Proceedings of the 15th European Simulation Multiconference (ESM2001).
June 2001. Prague [SCS]. ISBN 1-56555-225-3

APPLIED DISTRIBUTED DISCRETE PROCESS SIMULATION

Hans P.M. Veeke and Jaap A. Ottjes
Sub Faculty of Mechanical Engineering and Marine Technology, Fac. 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

KEYWORDS
Discrete simulation, Model design,
Process-oriented modeling,Transportation,
Automatic control

ABSTRACT

With the growing scale and complexity of simulation projects, stand-alone simulation models are becoming too limited to support the design phase in a sufficiently valid way. Besides that there is a trend to combine separately developed models and extend the use of modeled software to prototyping or even real environments.
This paper describes an open, easy and general concept for distributed modeling and simulation. The concept is suitable for research, educational and commercial design environments. The concept does not require special hardware or study and is, for the sequencing part, completely transparent to the simulationist. It is implemented in the free and open source simulation tool 'TOMAS', which is based on the process approach for discrete event simulation.
To show a concrete example, the concept is applied to a dimensioning problem for AGV systems.

INTRODUCTION

Simulation is nowadays being used in almost all complex design problems. Some developments changed the demands to the traditional simulation approach.
Firstly the size of the projects, where simulation is being used increased, so a simulation model became too big to be developed by only one single expert. This resulted in the need to be able to develop models, structured in such a way, that more experts could work them out in parallel.
Secondly the complexity of the design problems increased, which led to a design approach where simulation is needed during different design phases. A model cannot be specified immediately in all of its details. Some kind of aggregation facility is needed.
In the third place the investments in programming efforts increased dramatically because of the growing degree of automation in real systems. The need was felt to use modeled control programming code in the real environment.
And finally, because of the common use of simulation, it became a real topic to reuse some of the elements developed in earlier design projects. Object orientation has proven to be a real answer to this last aspect. For aggregational modeling there's no platform available yet.
Distributed simulation is presented in this paper as a solution to cope with the increased size of projects and with the need of using modeled control algorithms in a real control system. Although worldwide efforts are being made to make e general concept for distributed modeling and simulation [Fujii et al., 1999], and HLA is becoming a standard [Page/Smith, 1998], the authors felt a need to develop a simple, open and flexible facility to be used in educational and research environments. The concept is implemented in the free available process simulation environment of TOMAS [Veeke/Ottjes, 2000] and applied to model an AGV control system, where different traffic control algorithms must be tested on collisions, deadlocks and capacity bottlenecks.

THE CONCEPT OF DISTRIBUTED SIMULATION

A stand-alone simulation *model* can be considered a dynamic structure of *elements* sequenced on the same time-axis. The structure is called dynamic because of the arrival and departure of temporary elements (normally called the 'flow' elements). This is schematically shown in fig. 1 for a simple lock system, where ships arrive from each side, are being locked and leave. For explanation purposes the lock control and the lock itself are show separately.

Is a distributed *model* then a dynamic structure of simulation *models* sequenced on the same time-axis? The answer is no, this definition would restrict the qualification 'distributed' to simulation models exclusively. But in a simulation model with control elements it often occurs that the control algorithms are called at specific moments (events) and respond immediately (timeless). If we develop the algorithms in separate programs, then these programs don't have any notion of 'time'. In fig.1 'lock control' will only be called to select a ship that receives allowance to enter the lock. So the

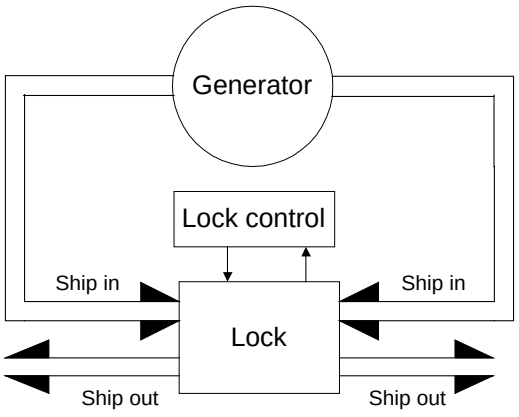

Fig. 1. Stand-alone lock model

definition must be: *A distributed model is a dynamic structure of member models and member programs, where the member models are synchronized to a common time-axis.*
The definition still contains the term dynamic to emphasize the possible arrival and departure of member models. In practice we encounter these situations easily. For example a model of a production center can be part of the distributed model, operating on jobs being exchanged between several centers. During maintenance the production center is temporarily unavailable.
Two elements are essential for the definition of a distributed model: synchronization and member models or programs. The next paragraphs will explain this further.

SEQUENCING OF EVENTS

The synchronization to a common time-axis is essential for the definition of distributed simulation. Often the term 'parallel simulation' is used, but 'parallel' is more than just distributed. In this paper we restrict ourselves to discrete process simulation as an expansion of discrete event simulation [Veeke/Ottjes, 2000]. A process is defined as an ordered sequence of time intervals that connect all the events of one element (class). This means that all state changes are performed at discrete moments in time and at any moment there is one and only one element cq. process 'current'.
In distributed simulation this statement still holds. If only one element can be current at one moment, in a distributed model only one member model can be current at any moment or even better: only one element of all the member models can be current. This also means the sequencing of events occurring at the same moment in simulation is still a point of discussion [p.e. Wieland, 1999].
We hold to the following rules :

- The first event generated for a specific moment will be the first event handled at that moment.
- In case of state events, the state of the system will be checked after each event and not only when simulation time proceeds.
- In a distributed model environment the member model with the smallest event time receives control. If two member models have the same event time, the member model with the event first generated will receive control, unless specifically requested by the modeler.

Mark the word 'generated' instead of 'scheduled'. The moment of event-generation is essential in preserving the reproducibility of the simulaton runs.
From the above can be concluded that a distributed model synchronizes events to a common time-axis by sequencing the first events of member models. Member models are assumed to sequence their events to their own local time-axis.

According to these rules distributed simulation still is quasi-parallel and absolutely not parallel simulation, because nothing is running in parallel; so distributed simulation in this sense doesn't result in faster models (as a consequence of distributing the member models over different PC's). In this type of distributed simulation there's also no problem with look-ahead periods etc.
Finally distributed simulation should not be confused with real-time simulation. These definitions lead in fact to the conclusion that a combination of distributed and parallel simulation is needed to make it real-time.

DEFINING A DISTRIBUTED MODEL STRUCTURE

In our example of fig.1 we can make the model distributed in more than one way:

- put the generator and the lock (control included) in different models
- put de lock and the lock-control in different models
- add more locks with each an own control in different models (see fig. 2).

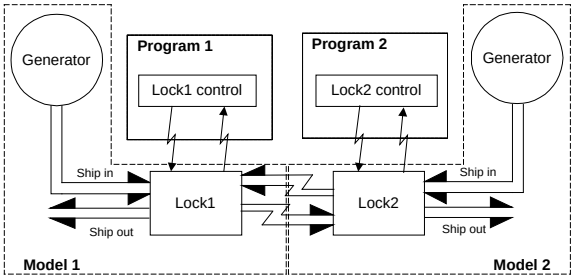

Fig. 2 Distributed 2-lock model

It completely depends on the goal of the simulation study, which of the distributed structures is the most appropriate. Studying different controls for the same lock, option 2 would be appropriate. If more locks with different attributes should be studied option 3 is appropriate. This automatically puts the question of how to structure a distributed model.
Defining a distributed structure from a stand-alone model is not a straight-forward task.
Starting from a stand-alone point of view, elements can easily be forgotten, because they are not considered elements there. For example, control functions are not always derived as separate elements, but mostly implemented as algorithms called by elements during the execution of their process. The problem becomes even more apparent when we consider the information flows between member models. In a stand-alone

model all information is available in local memory and one is often tempted to use it implicitly, especially in cases with a growing complexity of algorithms. Making a distributed model based on a stand-alone model then becomes a difficult - if not impossible-job.
Developing distributed models needs a systematic modeling approach, where member models appear according to well-defined structuring rules. Objective oriented modeling [Veeke/Ottjes, 2000] is such a modeling approach, based on the principles of systems theory [in 't Veld, 1998]. The approach starts and proceeds systematically with the goal of the design project, consistently distinguishes between operational and control functions and defines two particularizing principles: differentiation and specialization. Differentiation means a combination or division of functions, specialization refers to a combination or division of flows. However, the most important aspect of objective oriented modeling is the use of 'functions' (why) rather than 'tasks' (what).
From the function definition, processes can be derived which form the basis of each member model. The approach does start with a description of transformations the flow elements undergo, but
after that functions are derived by means of abstraction. These functions are minimally needed to reach a prescribed goal. This concept is necessary to find the right structuring (and is momentarily part of our research into aggregated modeling). Fig. 3 globally shows the steps to define a distributed structure.

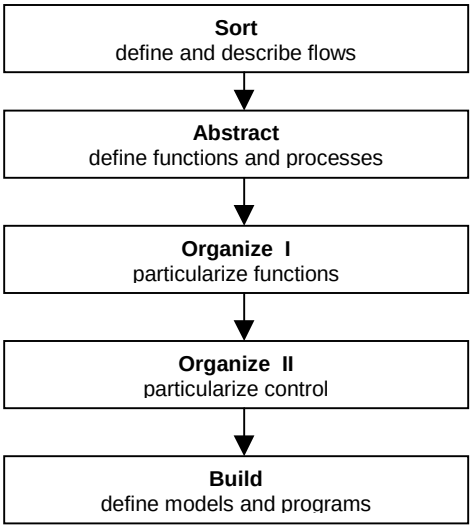

Fig. 3. Design steps for a distributed structure

Once the processes (or functions) and flows are defined and the structuring according to differentiation and specialization rules has taken place, the 'normal' approach [Veeke/Ottjes, 1999] can be applied to design member models in terms of elements and attributes.

IMPLEMENTATION

From the preceding paragraphs it follows that the next elements are necessary to create a distributed model:

1. a mechanism to sequence elements of a member model locally
2. a mechanism to sequence member models
3. a mechanism to exchange data and elements between member models and programs.

The local sequencing mechanism in each member model is a well-known discrete event mechanism for stand-alone simulation [Zeigler,1985]. In the earlier mentioned TOMAS platform the mechanism is extended to a discrete process mechanism, but the underlying concept is completely discrete event oriented. However, to connect it to the second mechanism some essential changes are required. They will be explained later.
The sequencing mechanism to synchronize member models can be considered a kind of "super sequencing mechanism". Each member model communicates its first future event to the mechanism and waits for allowance to perform the event. After allowance it becomes the 'current member', performs the event, communicates its next event and waits again.
From this point of view it is clear the mechanism performs the role of "time server" and therefore we've implemented it as a client-server concept.
All member models now have to address only one central server to become synchronized to the common time-axis.

Because each member can run on a different machine, the address is implemented
as being an IP-address. Messaging is based on the TCP/IP standard using basic Windows sockets. In
fact each member model can connect to the time server just by specifying the IP-address of the machine, the time server is running on. If no IP-address is specified the model is considered stand-alone.

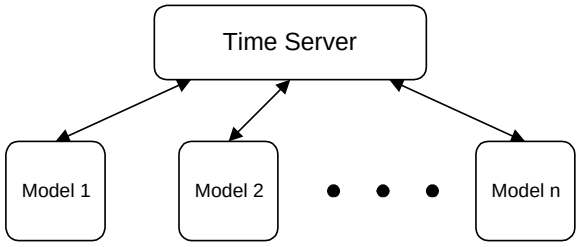

Fig. 4. Timer Server for synchronization

Besides the 'next event' message described above, the server mechanism must understand a 'start'- and a 'finish'- message to support the dynamics of the distributed structure. After a 'start'-message the server acknowledges its presence, makes the connection and communicates a unique model-identification to the member model. The current server time is also added to the message, so the member model will set its simulation clock correctly. A 'finish'-message disconnects the member model from the structure.
There is only one problem left to make this concept working. Each stand-alone discrete process/event simulation will end a simulation run automatically when the event chain becomes empty, because nothing will happen anymore. A member model however is not allowed to draw this conclusion autonomously. Each member model is assumed to communicate with other member models (why should it otherwise be part of the structure?). This means, that events

can be generated by other member models. So if the local event chain becomes empty, it still is possible future events for this member will be created by other members. Therefore a member model sends an 'empty'-message to the time server to express this situation.
If all member models have sent an 'empty'-message the time server finishes the distributed simulation completely by sending a 'finish'-message to all member models.
All messages described so far are transparent to the user of the model. They are a direct consequence of specifying the IP-address of the time server before starting the simulation run and implemented in the internal sequencing mechanism of the member models.
Member programs must be connected explicitly and therefore the methods 'ConnectToServer' and 'DisConnectFromServer' are available.

Inherently to the distributed concept member models, programs and Time Server must be able to communicate freely with each other. To support this the same messaging mechanism is being used, which leads to the general distributed concept of the
next figure.

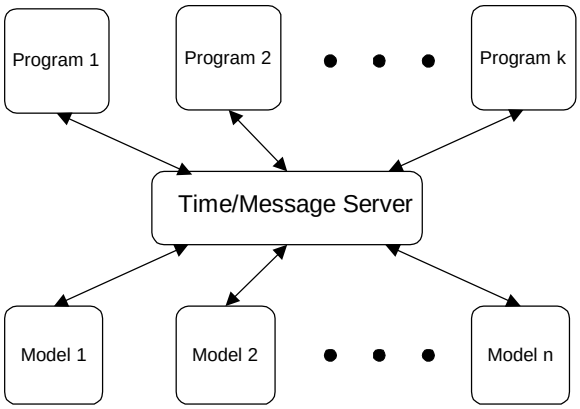

Fig. 5. Time/Message Server concept

First of all, all members must be able to communicate freely with the server. For example general information messages must be provided. Examples are requests about the simulation clock time, the wall clock time and a list of currently connected members. Requests can be made with the method 'SendMessageToServer(msg)'.
Member models and programs can communicate with each other by the method 'SendMessageToModel(model,msg)'. All messages still are being routed via the Server.
To receive and interpret messages each member must provide a receiving method. The local sequencing mechanism is the central reception point. It first checks if the received message is a standard message for the simulation. If not, it calls the receiving method provided by the user.
For reasons of ease-of-use some other methods are added, but they are not essential for the distributed concept.

VERIFICATION AND VALIDATION

The *verification* process must assure that the model is doing what it is supposed to do. In a stand-alone model this is normally achieved by tracing events and comparing simulation results with theoretic calculations. In a distributed environment message tracing becomes an essential part of the verification process. For this purpose a separate Message Trace module is added to each member and the time server to show the messages sent and received.

Distributed simulation influences (and can enhance) the *validation* process.
The authors are normally involved in projects, where a real system not even exists. So the validity of the simulation completely depends on the modeling approach being used. Recent papers addressed this issue in detail [Veeke, Ottjes, 1999-2000]. In distributed simulation the modeling approach plays an even more important role, because the modeled system must be divided into subsystems (member models and programs) beforehand. This was already shown in fig.3.
The validity can also be proven by incrementally replacing modeled elements by real (eventually scaled) equipment or machines. Especially control software will be tested in this way. At this point the interoperability becomes a topic. For that purpose the concept is now being made HLA-compliant (only the Time Server needs to comply to this standard, because the whole structure of members can be considered one model again!).

APPLICATION

The concept will now be applied to a structure with control elements. An application, where the concept is used for a structure of member models can be found in another submitted paper of this conference [Ottjes,Veeke,2001].
The application refers to a container terminal, where AGV's (Automatic Guided Vehicles) are used for the transportation of containers from quay to stacking area v.v. Major characteristic of this system is the high density of AGV's in a relatively small area, combined with high demands on the flexibility in AGV routings. Quaycranes regularly change position along the ships and dynamically block or release routing area. Traffic control is a key item for the capacity of the AGV-system. Prevention of collisions and deadlocks and the optimal use of the route layout (given the routes to the AGV's) are considered its main tasks. Because of the high costs of quay area no more space may be used for the AGV-system than is strictly necessary.
Quay cranes show large and unpredictable variations in operational speed. Because of that the system as a whole does not have a unique bottleneck. Sometimes the quaycranes determine the production capacity, sometimes the AGV-system and sometimes the stacking system. It is however not clear at what moment each system is the bottleneck. To answer that question it is necessary to be able to determine the real capacity of each system. For the quaycranes this is a rather simple question. For the AGV-system it is more complicated. Small changes in layout and traffic control can have enormous effects for the total capacity of the AGV-system.
The question is therefore to develop a simulation model to:

- determine the operational capacity for different layouts given a traffic control

- determine the operational capacity for different traffic controls given a routing layout.

The model should be developed in a way, that it can be used in a laboratory environment with real traffic control and scaled AGV's.
An AGV-system globally consists of 5 elements:
the set of AGV's, the area layout, traffic control, route assignment (routing) and job assignment. Navigation control is assumed to be a part of the AGV's themselves.
Because insight in operational capacities is asked, routing and job assignment will not be part of the model.
Instead, by running the model one must gain insight in the maximum throughput of the traffic system, so that routing and job assignment can be developed according to this knowledge.
The maximum number of AGV's through some route layout must be determined, so for each route the number of AGV's passing it is one of the important parameters.

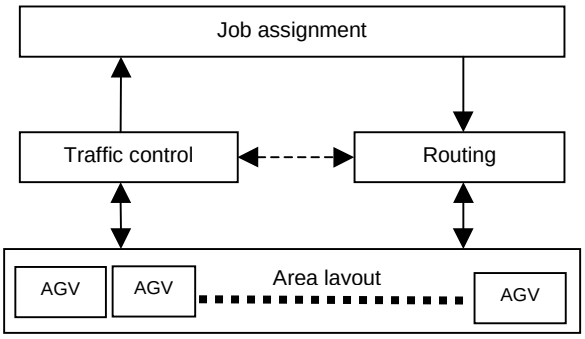

Fig. 6. AGV system with control functions.

One of the requirements is, to be able to connect real AGV's in the future. Modeled AGV's then still can be simulated in one single member model, but we have to make separate programs for layout definition (routes) and visualization. In this way single AGV's can communicate with these modules.
A route will be communicated to each modeled AGV, based on some kind of AGV arrival pattern for each route. To accomplish this, we need also a generator function and so the route definition program becomes a real member model.
Connection of real AGV's will be explained later.
If the level of AGV intelligence becomes an issue, we can also model this local control as a separate program. In this case we restrict the model to non-intelligent AGV's. So the structure of our distributed model looks like fig. 7.

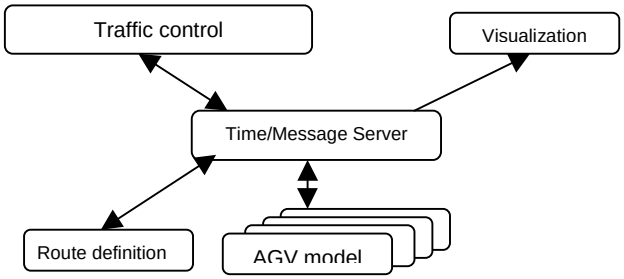

Fig. 7. Distributed model for the capacity model

Route definition is implemented as a interactive model with Delphi 5 and TOMAS. The user can draw route tracks or simply read a file with routes already defined.

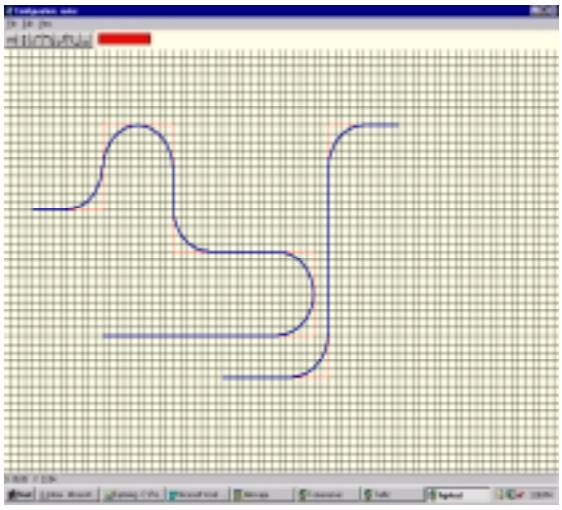

Fig. 8. Route definition window

Routes can be defined by selecting predefined technical movements of the AGV's. Theoretically perfect curves are available by default, but measured curves can also be entered consisting of (x,y)-coordinates and orientations, relative to the starting point. A route is composed as an ordered set of these route elements.
In this way a traffic situation can be created with parallel and crossing routes. To generate AGV traffic for this situation, the user can define arrival distributions of AGV's for each starting point of the routes. In this way the real traffic can be simulated as expected from quaycranes to stack etc. To determine the operational capacity however all kinds of distributions are made possible.

Fig. 9. AGV arrival patterns on a route

A special 'distribution' is specified as 'minimum'. In that case a next (modeled) AGV is generated at the moment the preceding AGV physically releases the source position. This can be considered the case with maximum AGV workload. Looking at the resulting arrival rate of AGV's at the endpoints of routes a good estimate of operational capacity is obtained.

THE AGV MODEL

To offer maximum flexibility in research goals, the modeled AGV is kept as simple as possible. AGV's are only assumed to be able to follow a given route. The route is communicated to the AGV model by the route definition program translated into a minimum number of relevant points. The AGV is able to translate this information into technical movements. The model uses a technical specification of an AGV based on data entered by the user.

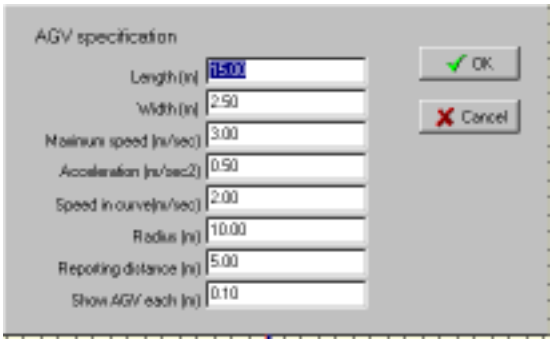

Fig. 10. AGV specification

Besides dimension and speed data, two parameters can be specified for the communication part. The 'reporting distance' specifies the interval length, after which the AGV informs traffic control its current position, orientation and speed. So this is the basis of the discrete image, that traffic control has of the current situation. On this image the decisions of traffic control will be based.
To facilitate a visualization of the system another interval can be specified. A small interval is used for smooth animation and a detailed view of collisions and deadlock situations, but increases the communication. Making the interval equal to the reporting interval, the visualization shows exactly what image traffic control has of the current situation.

TRAFFIC CONTROL

Traffic control controls the progress of all AGV's. AGV's are allowed to continue based on a general 'stop point' mechanism. After route definition has ordered the AGV model to generate a new AGV at some position, the AGV communicates its arrival to traffic control and tells it what route it will follow.
The AGV assumes its stop point to be its current position and waits for a message from traffic control. If traffic control decides, that the AGV's allowed to drive it calculates a position on the route, to which the AGV can safely move: the new stop point. This stop point is communicated to the AGV. The AGV reacts by accelerating to a feasible speed and stops at this point, unless meanwhile a new stop point has been received. It's up to traffic control to prevent collisions and deadlocks and too much acceleration or deceleration of AGV's.
By means of this mechanism many variations of traffic control can be implemented. Only communication must be based on stop points. It is even possible to let the AGV's drive freely, with no control at all. This is accomplished by sending a stop point equal to the endpoint of a route, at the moment of arrival of an AGV. This alternative can be used to investigate possible overlaps between defined routes and finding solutions for it. During the design phase, one always has the choice between a larger distance between routes or complicating the control.
This clearly shows the flexibility of a distributed model. Each member program or model can be changed without changing the other members, only the interface must be well-defined. In this case the interface mechanism with stop points is essential.

VISUALIZATION

The visualization program is kept apart from the AGV model, to support the visualization of real AGV's. During the first steps of development (with only modeled AGV's) it is not strictly necessary.
The visualization receives the route layout from route definition at the start of a simulation run. When an AGV is created it reports its position and orientation to the visualization program. As explained in the AGV model section this is repeated after each driven distance as specified in the AGV screen. Added to that the position and orientation are reported whenever an AGV comes to a standstill.
Visualization is normally used for two purposes:
a. presentation to the management
b. verification and validation.
AGV systems are perfect examples of models where visualization./animation is needed for verification and validation. Showing the AGV's makes it possible to check the correctness of traffic controls, because collisions can't be detected otherwise. Even deadlock situations can only be studied effectively if a visual image of the situation is available.

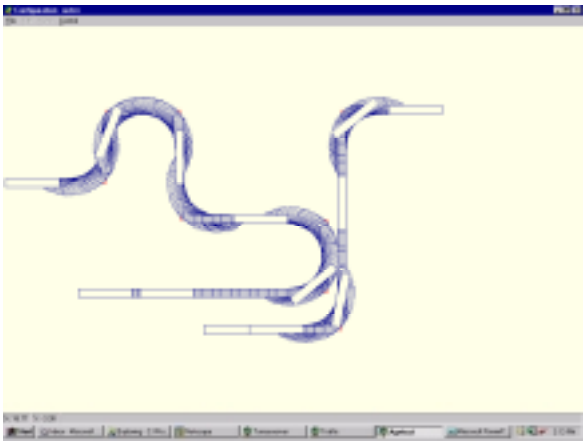

Fig. 11. Visualization

REAL AGV'S

Connecting real AGV's to this distributed model only becomes an issue at the moment traffic control and layout are well-defined. At that moment the primary goal of the model changes from research to validation and presentation. Real AGV's can't take part in the generation process of modeled AGV's, they will be permanently present in the simulation environment. Given a starting point of a real AGV, route

definition will be extended to generate routes for these AGV's based on a given layout. The only restriction in this case is, that the starting point of a real AGV must be a defined position on some route in the modeled layout. The server also needs to be able to run in real-time.

CONCLUSIONS AND FUTURE DEVELOPMENTS

In this paper a simple distributed simulation approach for research and education was presented, that offers many profits to the current simulation environments.
It enables a more 'natural' modeling of systems, that will be distributed in reality. For large projects it supports the division of simulation efforts and finally it extends the usability of software from the modeling environment to the prototyping and operational systems.
The concept is completely based on standard available communication mechanisms in Windows platforms.
The client server concept also makes it possible to do the research in an open and easy to use environment, while the complete distributed model still can comply to the HLA standard.
The concept was tested (and proved working) in internet sessions, where the time server program and a member model was running at Delft University and the other member models and programs in Wollongong, Australia.
Now the authors are experimenting with this concept in two ways:

- making a laboratory environment where a robotized production system will be connected to an AGV transportation system
- applying the concept to aggregated modeling, where the communication is used to connect global and detailed (zoomed) models. The process approach is already found to be the key item for these applications.

Finally, the simulation tool used "TOMAS" and a demo-version of the application described can be found on the website www.tomasweb.com [Veeke, Ottjes, 2000].

REFERENCES

Fujii, S. et al., 1999, "*Synchronization Mechanisms for Integration of Distributed Manufacturing Simulation Systems",* Simulation 72:3, pp.187-197, ISSN 0037-5497/99

Ottjes. J.A. and Veeke, H.P.M., 2001 *"experimenting with distributed modeling and simulation using the internet "*, submitted to ESM 2001, Prague.

Page, Ernest H. and Smith, Roger, 1998, "*Introduction to military training simulation: a guide for discrete event simulationists",* Proceedings of the 1998 Winter Simulation Conference (WSC 98), pp. 53-60, Washington, DC, 13-16 December 1998.

in 't Veld, Prof. J. 1998. *"Analysis of organisation problems"*. Educatieve Partners Nederland BV, 1998, ISBN 90 11 045947 (in Dutch)

Veeke, H.P.M. and Ottjes, J.A. 1999. *"Problem oriented modelling and simulation"*, Proc. 1999 Summer computer Simulation Conference (SCS'99*) Chicago, Illinois pp.110-114, ISBN 1-56555-173-7*

Veeke, Hans P.M., Jaap A. Ottjes, 2000. *"TomasWeb: web site:* www.tomasweb.com"

Veeke, Hans P.M., Jaap A. Ottjes, 2000. *"Tomas: Tool for Object-oriented Modelling 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, 2000, *"Objective Oriented Modelling"*. Proceedings of Industrial Systems 2000 (IS 2000), December 12-15, 2000, Wollongong. The International Computer Science Convention (ICSC).

Wieland, F., 1999, *"The Threshold of Event Simultaneity",* TRANSACTIONS of the Society for Computer Simulation International, Volume 16, Number 1, pp. 23-31, ISSN 0740-6797/99.

Zeigler, B.P., 1985. "*Theory of Modeling and Simulation*." Krieger, Malaba.