develop distributed models with the ease of stand-alone modeling and without the overhead usually associated with distributed modeling. The main objective is to obtain interchangeability of a stand-alone model and its distributed equivalent, maintaining the model structure in all models and to minimize code alteration and overhead. Important requirements are reproducibility of simulation experiments and model integrity. The process interaction modeling approach appears to be very appropriate for this. Most attention is paid to the interaction between model elements in stand-alone as well as distributed models. Some examples are show in pseudo language and demands on simulation software are derived.

INTRODUCTION

In traditional applied modeling and simulation of discrete systems generally 'stand-alone' models are used. Several languages and packages are available for this. A stand-alone discrete event simulation model can be characterized by having one time sequencing mechanism. Usually the model is contained in one program. Distributed discrete event simulation models are constituted of interacting sub models or member models. These sub models may reside in one or more computers. Sub models may have their own local time sequencing mechanism but have to be mutually synchronized by a central clock.

Reasons to use distributed modeling instead of or in addition to stand-alone modeling may be:

- Enabling concurrent modeling by several modelers each developing a part of the total model.
- Enabling zooming into a part of the model i.e. to expand one sub model into more detail letting the other models unchanged.
- Running sub models at different locations possibly with human interfaces, for example for training purposes.
- Running the model real time as a virtual environment for testing real resources and control functions, (Ottjes and Hoogendoorn, 1996).

In some cases having an (initial) stand-alone model as well may have advantages in terms of development, speed and testing, (Duinkerken et al, 2002).
In the next chapters the characteristics of process interaction modeling will be discussed in some detail. Then it will be applied to both stand-alone modeling and distributed modeling. Some simple examples will be shown as an illustration and the key demands for transparent modeling will be derived from it.

DISCRETE EVENT MODELING

Zeigler (Zeigler 2000) distinguishes three 'world views' on discrete event systems:

Event scheduling: An event-scheduling algorithm employs a list of future events ordered by increasing event times.
Activity scanning: concentrating on the activities in a system and the conditions that allow an activity to start.
Process interaction being a combination of event scheduling and activity scanning. It can be characterized as identifying the systems elements and describing the sequence of actions of each one. The sequence of actions of an element is called its process. There are two different views corresponding to a different assumption of what are the active and passive elements. The first view is the pure "process interaction". Here the active elements are taken to be the elements that do the processing. In the second view called the "transaction" view the active elements are the flow elements.

Though the transaction view is the most frequently used technique, being able to also describing the processes of the active permanent elements as well increases the flexibility of modeling, (Fishman, 2001). The first language applying this approach is "Simula" (Birthwistle, 1973); two recent tools are Silk (Healy and Kilgore, 1997) and Tomas (Veeke and Ottjes, 2000), (Veeke, 2003).

PROCESS INTERACTION MODELING

The process interaction modeling is quite straightforward using the following recipe:

- Distinguish the relevant classes of elements and their relevant attributes; elements of one class own the same set of attributes.
- Distinguish between active and passive element classes. At this stage the worldview emerges. Provide the process description of the active classes. A process of an active class displays the activities of elements of that class as a function of time. Consequently in a process 'time consuming' commands appear. Further an active element may interact with other elements. Therefore control commands are needed.

TIME CONSUMING COMMANDS

In a process description of an element class so called 'time consuming' commands occur. Time consuming refers to system time during simulation. If a time consuming command appears in a process description the process halts for a certain time interval and later on continues according table 1.

Table 1. Time consuming commands and rules for process continuation.

Command	Continuation rules
Advance	if triggered by "resume" command from other element
Advance τ	automatically after τ time units
Advance while/until condition	automatically after condition=false/true

If more elements are following their process at the same time quasi parallelism is obtained. For that a mechanism is needed that schedules the right processes at the right times in the right order. Moreover it has to bring up to date the system time. This mechanism is called the time sequencing mechanism. Every event based simulation language is provided with such a mechanism.

CONTROL COMMANDS

Active elements need the possibility to act upon the process of other elements in the model influencing the status of processes of these elements.

The basic commands for interaction are listed in table 2. The so-called 'dot notation' familiar in object oriented modeling, is used. Apart from Element classes a Set class is appropriate in modeling. A Set may contain elements. A set is used to represent a physical waiting queue of elements but is also very useful in modeling control and selection logic.

Table 2. Basic commands for interactions between processes of active elements E1 and E2

Command in process of element E1	Result
E2.Start T	Start the process of E2 from its beginning at time T
E2.Interrupt	Immediate interrupt the process of element E2
E2.Resume T	Let element E2 continue at time T with its process (after having been interrupted)
E2.Stop	Stop the process of element E2 immediately

Some typical Set related commands and attributes are listed in table 3.

Table 3. Set related commands and attributes

First	Refers to the first element of the Set
Last	Refers to the last element of the Set
Successor	Refers to the successor of some element
Predecessor	Refers to the Predecessor of some element
Leave / Remove	Methods to remove element from the Set
Enter / Add	Methods to add elements to the Set
Length	Actual number of elements in the Set

STAND ALONE VS. DISTRIBUTED MODELING

It is the purpose of this work to define an approach to develop models with the ease of stand-alone modeling and without the overhead usually coupled with distributed modeling which is the case for example in HLA applications (Fujimoto, 2000), (Klein and Strassburger, 1998). Both distributed model and its stand-alone version should be equivalent. Moreover we want to be able to switch between both stand-alone model and distributed model at any stage of development without code alteration or additional programming.

In a process interaction model the interactions between elements are modeled using the commands in table 1 and table 2 and the set operations are modeled using the commands in table 3. As a consequence elements need access to other elements they have to interact with. In a stand-alone model this is an obvious matter. In a distributed model special facilities are required. Moreover additional requirements are to maintain data and process integrity and reproducibility of simulation runs. In the next examples the stand-alone and equivalent distributed modeling will be illustrated in terms of an informal process description (pseudo)

language (Ottjes and Veeke, 2002) and consequences for the modeling and simulation software are discussed.

EXAMPLE

Suppose we have a generator of loads that have to be transported from a pickup location to various destinations. Automated guided vehicles (AGVs) are carrying out the transportation of the loads. Every load owns its specific travel time to its final destination. This time is an attribute of the class Load.

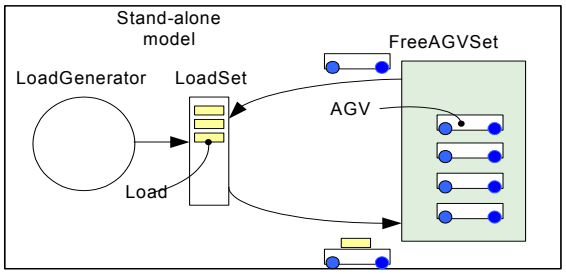

Figure 1. The stand-alone model (model 1)

First we will elaborate the stand-alone model (model 1) of this system. After that a distributed model (model 2) will be derived consisting of two sub models: The *Generator model* containing the Load-Generator and the *Transport model* containing the AGVs. Next a distributed model formed by the same Generator model, a reduced Transport model and for each AGV a separate distributed AGV model (model 3) will be discussed. Finally the AGV model is further elaborated into a control part and a part that represents the physical driving, (model 4). In figure 1 the stand-alone model is depicted.

Model 1: The Stand-Alone Model

Following the modeling rules as given earlier we distinguish two active element classes: the Loadgenerator and the AGV and one passive class: the Load, each with its specific attributes and methods: Figure 2a.

Classes	Attributes
LoadGenerator: ElementClass	ArrivalTime-Distribution TravelTime-Distribution PROCESS
Load: ElementClass	TravelTime
AGV: ElementClass	PROCESS
LoadSet: SetClass	
FreeAGVSet: SetClass	

Figure 2a: Definition Section of the stand-alone model

Two element classes own a process: The Generator shown in figure 2 b and the AGV shown in figure 2c.

("Now" means the current system time; "//" precedes a comment)

```
NewLoad: Load   //local reference
FreeAGV:AGV    //local reference
Loop
  Advance InterArrivalTimeDistribution.Sample
   NewLoad = Load.Create
   NewLoad.TravelTime=   TravelTimeDistribution.Sample
   NewLoad.enter(LoadSet)
   If FreeAGVSet.Length>0 then
     FreeAGV = FreeAGVSet.FirstElement
     FreeAGVSet.Remove(FreeAGV)
     FreeAGV.Resume Now
```

Figure 2b: The LoadGenerator Process in the stand-alone model

```
MyLoad : Load          //local reference
Loop
   While LoadSet.Length > 0 do
     MyLoad = LoadSet.First
      Loadset.Remove(MyLoad)
      Advance MyLoad.TravelTime
      MyLoad.Destroy
   Enter(FreeAGVSet)
   Advance
```

Figure 2c: AGV Process in the stand-alone model

Figure 2d depicts the initialization part of the model.

```
LoadSet             = Set.Create
TheGenerator        = Generator.Create
TheGenerator.Start Now
FreeAGVSet          = Set.Create
Loop(NumberOfAGVs)    //create the AGVs
   NewAGV = AGV.Create
   NewAGV.Start Now
```

Figure 2d: Initialization Section of the stand-alone model

Model 2: Two Sub-Models

Now we will split the stand-alone model into two interacting sub models forming a distributed model:

- The Generator model containing the classes LoadGenerator and Load and the LoadSet
- The Transport model containing the class AGV and the FreeAGVSet.

We intend to use the same process code and will discuss the consequences of that. Therefore it is necessary to identify at what occasion one model needs information from or wants to interact with the other model. Looking at the processes the Generator has to 'know' about the FreeAGVSet and AGVs defined in the Transport model and each AGV has to 'know' the

LoadSet and Loads in the Generator model. Moreover an AGV should be able to access the attribute TravelTime of its Load. Further the Generator should be able to remove an AGV from the FreeAGVSet and resume its process. Its sounds reasonable to require that it should be known in a model whether references to a Set or an Element concern 'own' items or items that are created in another sub-model. We will refer to sets and elements in another model as 'remote' set class and 'remote element class'.

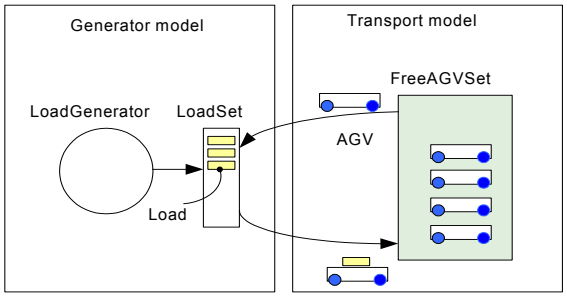

Figure 3. Two sub-models (model 2)

The Generator sub-model then shows as follows:

Classes	*Attributes*
LoadGenerator: ElementClass	ArrivalTime-Distribution TravelTime-Distribution PROCESS
Load: ElementClass	TravelTime
AGV: **RemoteElementClass**	
LoadSet: SetClass	
FreeAGVSet: **RemoteSetClass**	

Figure 4a: Definition section of the Generator model

In the definition section only the definitions of reference to remote element classes and sets are changed accordingly.

```
LoadSet  = Set.Create
TheGenerator      = Generator.Create
TheGenerator.Start Now
FreeAGVSet        = RemoteSet.Create
```

Figure 4b: Initialization Section of Generator model:

The initialization section of the Generator model only contains the relevant definitions of figure 2d
The process description in the LoadGenerator class remains unchanged.

The Transport sub- model of model 2 is defined in figures 4a and 4b.
The process description of the AGV class remains unchanged. It is clear that splitting models into sub-models bares the risk of disruption data integrity. For example if in one sub-model an attribute of an element is changed then this alteration should be unambiguous. This problem can be solved in principle by letting each element have an unambiguous owner sub-model. In model 2 for example the owner model of a load is the Generator model.

Every alteration of attributes and status of the element has to be adjusted in the owner model at the right moment. This puts further demands on the implementation of the simulation software. When switching between stand-alone model and distributed model the results of simulation runs should be identical and reproducible. Reproducibility is obtained if the time sequencing mechanism of the distributed model is implemented conservative.

Element: Class	*attributes*
Load: RemoteElementClass	
AGV: ElementClass	PROCESS
LoadSet : RemoteSetClass	
FreeAGVSet: SetClass	

Figure 4a: Definition Section of the Transport model

```
LoadSet  = RemoteSet.Create
FreeAGVSet        = Set.Create
Loop(NumberOfAGVs)
   NewAGV = AGV.Create
   NewAGV.Start
```

Figure 4b: Initialization Section of Transport-model

Requirements Derived
If we require that the implementation of both stand-alone and distributed models do not require more adapting than in the pseudo model i.e. just discriminate between 'own' and 'remote' element classes and sets and leaving processes unchanged, this determines the requirements on the simulation software so far:

In a distributed model:
- Each sub-model should be able to refer to elements and sets in all other sub models.
- Each sub-model should have access to (read and write) attributes of elements and sets in all other sub-models
- Each sub-model should be able to control the process of elements in all other sub-models (cancel, resume, remove, add, etc.)
- Data integrity should be maintained.

Next the example will be further elaborated into more detail.

Model 3: A Sub-Model For Each AGV
We will now create for each AGV its own AGV model. Consequently the Transport model only contains the FreeAGVSet.

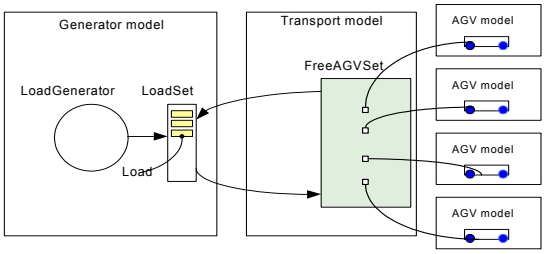

Figure 5: All AGVs have their own sub-model (model 3)

The Generator model as defined in model 2 does not need any adjustment.

The definition section of the Transport model turns to the one shown in figure 6a:

Element: Class	*attributes*
LoadSet : RemoteSetClass	
FreeAGVSet: SetClass	

Figure 6a: Definition Section of the Transport model

And the initialization section is reduced to figure 6b:

LoadSet	= RemoteSet.Create
FreeAGVSet	= Set.Create

Figure 6b: Initialization Section of Transport model

The AGV model

Element: Class	*attributes*
Load: RemoteElementClass	
AGV: ElementClass	PROCESS
LoadSet : RemoteSetClass	
FreeAGVSet: RemoteSetClass	

Figure 6c: Definition Section of the single AGV model

FreeAGVSet	= RemoteSet.Create
LoadSet	= RemoteSet.Create
NewAGV	= AGV.Create
NewAGV.Start	

Figure 6d: Initialization Section of the single AGV model

The process description of the AGV class remains unchanged, see figure 2c.

As a conclusion we can say that going from model 2 to model 3 no statement has changed. They only are re-distributed over the sub-models.

Model 4: Zooming Into The AGV Model

If we now concentrate on the AGV model and especially on the AGV process we see that the process model has three different functions:

1: Waiting for a new Load
2: The selection of a new load; selecting means here simply taking the first load in the LoadSet. Selection however also may consist of any other selection procedure. We can say that the AGV has its own "local intelligence" aboard with respect to selecting loads. Of course the selection of loads also could have been modeled using a central dispatcher located in the Transport model.
3: The actual driving, modeled simply by suspending the AGV process during its loads travel time, see figure 2c.

Zooming

The next step is zooming into the driving part of the AGV process, taking into account a road map, traffic control (Duinkerken et al, 1999), and AGV physical driving characteristics. The information needed from the load should be given in terms of destination rather than travel time. With such a model load assignment algorithms and traffic control methods can be tested and optimized. As simulation speed of distributed models decreases by delays caused by messaging between the sub models, this testing of control algorithms is best be done using the stand-alone model variant.

Splitting Up Processes

A further step is to split up the AGV process in parts implemented in *different* sub-models. In that case an AGV has to be "transferred" from one sub-model to another following in each of these sub-model its process part. Here for example the 'waiting for a load' part of the model could be implemented in the generator model leaving only the 'driving part' in the AGV model. Apart from data integrity now we also have to preserve "process integrity" to be defined as the demand that, if a process of an element is distributed over several sub-models, the element may only be active in one of the sub-models at the time. In all other sub-models its process should be disabled. We will not further go into this matter in this paper. In (Veeke and Ottjes, 2003) the implementation consequences are elaborated.

The final step is the introduction of the real equipment e.g. one or more AGVs. The distributed models then act as a virtual environment in which real AGVs can function in cooperation with simulated ones. Obviously the model should run real time in that case, (Lindeijer, 2003). Figure 7 shows a scale model of an AGV equipped with a computer, a wireless network connection and an interface between the physical driving system and the control part of the AGV process.

It is possible now to control the AGV using a distributed simulation model. The AGV is used in current research.

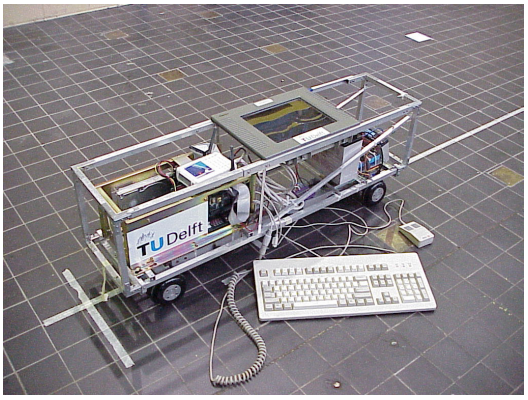

Figure 7. Scaled AGV with a "sub-model" on board contained in a distributed model with wireless communication via the Internet.

CONCLUSIONS

An approach is discussed to design and develop distributed models with the ease of stand-alone modeling. The main objective is to obtain interchangeability of a stand-alone model and its distributed equivalent, maintaining the model structure in all models and to minimize code alteration and overhead. To that end simulation software has to fulfill the following demands:

- Each sub-model should be able to refer to elements and sets in all other sub models.
- Each sub-model should have access to (read and write) attributes of elements and sets in all other sub-models
- Each sub-model should be able to control the process of elements in all other sub-models (cancel, resume, remove, add, etc.)
- Data integrity should be maintained.
- Process integrity should be maintained.
- Simulation runs in both stand-alone model and distributed model should be reproducible.

CURRENT AND FUTURE RESEARCH

The philosophy explained is being implemented in the simulation tool Tomas. The results will be applied in prototyping of logistic systems in production and transportation. Currently a project is carried out to realize "hardware in the simulation loop".

REFERENCES

Birtwistle, G. M., O. J. Dahl, B. Myhrhang, K. Mygrard (1973), Simula Begin, Van Nostrand Reinhold, New York.

Duinkerken, M.B., Evers, J.J.M., Ottjes, J.A. 1999. *TRACES: Traffic Control Engineering System.* Proceedings 31st Summer Computer Simulation Conference. Chicago [SCS] 1999, pp. 461-465.kl

Duinkerken, M.B, J.A. Ottjes, G. Lodewijks, 2002; *The Application of Distributed Simulation in TOMAS: Redesigning a Complex Transportation Model.* Proceedings of the 2002 Winter Simulation Conference (WSC 2002). December 2002. San Diego. ISBN 0-7803-7615-3

Fishman, G.S. 2001. *Discrete Event Simulation. Modeling, Programming, and Analysis.* @001 Springer-Verlag New York, Inc. ISBN: 0-387-95160-1, 52-59

Fujimoto R.M. 2000. *Parallel and Distributed Simulation Systems.* Wiley Series on Parallel and Distributed Computing. Wiley, New York.

Healy, J. R.A. Kilgore, 1997. "Silk,: A Java-Based Process Simulation Language". *Proceedings of the 1997 Winter Simulation Conference, IEEE*,

Klein, U., Strassburger, S., Beikirch, J. *Distributed Simulation with JavaGPSS based on the High Level Architecture*. International Conference on Web-based Modeling and Simulation, Jan. 11.-14. 1998, San Diego.

Lindeijer, D.G, 2003, *"Controlling Automated Traffic Agents"*, Dissertation, Univ. of Techn. Delft, Februari 2003, ISBN 90-5166-948-8

Ottjes, J.A., F.P.A. Hogedoorn, 1996; "Design and control of multi-AGV systems: reuse of simulation software." *Proceedings of 8th European simulation symposium.*
Society for Computer Simulation International Genoa 1996, p. 461-465. ISBN: 1-56555-099-4

Ottjes J. A. , Veeke, H. P.M., , 2002 "*Prototyping In Process Oriented Modeling And Simulation"*,
Proceedings 16th European Simulation Multiconference ESM 2002, pp 20-26, ISBN 90-77039-07-04, Darmstadt, Germany

Veeke, Hans P.M., Jaap A. Ottjes, 2000. Tomas: Tool for Object-oriented Modeling And Simulation. *In proceedings of Advanced Simulation Technology Conference (ASTC2000).* April 16-20, 2000, Washington, D.C. pp. 76-81. The Society for Computer Simulation International (SCS), ISBN: 1-56555-199-0

Veeke, Hans P.M., Jaap A. Ottjes, 2002. TomasWeb: web site: www.tomasweb.com

Veeke, H.P.M, 2003, *"Simulation Integrated Design for Logistics*", Dissertation, Univ. of Techn. Delft, June 2003, ISBN 90-407-2417-2

Veeke, H.P.M., Ottjes, J.A., 2003, *"Model Integrity and Process Interaction Simulation*", Submitted to the European Simulation and Modelling Conference (ESM2003) Naples, Italy,

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