A Highly Interactive Discrete Event Simulator designed for Systems in Logistics

A. Graber, D.Mutaaga, Dr. H. Ulrich, D. Schweizer *, A. Zimmermann
Institute for Operations Research
* Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology Zurich

F. Bolay,
Alcatel-STR, Au-Wädenswil, Switzerland

Abstract

Simulation is considered to be a method to improve planning and control efficiency in the field of logistics. In our view no existing simulation tool meets the basic requirements sufficiently. We therefore developed a new simulator with the following characteristics:

- Simulation principles: General purpose discrete event simulator.
- Modeling basis: Meta model comprising few specific basic elements to be used according to a well-defined syntactical structure.
- Modeling principle: A separation of material flow and information flow is enforced. Logically consistent hierarchical structures are provided.
- Implementation: Realization in an object-oriented language (Smalltalk-80)
- Working environment: Highly interactive modeling process to be performed in a man-machine-dialogue. Flexible visualization of the simulation model and the simulation results are provided.

A brief summary of our experiences in simulation is given. Of our approach to develop an adequate simulator, we describe some basic elements as well as one specific application.

1. Introduction

Mastering the complexity of large dynamic systems is a major challenge in the management of today's economy. Generally simulation is considered as a promising approach to improve planning and to control efficiency. But, in spite of single successful examples, simulation is not yet the state of the art for the planning and control task. The main reason is still the cost-efficiency relation for simulation projects which only in rare cases proves favorable.

The tendency over the last years has been to develop simulation tools with increasing specialization to specific application fields. But if the simulation tools are not explicitly conceived for the problem to be solved -- this is unfortunately the normal case --, the results are in general unsatisfactory. There is a lack of flexibility within these tools and the run-time performance is often not acceptable.

The requirements for a simulation tool depend on the specific field of application . The latter can be characterized as follows:

Dynamic aspects in logistics, e.g., material flow in production or transportation networks. The analysis is concentrated on the decision problems represented on an abstraction level that does not consider a visual representation of the real process or lay-out details.

For this application we aim to develop a simulator which efficiently supports the modeling task and allows a simulation analysis for a problem dimension relevant in practice with an attractive run-time performance.

We are not aware of such a simulator, however we found a number of attractive elements in different existing tools on which we could base our approach. Worth mentioning are the simple abstract simulation elements of simulation languages as in SIMAN and SLAM or interactive facilities of a Petri Net Simulator as in PACE [4], [5].

2. Requirements

The requirements for a simulator envisaged here focus on two major topics:

1. The power of the *modeling basis* to support the model building process.

2. The quality of the *working environment* on computers which has to enhance programming and the subsequent simulation analysis.

- The *modeling basis* should comprise of a formalism providing elements conceived especially for our problem field. These elements are to be designed as simple as possible. The user can either modify these elements in their behavior or join them to more complex modules, according to his requirements.
- The user should not be limited in his choice of an adequate abstraction level. He even should be enabled to alter this level during the modeling process. A top-down approach with a stepwise increase in degree of detail is efficiently supported.

• Debugging facilities as a fast detection and indication of logical programming errors as well as immediate visualization of the model in the programming process facilitate the modeling task.

• As to the *working environment*, the simulator should be conceived for a hardware platform equivalent to standard modern work-stations. There, a highly interactive modeling in a man-machine dialogue can be established.

• The simulator should be designed to cope with system dimensions relevant to the often huge real world problems. As to the run-time performance, the results should be attainable within a 'reasonable time'.

• An easy interpretation of the results should be enabled through the availability of utilities for statistical purposes, supported by graphical representation facilities.

3. Experiences with some Widely Used Methods

3.1. Overview

Today's modeling and simulation techniques -- and the associated tools -- are still in a very unsatisfactory intermediate state [11]. The huge number of available tools is an indicator, for a not yet established methodology.

One can see the following two main categories (taking into account, that not all methodologies fit into this scheme):

• General purpose simulators that are based on systems like GPSS, SLAM, SIMAN and Simscript.

• Module oriented, application specific simulators like e.g.: Simple++, Dosimis, Automod II, STEM, and many others. The roots of those tools are mainly a parametric system, combined with new graphical oriented SW user-interface.

The general purpose systems contain -- due to their age – a huge amount of knowledge in modeling and development of efficient algorithms for the implementation of the simulator. Due to their SW-technology, they are presently used less in the academic world.

We summarize the relation between those two categories in the following figure:

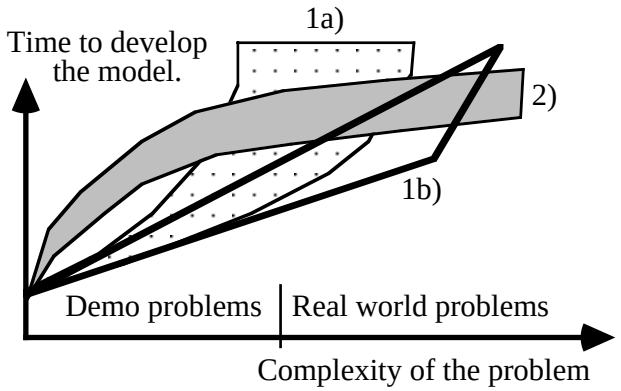

Fig. 1: Development - time versus problem - complexity.

1) Development time with module oriented simulators.

1a) Practical behavior. A large amount of time is used to *bend* the modules for the needed functionality. Real world problems are often not feasible.

1b) The module fits exactly the needs. This behavior is seldomly reached.

2) General purpose simulators tend to have a longer introduction phase, but are more suitable to tackle large scale problems.

Due to the fact, that a module oriented simulator always lags behind the recent problems, the end users in practice are faced with characteristics 1a (after they expected to have bought the powerful tool of characteristics 1b).

Run-time performance:
The typically available computer system for solving logistics problems is a high end PC or a workstation. Fig. 1 does not make any statement about the run-time efficiency of the two main classes of simulators.
Even considering the increasing power of computers, one will always be faced with the problem of performance.

3.2. Criteria for judging different approaches

Our judgment is based on large scale systems like the entire cargo traffic of Swiss railways' network [12] or the flow of material and information of three coupled fully automatic factories in the chemical industry.

In such projects one needs flexibility:

1. to find the right level of abstraction (which might change during the life-cycle of the model),
2. to include external control systems,
3. to enter original data from MRP systems,
4. to get simulation results fast. One run of significant length must have a run-time clearly below 30 minutes, otherwise we loose the benefits of repetitive simulation runs.

To achieve good results, it has to be possible to perform many repetitive runs

3.3. General Purpose Simulators

Benefits:

• Elementary medium level simulation instructions, without limitations in adapting the abstraction level to the actual problem.

• Through the possibility of including procedural code (C, Pascal, ..) one is enabled to implement complex control systems and procedures for input and output data in any format.

• General purpose simulators can adapt their abstraction level to the needs of the user and herewith help to save computing power.

• As to run-time, this approach is very efficient (simulation 50 - 100 times faster than in reality).

Problems:

For the implementation of a model, this approach definitely needs skilled people with expert knowledge in computer science. The initial learning time cannot be neglected.

3.4. Module oriented Simulation

Benefits:

• Very short initial learning time, with immediate results in the first stage of partial modeling.

• The graphical user interface -- often combined with an on-line animation -- is highly motivating for novice users.

Problems:

• As the project evolves from the demo model to the final model, special unforeseen situations may consume a huge amount of time (which is the normal situation in many projects).

• Due to the given set of modules one is forced to model in detail, whereas a higher abstraction level is indicated. This tends to consume an unacceptable amount of cpu-time (simulation is slower than reality) and therefore frequent repetitive simulation runs are hardly possible.

3.5. Petri nets

Our first approach to develop a general purpose simulator, was based on the interactive Petri net tool PACE [4], [5] for modeling logistics systems in an industrial environment. Below we shall outline some benefits and problems we encountered while using this tool.

Benefits:

• Petri nets [6], [7], [8] provide a sound mathematical foundation for modeling and simulation.

• The basic semantics of Petri nets is extremely easy to understand and their representation as bipartite graphs leads to an immediate intuitive understanding of what is happening on the level of particular nodes such as places and transitions.

Problems:

• Although Petri nets have a simple semantics, it is not easy to understand a large system represented in terms of Petri nets. Two projects[1)2)] led to the insight that the abstraction mechanisms provided by hierarchical Petri nets were too weak to represent the structure of the systems appropriately.

• "Hierarchical Petri nets" as they were provided by the tool allow to represent parts of a large net as a single box. Herewith you get a different view of the net with arbitrary chosen "black boxes" that do not behave as a place or a transition of a Petri net and therefore get only a meaning if you know exactly what's happening inside the boxes. Hence there is no real abstraction mechanism.

• Based on the wish to get more modeling power by abstraction, there was an attempt to provide a set of powerful subnets (modules) that may be used to compose arbitrary models [9]. This approach was dismissed for the following reasons:

- The semantics of the new modules is only defined through their internal structure (and of course informally through the names of its interfaces and its own name). Thus the behavior of a module cannot really be understood without a detailed knowledge of its internal structure, and therefore, the benefit of the essentially simple semantics of Petri nets gets lost.

- With the creation of new modules one virtually introduces a new semantic. Additional consistency checks on these new modules may lead to a simulation - overhead of up to 65% [9].

4. Our Simulation Concept

4.1. Simulation Methodology

In our context a simulation analysis comprises problem solving and model development as two highly correlated processes. Therefore, an iterative development process seems to be the most appropriate procedure. We therefore propose the following steps:

1. Decomposition of the system into single processes on an abstraction level as high as possible, in a way that the relevant features of the system can be represented adequately.

---

1) Control system for HYBRID III automotive research platform

2) Production planning for printed circuit boards

2. Run the simulation and use the results to find potentially critical parts of the system, that are not covered sufficiently by the chosen abstraction level.

3. Refine the processes identified in step 2 to a more detailed abstraction level.

See also [10].

This approach leads only to good results if an expert, familiar with our class of systems is in charge of the modeling task. He should be able to estimate the most relevant system parameters. A successful application of this methodology requires a highly interactive modeling environment, providing immediately understandable results (graphics) in a short time.

4.2. The Meta - Model

The term *Meta - Model* designates a formalism suitable for the description of a *model* of a real world system.
The use of Petri nets for the description and simulation of logistics systems inspired us to provide a model based on a sound formal description, omitting the already mentioned drawbacks of Petri nets. The definition of the basic elements of our meta - model is based on our experiences in the use of established simulation languages like SIMAN and SLAM.

4.2.1. Design Guidelines

In our approach we tried to meet the following requirements:

- The meta - model has to provide a syntactical structure for the description of logistics systems we deal with. Although the nodes in our net should provide a higher level of abstraction than places and transitions in a Petri net, the number of different types of nodes should be as small as possible. People familiar with queueing systems should be able to understand their meaning intuitively. The syntax is to be checked incrementally during the editing session.

- A simple graphical representation of all possible language constructs has to exist.

- The meta - model itself should provide hierarchical structures in the sense that a node of level n + 1 contains a subnet of level n. Every node that doesn't contain a subnet has a default behavior. To encourage a stepwise refinement process, subsequent modifications of a node by describing its behavior with a subnet does not change its interfaces. The hierarchy relation *H* means that if *a H b* holds, then the behavior of a node *a* depends on the behavior of a node *b*: N designates the set of all nodes; S_i the set of all nodes on level i.

$$\forall a \in N: a \in S_i \wedge i > 0 \Rightarrow$$
$$\exists b \in N: aHb \wedge b \in S_j \wedge 0 \leq j < i$$

There is a subset S_0 (level 0) of the set of all nodes N such that

$$\forall a \in N: a \in S_0 \Rightarrow \neg(\exists b \in N: aHb)$$

The relation H is transitive, irreflexive and anti symmetric. See also [1], [2].

- Considering that a simulation model must be able to reflect the real structure of a system, it should provide also an appropriate abstraction for the edges of the net. Here we distinguish at least two different types of edges in the net describing the material and the information flow respectively [3].

- The proposed meta - model should provide mechanisms flexible enough to describe any real structure of a queueing system in a consistent way. Features to extend the functionality of particular nodes are provided. The syntax of the meta - model should not be affected by these extensions. This behavioral description on a rather low level of abstraction could be made in terms of Petri nets, state machines or any suitable language.

- The information flow has to be separated strictly from the material flow. This offers the possibility to run a model under different control strategies.

4.2.2. Syntactical Structure and Semantics

In the following section, we present a brief description of the basic elements of the meta - model followed by a short formal definition of the syntactical structure (only the static parts, i.e., without resource allocation). A complete formal description may be obtained from the authors.

Every model will be represented as a directed graph with three different types of nodes and two types of edges.

We propose the following types of nodes:

- **Stations** may be hierarchical structures in the sense mentioned above (a *station* may contain a subnet, which defines its behavior). They model an arbitrary process. Every *station* has a default behavior to allow a simulation run at any time. *Stations* can always generate, duplicate, split, modify and remove entities.

- **Logic elements**. These elements can inspect other nodes in the net as well as modify their state. All of these interactions are well defined through the definition of channels enabling information exchanges (information flow). The behavior of these *logic elements* may be specified in a formalism mentioned in the previous section.

- **Queues.** The queueing rules are configurable either statically or dynamically through commands received from connected *logic elements* mentioned above.

One may consider these nodes as a set of abstract data types. Their internal behavior could be defined in terms of an algebraic specification.

Furthermore we have chosen the following types of edges:

- **material flow channels** allow entities to move from one node to another along the edge.

• **information flow channels** allow state inspection or modification between connected elements in a well-defined manner. No movement of entities is allowed over these edges.

In addition there are **resources** which can be allocated dynamically. They contain information about their availability and their behavior.

5. The Implementation

5.1. Object Oriented Approach

We implemented our tool in Smalltalk-80 because:

• Object-Oriented Programming is suitable for developing simulations for all kinds of applications, using all kinds of conceptual frameworks [13].

• In an object oriented environment it is easier to implement abstract modeling elements. "World is composed of 'objects'" [14].

• The sending of messages to objects corresponds to the concept of event-oriented simulation.

• Two major demands of the user to a simulation program are the ease in its use and an all-powerful functionality. With Object-Oriented Programming it is possible to aim at both goals.

• In particular we have chosen Smalltalk-80 because of the very comfortable programming environment.

5.2. Implementation of the Meta - Model

When constructing models we use an interactive graphical interface, where elements are added to the network by clicking on icons and dragging them. Connecting two elements defines a material or an information flow between them. When adding a new connection, the program checks automatically all syntactic rules, so that the user gets aware of errors while modeling.
For each *station*, a subnet can be defined. It will appear in a separate window.
We distinguish between functionality and graphical information by dividing the implementation model into two levels: the simulation model, with the data specific for the simulation, and graphical models, with the information for displaying the network.

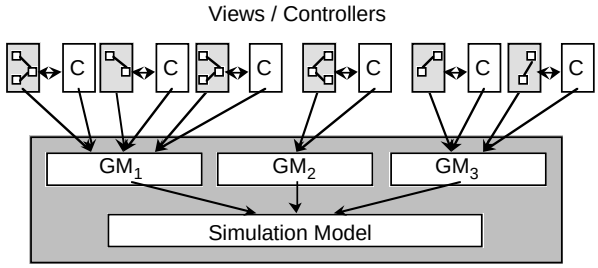

Fig. 2: Implementation model consisting of the simulation model and three different graphical models GM1, GM2, GM3.

• The simulation model contains the functional description of the process to be modeled. It consists of the main network, subnets and specific resources for the model. A special list contains the descriptions of each type of entity.

• The graphical model contains information about the appearance and location of the nodes as well as methods to manipulate them. Several graphical models can be superposed on the main network or any of its subnets.

• Views are used to display the graphical models.

For monitoring the simulation, the model and the data can be managed and visualized in several different ways.

5.3. The Net Elements

We construct our models using a network, subnets, entities and resources.

We implicitly fix the topology of the net as well as the predecessor successor relationships between the net elements by defining the flow of material between the net elements.

Entities (or tokens) are objects (materials, clients, orders) which traverse the net. Each movement of an entity in the net causes a change of its state. These changes are recorded in an event list which in turn is executed by the scheduler. Entities have system specific attributes, e.g., identification, source position, destination position, current position, arrival time, etc., as well as user defined attributes, e.g., type, occupied resources, expected signal, etc.

Resources are abstract structures with the ability to autonomously decide about their availability. Resources have the following user definable attributes: capacity, usage level, maximum and minimum values; as well as system specific attributes, e.g., which *queues* contain entities awaiting allocation, which entities occupy the resource; location and reserved flag (in case of movable resources). Allocation and deallocation can be performed using default mechanisms or user definable subnets.

The blockage of entities in a *queue* depends on the actual state of the network.

The net elements (*queues*, *stations*) and resources provide an interface for data collection. We have defined a set of *statistic elements* that may be connected to those interfaces. They range from a simple digital or analog display up to bar graphs, pie charts, panels to more sophisticated elements like plots and gant charts.

5.4. The Simulation

The diagram below shows the implicit flow of information which in return triggers the flow of material. The standard sequence of events described here is invisible to the user:

1) Entity e arrives at *queue* Q_1

2) *Queue* Q_1 sends a message to *station* S_1 indicating the arrival of entity e

3) *Station* S_1 gives control to *logic element* L_1

4) *Logic element* L_1 checks the relevant system state and sends a message back to *queue* Q_1 indicating whether entity e can be forwarded to *station* S_1 or not

5) Entity e arrives at *station* S_1. *Logic element* L_1 controls the actions to be performed. If the *station* has a defined subnet, the entity is forwarded to the first *station* of this subnet. After reaching the last *station* of the subnet the entity returns to *station* S_1

6) *Logic element* L_1 decides whether entity e should be disposed of or forwarded to Q_2 or Q_3.

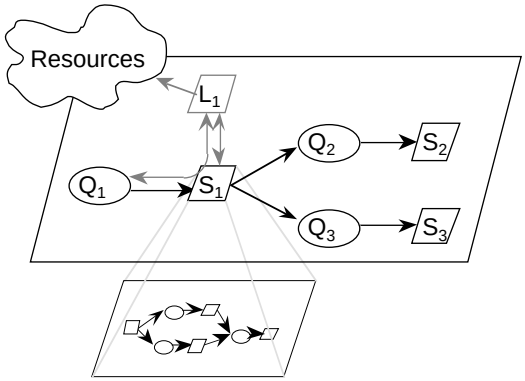

Fig. 3: The network with a subnet, showing the flow of material and information.

6 . Applications:

We carried out the following project with Alcatel-STR, a large manufacturing enterprise in the field of telecommunications:
The requirement planning of financial resources has to consider the time dependent variation of their demand. The material flow in production is the basic cause of this variation. By simulating this material flow and analyzing the related flow of financial means, it is possible to improve this planning process considerably.

Basic data:

- Data of the Production Planning and Control System (PPC). This Data includes among others a description of all products and their components. As this information is normally very detailed, its aggregation was necessary.

- Distribution, availability, maximum capacity and minimum threshold values of all kinds of resources.

Parameters:
- Ordering policy of different raw materials
- Delivery program for each product
- Current rate of inflation.

With these parameters, we simulate the different accounts thus providing insight into the cash flow within the enterprise. We further investigate the exploitation of the available resources and its influence on the cash flow.

By making use of the hierarchy concept of our simulator, we use stepwise refinement to model any necessary details of the enterprise.

7 . Conclusions

For the simulation of systems in logistics, no existing simulator could meet our basic requirements sufficiently. HIDES, our approach for a new general purpose discrete event simulator in this application field, focuses on:

- a modeling basis with a well-defined syntactical structure comprising consistent hierarchical model levels,
- a flexibility to integrate any control concept using Petri nets, state machines or any suitable programming language,
- a graphical representation of the simulation model by several different views,
- a highly interactive modeling in a man-machine dialogue.

An initial version of a prototype of our simulator has been realized. The first experiences with its application are promising.

8 . References:

[1] David Parnas: *On the Criteria To Be Used in Decomposing Systems into Modules*; Communications of the ACM, Vol. 15, No. 12; December 1972.

[2] David Parnas: *On a 'Buzzword'*; *Hierarchical Structure*; IFIP Congress Stockholm; 1974.

[3] Michael Rüger, Ulrich Hoppe, Heiko Kirchner: *Objektorientierte Modellierung von Bausteinen innerhalb der Simulatorenentwicklungsumgebung "Create!"*; Fraunhofer-Institut für Materialfluss und Logistik; TU "Otto von Guericke" Magdeburg; 1990.

[4] J. Dähler: *Ein Werkzeug für den Entwurf verteilter Systeme auf der Basis erweiterter Petri-Netze*; Ph.D. thesis ETH Nr. 8770, Swiss Federal Institut of Technology, Zurich; 1989.

[5] J. Dähler et al.: *A Graphical Tool for the Design and Prototyping of Distributed Systems*; Proceedings of the International Zürich Seminar; 1988.

[6] W. Reisig: *Petrinetze*; Springer-Verlag; 1982.

[7] R. König, L. Quäck: *Petri - Netze in der Steuerungstechnik*; VEB Verlag Technik Berlin; 1988.

[8] J.L. Peterson: *Petri Net Theory and the Modeling of Systems*; Prentice-Hall; 1981.

[9] Haller, Lanz: *Logistikbaustein für den Bereich Fertigungstechnik*; Diploma thesis, Swiss Federal Institut of Technology, Zurich; 1989.

[10] Garzia M.R.: *Discrete Event Simulation Methodologies and Formalisms*; Simulation Digest, Vol. 21:1, (pp. 3-13); 1990.

[11] Becker B.-D., Dangelmaier A.: *Vergleich und Entwicklungsrichtungen von Werkzeugen zur diskreten Simulation von Fertigungssystemen*; Informatik Fachberichte 150, Springer-Verlag.

[12] Graber A, Ulrich H: *Simulation des Güterverkehrs für die SBB*; Forschritte der Simulatiostechnik Vieweg-Verlag 1991.

[13] T. Thomasma, Y. Mao, O.M. Ulgen: *Manufacturing Simulation in Smalltalk;* 1990.

[14] D.P. Bischak, S.D. Roberts: *Object-Oriented Simulation;* Proceedings of the 1991 Winter Simulation Conference, 194-203.