Proceedings of the Business and Industry Simulation Symposium (ASTC 2000).
April 2000. Washington D.C. [SCS]. ISBN 1-56555-199-0

TOMAS: TOOL FOR OBJECT-ORIENTED MODELING AND SIMULATION

Hans P.M. Veeke, 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, process-oriented, simulation tool, animation

SUMMARY

TOMAS is presented as a softwarepackage especially developed for discrete event simulation of complex control problems in logistic and production environments. Based on the experience, that the major part of simulation-effort on these systems consists of programming algoritms, TOMAS is implemented as a toolbox in the application-development environment of Delphi.
The design of TOMAS primarily focuses on problem oriented modelling (ease-of-use), communication with problem-owners, speed and flexibility.
A TOMAS model is described by means of the process-oriented approach, where processes are 'normal' object-methods instead of threads.
For verification and presentation purposes TOMAS supports full 3D-animation.
TOMAS is now being applied in the simulation of Automated Guided Vehicle-systems for containertransport and for operational scheduling in ERP-environments.
Finally, TOMAS will now be further developed to support distributed simulation and will be connected to qualitative modeling techniques.

1. INTRODUCTION

In this paper the backgrounds of TOMAS (Tool for Object-oriented Modeling And Simulation) will be explained.
TOMAS is a softwarepackage developed for discrete event simulation focused on complex control problems in logistic and production environments. This means the package must offer maximum flexibility in describing (often) unique control processes (par. 2). To support the designprocess of such complex systems, the way the model is described must closely connect to modeling techniques used in business and logistic management (par. 3).
To guarantee the model is working correctly, extensive facilities must be implemented for verifying the behaviour of the modeled processes. Not only tracing is needed for this purpose, but particularly for logistic simulations verification can only be done by means of animation. And of course, nowadays a simulationpackage isn't complete without full 3D-animation. So for presentation purposes this is also included in TOMAS (par.4).
Because TOMAS is meant to be used in extensive projects, it must support the possibilities to register experience and reuse this experience in future projects (par. 5).
Finally the need exists to implement the simulation model as the real control system. Therefor the model must be insensitive for the actual implementation of components; a component can be a simulation model on its own, but also a real piece of equipment. How this will be realized is discussed in par. 6. This article will explain the items discussed by means of a simple job-shop example.

2. FLEXIBILITY

Whenever simulation is being used to investigate complex control problems, the majority of time in the modeling phase is spent in just programming the algoritms.
Simulationpackages are often built as "closed" end-user applications, offering their own supporting language or a lacking interface to a general programming language [Kilgore and Healy, 1998]. These packages are normally based on visual modeling (click and define), and consider programming as an escape for non-standard cases. The projects under consideration here are however all special cases and thus rely completely on these escapes.
Furthermore, starting the modeling in a visual way and changing to a programming approach on a detailed level is difficult and the modeler has already lost the touch with definition details of the components in the model.
Modeling complex logistic or production projects will never be a standard problem; each project is unique in at least some of its aspects. Writing a good model for these cases demands full control over the modeled objects, method-contents and sequencing. For this reason TOMAS is defined at a third-generation programming-level. The price to be paid for this approach is, that modeling simple problems takes more time.
Looking for a general programming platform the choice seemed to point evidently to C++ or BASIC. However, BASIC is an unstructured language and extended models

(with often thousands of elements) can't reach the required speed. C++ on the other hand is widely used, offers all language features needed, but demands great programming skills and discipline. The last facets aren't normally found in an experimental environment of simulation. Simulation models often start with "global sketches" of the system, and under these circumstances extensive obligations on programming conventions only delay the modeling process. Considering this we've decided to develop TOMAS for an Inprise Delphi environment (Visual Pascal). This Pascal-implementation possesses a good object-oriented definition, consistent syntaxis and extended error-prevention and debugging facilities. This is the basis for building robust applications. Above that, Delphi is also widely used and offers all possibilities for interfacing to other packages and programming environments according to standard conventions.

3. MODELING TECHNIQUE

TOMAS offers discrete event simulation (Zeigler, 1985) based on the process approach. Normally the use of this approach is only argumented by the fact, that processes seem to be an 'intuitive' way of describing a system. This is true, but using process-descriptions also keeps the connection with business modeling. Business or enterprise models are nowadays almost always process-oriented (p.e. BPR). Process orientation is therefor necessary.
As a starting-point of modeling in TOMAS, systems theory [in 't Veld,1998] is being used. In systems theory the entities are called 'elements' and the theory makes a distinction between active or 'processing' elements (processelements) and passive or 'to be processed' elements (flowelements). However, cases can be easily found, where flowelements perform part of a process themselves, so TOMAS only defines a general class '*TomasElement*'. Specific elements can be defined as descendants of a TomasElement, and each TomasElement may or may not have a process-method.
A definition-example is shown in the next code, where a machine-, a job- and a generator-object are being defined. By overriding the default process-method of a TomasElement, the MachineClass and GeneratorClass are defined as *active* elements.

In the initialization-part of the module (or activated by a button-press on the user's form), the modeler can create machines and a generator and activate them. Here we reach the point where some explanation is needed about the sequencing-mechanism of TOMAS. The process-approach results in the generation of events during process-execution. As can be seen in the above example, processes are just normal object-methods of a class. To achieve parallel process-execution, other simulationtools use the thread-mechanism of Windows (Healy and Kilgore, 1998). Threads can perform well, but the number of threads is limited. Increasing the number of threads degrades runtime speed significantly.

```
Type
  MachineClass = class;
  JobClass = class;
  GeneratorClass = class;

  MachineClass = class(TomasElement)
     Job: JobClass;
     JobList: TomasQueue;
     Published
     Constructor Create(Name: String);
     Procedure Process; Override;
     End;

  JobClass = class(TomasElement)
     Duration: Double;
     Constructor Create(Name: String; Duration: Double);
     End;

  GeneratorClass = class(TomasElement)
    ArrivalDis: Distribution;
    DurationDis: Distribution;
    Published
    Constructor Create(Name: String);
    Procedure Process; Override;
    End;
```

Example 1. Definition of elementclasses

Therefor TOMAS uses a programmed sequencing mechanism, that is able to collect events and *activating the programming code at any point within a method*. By using this principle TOMAS models are extremely fast [Veeke and Ottjes, 1999]. To achieve this two kind of methods are provided:

a. *Methods of the current process* (implicit TOMAS-object). The next table explains some examples.

Method	Meaning
Hold(T)	Wait/Work for T time-units
Standby	Wait for condition to become TRUE
Passivate	Wait
Terminate	Terminate the process

By calling these methods from within a process-method of a TomasElement, programcontrol returns to the sequencing-mechanism and another process is selected to become 'current'. All these methods tell the sequencing mechanism that there will be no state-change for some time as far as the current process is concerned.

b. *Methods of a TomasElement*. These methods control the initialization, continuation and cancelling of an element's process.

Method	Meaning
StartProcess(T)	Start execution of the process-method at time T
ResumeProcess(T)	Resume execution of the process-method at Time T
InterruptProcess	Interrupt execution of the process-method now, saving the resttime
CancelProcess	Cancel execution of the process-method now

Returning to the example of the simple jobshop the processes of a machine and a generator may look like example 2.

```
Procedure MachineClass.Process;
  Begin
  While TRUE Do
    Begin
    While JobList.Length = 0 Do
      Begin
      StandBy;          {wait until job available}
      End;
    Job:=JobList.GetFirstElement;
    JobList.Remove(Job);
    Hold(Job.Duration);
    Job.Destroy;
    End;
  End;    {MachineClass.Process}
```

```
Procedure GeneratorClass.Process;
  Var
    GenJob: JobClass;
  Begin
  While TRUE Do
    Begin
    Hold(ArrivalDis.Sample);
    GenJob:=JobClass.Create('Job',DurationDis.Sample);
    Machine.JobList.AddToTail(GenJob);
    End;
  End;    {GeneratorClass.Process}
```

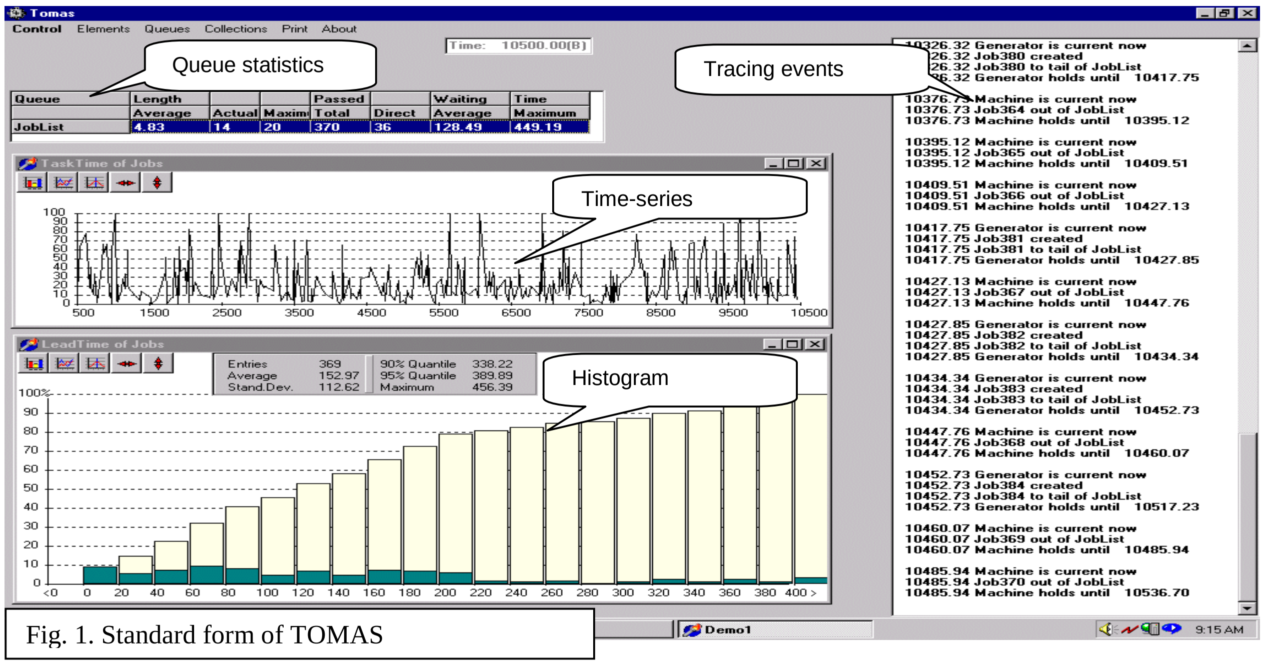
Example 2. Process-descriptions

From the examples it becomes also clear, that TOMAS contains the usual queuing-objects (called TomasQueues) to model waitinglines easily and to generate automatically statistical data on waiting-times, leadtimes and occupation. These are the kernel-objects and methods of TOMAS.

Added to it are of course the complete functionalities of a Delphi-environment. The modeler is able to define his/her own forms to visualize whatever is needed. TOMAS is implemented as a toolbox, so simulation can be made part of any Delphi-application. At this moment a TOMAS-simulation is being incorporated into an ERP-environment. The simulationmodel is used therein for the detailed shopfloor-scheduling.

4. VERIFICATION AND 3D-ANIMATION

To check the correct working of the model TOMAS offers a number of options (see fig. 1).



Queue	Length			Passed		Waiting	Time
	Average	Actual	Maxim	Total	Direct	Average	Maximum
JobList	4.83	14	20	370	36	128.49	449.19

Entries	369	90% Quantile	338.22
Average	152.97	95% Quantile	389.89
Stand.Dev.	112.62	Maximum	456.39

```
10326.32 Generator is current now
   26.32 Job380 created
   26.32 Job380 to tail of JobList
    6.32 Generator holds until   10417.75

10376.7  Machine is current now
10376.73 Job364 out of JobList
10376.73 Machine holds until   10395.12

10395.12 Machine is current now
10395.12 Job365 out of JobList
10395.12 Machine holds until   10409.51

10409.51 Machine is current now
10409.51 Job366 out of JobList
10409.51 Machine holds until   10427.13

10417.75 Generator is current now
10417.75 Job381 created
10417.75 Job381 to tail of JobList
10417.75 Generator holds until   10427.85

10427.13 Machine is current now
10427.13 Job367 out of JobList
10427.13 Machine holds until   10447.76

10427.85 Generator is current now
10427.85 Job382 created
10427.85 Job382 to tail of JobList
10427.85 Generator holds until   10434.34

10434.34 Generator is current now
10434.34 Job383 created
10434.34 Job383 to tail of JobList
10434.34 Generator holds until   10452.73

10447.76 Machine is current now
10447.76 Job368 out of JobList
10447.76 Machine holds until   10460.07

10452.73 Generator is current now
10452.73 Job384 created
10452.73 Job384 to tail of JobList
10452.73 Generator holds until   10517.23

10460.07 Machine is current now
10460.07 Job369 out of JobList
10460.07 Machine holds until   10485.94

10485.94 Machine is current now
10485.94 Job370 out of JobList
10485.94 Machine holds until   10536.70
```

Fig. 1. Standard form of TOMAS

First a complete trace of events in the simulation can be shown. All events that change the state of a process or change the contents of queues are added automatically. Furthermore, the user can add his own checking by means of the 'writetrace' method of a TomasElement.
For each queue TOMAS gathers statistical data, so verification by global calculation is easy.
Also, for each TomasElement the history of events is being collected to support the verification at the process-level of the element. And finally user-defined items can be collected by means of a TomasCollection-object, that can be represented as a graphical time-series or as a histogram. All data can be exported to standard applications as Excel, Access etc.

For logistic systems verification should support at least 2D-animation. For example to verify two vehicles don't collide, a visual representation is needed. The drawing facilities of Delphi already provide the means for this kind of verification.

presentation tool, speed requirements fall within 1 to 5 times realtime-speed.
Speed is generally accomplished by optimizing the refresh rate of the screen. Knowing this, TOMAS implicitly has a big advantage on general animation environments.
For example, when 5 vehicles change position at the same eventtime, the screen must be refreshed only once. Therefor TOMAS refreshes the screen only once per eventtime at the most. By implementing the refreshrate as a function of event-interval, speeding up an animation is becoming very simple. All rendering methods in TOMAS are based on a Graphical Component TGMP (Dove and Peer,1997). The TGMP-component has been optimized according to the above-mentioned refresh-mechanism. Until now no OpenGl or DirectX-interface is needed, because quality and speed seem to be sufficient even in extended models.
The TomasAppearance-class is a descendant of the TomasElementClass, with animation-attributes and –methods added to it.

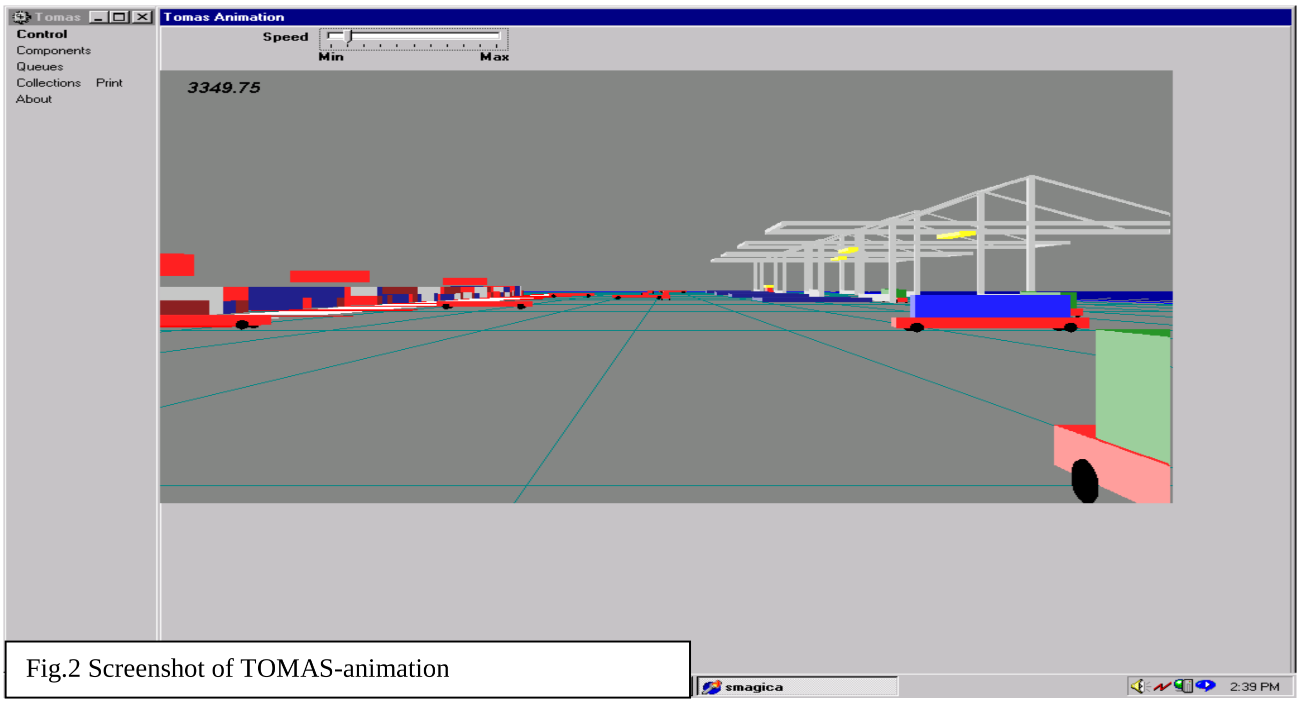


Fig.2 Screenshot of TOMAS-animation

For presentation purposes however 3D-animation is nowadays required. This kind of animation combines the visual verification and presentation-needs. The implemented animationtools into TOMAS are based on the general principles of speed and ease-of-use.
Maximum speed however is not always necessary. Using animation as a verification tool, the speed of the animation must even be relatively slow to assure, that the user is able to conceive all the details. Using animation as a

A TomasAppearance is described by means of geometrical shapes. Standard shapes as boxes, spheres and wheels are predefined; general shapes can be constructed from wireframe-definitions. Besides the methods S*how* and *Hide* there are other methods to define movement such as *MoveTo* and *Bend2D*. Several rendering modes such as *wireframe* and *solidshade* are available.
For animationpurposes a standard Camera-object is available, that can be programmed as well as moved interactively during simulation. It can also be attached to

one of the animated components, so a high degree of 'virtual reality' can be reached by simple programming or manipulating. In fig.2 a screenshot is shown of a containerterminal with Automatic Guided Vehicles (AGV's), where the camera is connected to one of the AGV's.

5. STANDARD ELEMENTS

To support the reuse or generalisations of once defined elements, the Delphi-environment offers a perfect option by means of its component-library. TOMAS can now be easily extended with two possibilities:

a. Equipment, specific for a company, must be described (programmed) only once and is then added to the library for future reuse in other simulation-models.
b. From the TOMAS-development point-of-view, generic elements can be defined. At this moment a TomasResource is already available, which is a generalization of the MachineClass in example 1. Defining Machine as a TomasResource the process-method of the machine isn't needed anymore.

```
Procedure GeneratorClass.Process;
  Var
    GenJob: JobClass;
  Begin
  While TRUE Do
    Begin
    Hold(ArrivalDis.Sample);
    GenJob:=JobClass.Create('Job',DurationDis.Sample);
    Machine.Claim(GenJob,1,GenJob.GetDuration);
    End;
  End;    {GeneratorClass.Process}
…
{define the resource with minimum capacity 0,
  maximum capacity 1 and the FIFO-selectionmethod}
Machine:=TomasResource.Create('Machine,0,1,FIFOSelect);
…
```

Example 3. Use of Resources

Example 3 shows the process of the Generatorclass, which is now sufficient to describe the jobshop-model. After the process-method the create-statement for the machine is illustrated. This also illustrates the flexibility of TOMAS, because in the creation the selection-method for jobs in the JobList of the resource is defined. In this case a standard method 'FIFOSelect' is chosen, but the modeler is free to attach here a complex user-written method.
Besides the TomasResource another standard element is the TomasSemaphore. Contrary to the TomasResource, a TomasSemaphore doesn't possess its own process-method, but is able to suspend TomasElements competing for the same limited capacity.

Remember, it always remains possible to create descendants of these standardelements with their own attributes and process-method.

6. CURRENT AND FUTURE DEVELOPMENTS

At this moment the first implementation for distributed modeling is available. As can be seen in another presentation of this conference (Ottjes) an agent-based jobshopmodel and a critical-path model are being combined into one synchronised simulation-model. In this case the critical-path model is completely controlled by the jobshopmodel. For general distributed modeling TOMAS now consists of a server-application, that synchronizes several client-models. Communication between the models is for the time-being based on the standard Windows 'mailslots'. It's a primitive way of communication, restricted to local networks, but extremely dynamic: just exchange strings of data. Before using advanced and standardized communication-interfaces like CORBA, we first want to define the communication-needs in detail. It already became clear, that a small number of messages with only a few data are necessary to facilitate full-distributed simulation. In fact, only messages for the process-sequencing methods as defined in par. 3, are necessary.
The next step will be the definition of an interfacelayer to make the server-application independent of the client-applications. It must be of no concern, either the client-application is a real piece of equipment or a simulationmodel.
Parallel to these developments, research is being done to combine qualitative business/logistic modeling and quantitative modeling with simulation. In simulation literature little can be found on the theory of modeling. What is called modeling refers almost always to the technique of simulation. Little is said about the way to define elements and processes derived from an initial problem-statement. Systems theory however is such a qualitative modeling theory. The world of simulation is primarily focused on the question: 'Is the model good?' (verification), but seldom the question: 'Is this a good model?' (validation) is fully answered. Therefor the development of a software package PROMOTE (PROcess Modeling Tool for Engineering) is started, that supports systems theory and should finally generate simulation-models (with TOMAS) automatically, thereby guaranteeing the correct translation of problem to model.

REFERENCES

Kilgore, R. and K. Healy, 1998. “JAVA, Enterprise Simulation and the Silk simulation language.” *Proceedings of the 1998 International Conference on Web-based Modeling & Simulation*. SCS, San Diego CA.

Zeigler, B.P., 1985. “Thery of Modeling and Simulation.” Krieger, Malaba.

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

Healy, K. and R. Kilgore, 1998. “Introduction to Silk and Java-based Simulation”. Proceedings of the 1998 Winter Simulation Conference, IEEE, Piscataway, NJ.

Veeke, H.P.M. and J.A. Ottjes, 1999. *“Problem oriented modeling and simulation.”* Proceedings of the 1999 Summer Computer Simulation Conference , Chicago, Illinois, ISBN #1-56555-173-7.

Dove, P. and D. Peer, 1997.”Getting DIBS on speed.” Delphi Informant, Vol. 3 Nr. 4 (April).