Proceedings of the 1999 Winter Simulation Conference
P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds.

INCREMENTAL SYSTEM DEVELOPMENT OF LARGE DISCRETE-EVENT SIMULATION MODELS

Lars G. Randell
Lars G. Holst
Gunnar S. Bolmsjö

Department of Mechanical Engineering
Lund University
Box 118, SE-221 00 Lund, SWEDEN

ABSTRACT

The paper presents a methodology for incremental development of large discrete-event simulation models. The proposed methodology has evolved during a project performed in collaboration with BT Products.

The methodology is based on configuration management to secure simulation model integrity and modularization of the model. Modularization reduces complexity, allows modeling at a higher level of abstraction and is a prerequisite for both configuration management and incremental development. These are well known methods in the software arena which have been merged and applied to discrete-event simulation. Concurrent development was performed in a heterogeneous environment at geographically separated sites.

Due to the incremental approach, results are implemented at successive stages which increases flexibility. Being performed in increments the modeling effort is less susceptible to changes in the studied system thus reducing risk. The lead-time to implementation is reduced since each successive stage is ended with an implementation phase.

1 INTRODUCTION

Building large discrete-event simulation (DES) models of large manufacturing systems is motivated by the fact that even small improvements yield substantial cost reductions. Simulations with a holistic view capture subsystem interactions and avoid sub-optimizations.

As the size of the simulation models increase so does the development lead-times and thus lead-time to implementation. Lead-time can be reduced by using multiple developers who work concurrently. We need a method that facilitates project management and at the same time allow concurrent development of simulation models.

With incremental development and successive implementation of simulation results, costs can be balanced with cost reductions as the simulation study progresses. Incremental development also reduces the overall lead-time.

Pidd (1996) gives a number of principles for simulation modeling where two of them are emphasized here: (i) Start small and add, and (ii) decomposition. The methodology presented is based on those principles.

Large models increase model complexity. A method for model design that increases the level of abstraction and modularizes the simulation model would reduce the level of complexity. Previous work in this area in the simulation arena has been done by Pidd and Castro (1998) and Pidd (1996).

We will present how modularization and configuration management (CM) can be combined with incremental development to reduce lead-time to implementation. The proposed methodology is based on well established principles for software development (Babich 1986, Bersoff 1980, Rumbaugh et al. 1991).

In a recently performed discrete-event simulation project, configuration management was used for project management and the model was modularized. Parts of the model where treated as black boxes to reduce the modeling effort and reduce lead-time. The black boxes can be modeled in detail in later modeling stages.

2 MATERIALS AND METHODS

The proposed methodology was developed during two case studies of which the most relevant is presented in this paper. Results and conclusions has been achieved by actively participating in the simulation projects, building the models, managing the projects, administrating the CM system, and through literature studies.

In the described case QUEST was used. QUEST is a DES system from Deneb Robotics Inc. with integrated

3D capabilities. Models are built from a Graphical User Interface (GUI) and customized logic is coded with the Simulation Control Language (SCL). To run experiments there is a Batch Control Language (BCL) which also can be used to dynamically alter a model.

To support configuration management Concurrent Versions System (CVS) from the Free Software Foundation was used. CVS, as the name implies, is a version control system allowing concurrent development. CVS can handle binary files, e.g. a CAD file or word processor file. Another benefit of CVS is that of being able to work in heterogeneous environments, at different geographical sites. In this case the environments were IRIX, a Unix System V flavor running on Silicon Graphics workstations, and Windows NT on INTEL based workstations.

The CVS package contains no GUI itself, but there are a number of GUIs available. The GUIs adds functionality like easy browsing of revision trees, log messages etc. On IRIX, tkCVS, was used and on Windows NT, WinCVS, was used.

To utilize repository access via Internet a server was setup on IRIX at Lund University. The server was set up for password authentication thus protecting the repository to some extent.

3 INCREMENTAL DEVELOPMENT

The proposed methodology is based on incremental development of a model with few elements and little detail capturing a holistic view of the studied system. The model is therefore initially not large, but might eventually become so as development progresses. The case described is from manufacturing, but the methodology could be applied to any type of simulation study.

Simulation models within this context are supposed to exist in parallel with the studied existing system over a period of time. Simulation models are often built for fast analyzes of small parts of a manufacturing system. This has its application, but we argue that simulation is more cost efficient when a holistic view of the manufacturing system is applied and where manufacturing system defects are removed continuously over time.

Lead-time for a project can be measured from the start of the simulation study until the presentation of the simulation results. A more correct figure would be received if the lead-time was measured until completed implementation. With this view of lead-time, fast implementation becomes an issue.

3.1 Dividing the Simulation Study into Stages

Two ways of dividing the simulation study into stages can be identified. One is to work vertically first and then horizontally, i.e. develop small simulation models of small parts of the manufacturing system and implement the results. When the sub-models are ready they are merged into one large model. The other way is to take a holistic view of the manufacturing system and generate a draft model. With this approach we work with a top-down approach, i.e. first horizontally to identify where the defects are, and then vertically to identify and remove the defects, see Figure 1. The latter approach has two benefits: (i) The risk of sub-optimization is minimized, and manufacturing subsystem interactions are captured. (ii) Only the parts of the manufacturing system that are considered bottle-necks need to be studied in detail, thus reducing the overall effort to identify and implement improvements.

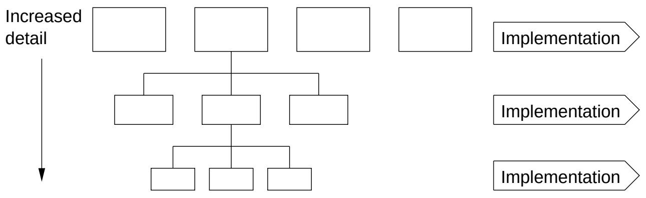

Figure 1: Incremental Development with Fast Implementation

3.2 Costs and Incremental Development

Although large simulation models can help reduce costs they also tend to generate substantial costs during development. This fact makes managers reluctant to start such simulation studies. If the simulation study could generate cost savings fast, and thus finance itself continuously over time, the high costs would no longer be a strong argument. Decomposition of the simulation project with smaller budgets in each stage makes it easier to motivate each stage of the simulation study. Each incremental stage is also followed by an analysis that will give good estimates of possible gains if the simulation study would be continued.

The proposed methodology allows costs to be balanced with cost reductions as the simulation study proceeds. Implementation phases between stages will yield the cost reductions that allow the simulation study to continue. This results in a more balanced flow between costs of simulation model development and implementation costs versus the cost reductions or increased incomes resulting from the implementation of simulation results, see Figure 2.

3.3 Reduction of Risks

Another result of the reduced lead-time is the risk reduction. The time span the project team and the studied system is susceptible to change is reduced. Of course the studied system will change and there will be personnel turnover, but

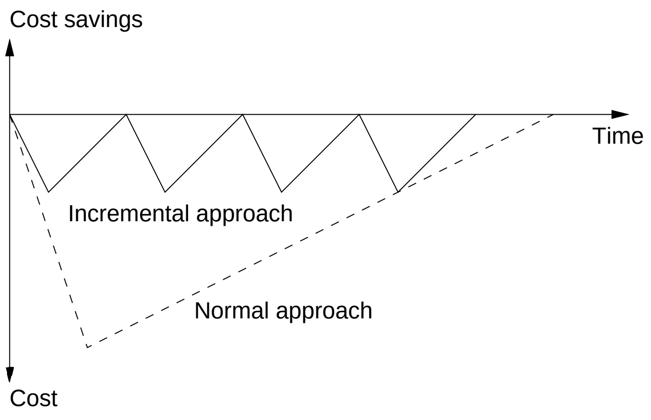

Figure 2: Cash Flow for Simulation Projects with the Normal Approach Versus Incremental Development

the risk of having that in the middle of a stage is minimized. Changes occurring in parts of the manufacturing system not yet modeled in detail will not affect the simulation study in contrast to a system modeled as a whole from the start.

4 CONFIGURATION MANAGEMENT

Configuration management is about managing the life cycle of a system. The overall goal is to attain and maintain product integrity throughout the life cycle. Integrity is defined as being the intrinsic attributes (Bersoff 1980): (i) which characterize a product that meets user requirements, (ii) which facilitate traceability, (iii) which characterize a product that meets specified performance criteria, and (iv) which live up to cost and delivery expectations of the user and buyer.

The four classic operational aspects of configuration management are (Bersoff 1980, Dart 1990):

Identification: The identification is about how configuration items are arranged relatively over time.

Control: Configuration control is how to control the development of the system throughout the life-cycle.

Status Accounting: During development the status of components and change requests are logged.

Audit and review: Audit and review is used to validate the completeness of a product and maintaining consistency among the components.

Although there is overhead involved in using configuration management, it is generally agreed that the consequences of not using configuration management can lead to many problems and inefficiencies (Babich 1986, Bersoff 1980, and Dart 1990).

4.1 Configuration Control Board

Project control is done through a Configuration Control Board (CCB). Representatives from the developer, customer, and user should be members of the CCB. For a simulation project another member can be identified, that of being the supplier of input data. This is an important role since the input data greatly affects the model design and simulation results. How and when data has been captured affects how the data is to be implemented in the simulation model and thus a tight contact between the developers and the input data suppliers is desired. A CCB with all representatives is usually difficult to achieve, but recommended.

Requirements are proposed by the CCB and handed over to those initiated in the problem area. Initiated personnel locates possible problems and return a change request with a motivation. This interaction is iterated until all parties are satisfied.

4.2 Baselines and Development Phases

Law and Kelton (1991) and Banks, Carson, and Nelson (1996) describe a number of phases in a simulation study. In Bersoff's (1980) and Babich's (1986) terms each phase is concluded with a *baseline*. Each phase generates design objects for the next phase e.g. function X implies Y, i.e. each baseline is a base for the next phase in the development cycle. Proposed baselines in a simulation context are presented in the following paragraphs. In general, projects might need more or less baselines.

The functional baseline (FB) is a product of the model concept formulation phase and is typically a requirements document. There should be consensus on the requirements between the producer, buyer, and user. To attain consensus the producer, buyer, and user should be part of all phases in the life cycle, which is done through the CCB. In the FB audit it is evaluated whether stated requirements are complete, consistent, and unambiguous. This base line is the top functional basis of the entire project so it defines what is to be established.

The design baseline (DB) is the product of the detailed design phase. The model design documents contain three sections; (i) A decomposition with modules and interfaces; (ii) Descriptions of each module with data requirements; and (iii) A description of each interface. Definition of interfaces will allow interchangeability of versions of modules and thereby facilitate configuration management (Babich 1986). Note that no model building is performed until after this baseline.

During the design phase not only the design of the model should be performed but also the experimental design with respect to the functional requirements. Law and Kelton (1991) and Banks, Carson, and Nelson (1996) place the experimental design after the PB. However, experimental design puts requirements on the model design and it is therefore suggested to place it in the design phase.

Data requirements for the study is revealed during this phase. It should be made clear whether the required data is available in a suitable format and resources needed to collect the data.

The product baseline (PB) is the working simulation model. The model is verified and validated as well as modified according to additional requirements.

The experimental baseline (EB) is the end of the production runs, analyzes and documentation of results and findings.

The implementation baseline (IB) concludes the project, or subproject if incremental development is employed.

When the implementation is concluded an evaluation of results compared to simulation results should be made. With this information it is possible to enhance prediction validity in successive stages or future projects. The implementation baseline then forms the base for the functional baseline in the next increment of the system development.

Although presented here as a linear development one realizes that changes might need to be made to the baselines. This results in change requests and then in updates of the baseline. However, the whole point of configuration management is to reduce the number of back-steps by investigating everything thoroughly in advance. By formally issuing change requests their refusal or acceptance and implementation is under control.

5 MODULAR DESIGN

Large simulation models are inherently complex. With multiple developers to reduce lead-time project management becomes even harder, as seen in Figure 3.

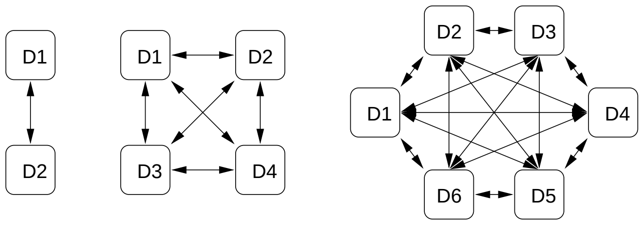

Figure 3: The Increasing Number of Communication Channels with Multiple Developers (Babich 1986)

Complexity can be handled using modules and information hiding when designing the model. This interacts well with the incremental approach of developing a large simulation model as well as with configuration management.

Modular design is a prerequisite for incremental development. If there are direct connections between the different internal details of modules it would be difficult to remodel one of the modules without risking the entire model. This would have severe impact on the incremental strategy.

5.1 Modularity

Modularity is based on *locality* and *encapsulation* (Pidd and Castro 1998). Locality is the notion that all information relevant to a design decision should be kept in one place, i.e. within a module. Encapsulation, or *information hiding*, separates internal, hidden aspects of an module from the external (Rumbaugh et al. 1991). A modular model should satisfy two conditions according to Pidd and Castro (1998): (i) The model or component must not directly access the state of any other model or component. (ii) The model must have recognized input and output ports through which all interaction with the exterior is mediated.

It isn't necessary to build the detailed models within the top-level model but it is preferred since system interactions are captured. The model size might cause the simulation runs to be too long for practical use. In these cases the sub-models have to be run separately. For the simulator used in this case there is a client-server software, SysLink from Tehdasmallit Oy, that can connect simulation models over a network in a heterogeneous environment. With this methodology sub-models can be considered black boxes from the top model view. In later stages of the incremental development the sub-models can be modeled in more detail without changing anything in the overall structure. With the client-server methodology the models can be run on one or several computers depending on requirements.

6 CASE STUDY

In a case performed with BT Products, a world leading manufacturer of electrical warehouse trucks in Sweden, there were four developers of the model at two geographically separated sites. Developers used different operating systems; Windows NT and IRIX. Configuration management was used to manage the project and each developer was assigned modules to develop. Concurrent development was performed through a central repository via Internet. The geographical separation posed only a minor problem with the firewall at BT Products and a modem was used to get around the firewall. The goal of the study was to reduce the lead-time for truck mast manufacturing. Figure 4 shows the central part of the model.

6.1 Modules

First a base model was built with the basic elements and the layout. This stage was used to place the model elements correctly on the layout. The model was then split into four modules and checked in to the repository. When all the modules were considered complete the modules was merged again to form the final model.

QUEST has little support for modular development. There is no generic way of building modules and then link

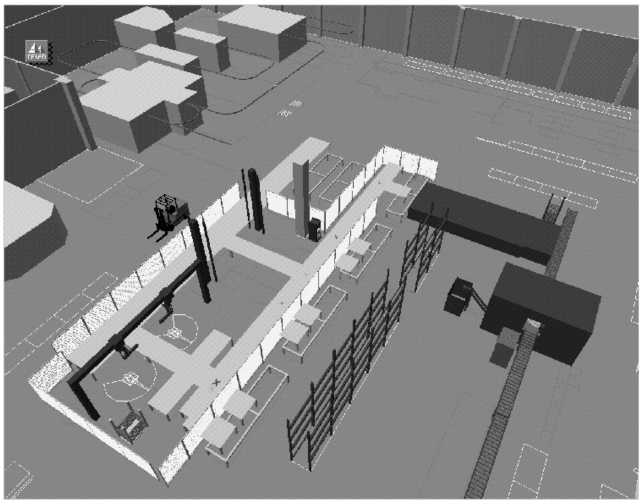

Figure 4: The Central Part of the Simulation Model at BT Products

them. To do this additional software has to be used. In future projects SysLink will be used to connect modules which render possible the notion of black boxes and distributed simulation runs.

The interfaces between the modules were defined by (i) the type of connection between elements, (ii) part definitions, (iii) common global variables, and (iv) attributes.

During development of the modules *stubs* were used simulating the not yet developed modules connected to the module under development. The stubs did not have logics but generated the same output or accepted the same input with the same interface as the real module would have done. In this case the stubs were sources and sinks simulating the preceding and succeeding modules in the production flow.

6.2 Incremental Development

The paint shop was treated as a black box leaving details for later stages. There were two reasons for leaving the paint shop with less detail: (i) the amount of data needed was reduced considerably, and (ii) it was considered unnecessary for the goals set for the first part of the simulation study. The lead-time for the first stage in the simulation study could therefore be reduced both in the modeling effort and in the data collection effort.

The production runs showed that the paint shop was the bottle neck. During the input data collection it was discovered that throughput could be improved by reorganizing the work in the paint shop. The study gave at hand that more details in the paint shop module would not be needed since improvements could be made with the information from the input data collection. This study thus reduced the effort in this stage and also showed that it was unnecessary to continue the study of this module, reducing the over all simulation effort.

The robot welding cell was modeled in detail which took considerable time. It would have been better to treat this cell as a black box, at least initially, to reduce modeling lead-time.

6.3 Configuration Management

There are two basic approaches to develop simulation models: *simulation languages* or *simulators* (Ball 1996, Ball 1994, Boughton 1995, Shewchuk and Chang 1991). Hybrid systems is a third approach which combines the flexibility of a simulation language with the user-friendliness of a data driven system.

Most version systems saves different revisions of a file as the difference between the previous revision and the current. With concurrent development conflicts occur when two developers are editing the same lines in the code. This rarely happens when programming is done manually as with a simulation language or a general purpose language. However, when using a simulator or a hybrid system the programming is done by the simulator which might cause conflicts when checking in a new revision. This was the case with QUEST and might be the case for other hybrid systems or simulators as well.

In the described case only one developer at a time could work on a model because of the check in conflicts. This was another reason to divide the model into modules. With several modules concurrent development was possible with one developer assigned to one module at a time. To avoid unintentional conflicts the currently edited model file was locked. Locking a file allowed other developers to view and edit a model file, but they could not perform a check in.

The notion of well defined base lines was not fully implemented in this project, but is highly recommended. Interfaces were communicated between developers verbally and changed slightly during development.

Tagging was done each time a model passed a vital development stage. Although there never was a need for retrieval of old configurations, it felt secure to be able to step back if something would have failed.

The tight contact between the CCB and the developers prescribed in section 4 was not entirely implemented in this project. Managers was informed about the project status but had no active part in the development process. The project group itself had all the roles of the CCB. One of the members in the project team was the customer, user, and data supplier. Formal written change requests were not used but communicated verbally.

6.4 Directory Structure

It is vital to plan the directory structure beforehand. We didn't and some extra book keeping was necessary. The documentation and model files should be placed in such way that the tags, i.e. the base line marks, mark both documentation and model files. Each module should be in a subdirectory with subdirectories for data, logics and documentation respectively. Tagging thus only affects the files in that module. Module directories are then organized according to the hierarchical structure defined during the design phase. A proposed directory structure that work well with CVS would then look like:

```
model/
   doc
   include
   src
   data
   module_1/
      doc
      include
      src
      data
      submodule_1_1
         ...
   module_2
      ...
```

Depending on what simulator used the directories might be organized slightly different.

We used one file for each type of logic (route, request, process, etc.) and module with a structured naming convention which made it easy to keep track of the sources. However, it would have been better if each element had it's own source file. We also used a separate directory with files for common definitions and declarations such as constants, attributes and procedures. To avoid the risk of name clashes when merging the modules each element, file and routine name in a module was prefixed with a two letter combination identifying the module.

6.5 Multiple Developers

As mentioned in section 5 multiple developers makes project management complex. In this case the developers was not experienced and therefore could not work entirely on their own which caused an excessive amount of communication. This was expected to some extent since the case was for learning discrete-event simulation at BT Products, but the amount of communication needed was underestimated.

6.6 Documentation

The documentation was hard to keep updated as development progressed and new information became available. Documentation was transfered via email in several instances, causing the expected problems with different versions on different computers. This proved that it is not possible to perform a project without some kind of central repository.

A major problem writing the documentation was that it was spread in several files, which Babich (1986) refers to as *the double maintenance problem*. Some documentation was placed in the code, explaining details. The general documentation was placed in the report document that was to be the final documentation of the model. If possible, it would have been desired to have documentation fields for each model element. In QUEST there are no such fields, thus notes has to be done in some other document outside the simulator environment which complicates documentation of the model.

This is a complex problem. We want the documentation in one place to avoid the double maintenance problem. At the same time we want it to be easily available for the part of code, or element under study at the moment. We had no solution and thus had the double maintenance problem.

6.7 Data

In the design phase data requirements were determined. As in many other simulation projects the data available was not always suitable. Either the data did not exist, was in the wrong format, not easily accessible, or corrupt in some way. For each change in data, the design has to be modified slightly. The data requirements should therefore be determined early and checked before continuing the design effort. The tight connection needed between developers and the suppliers of data cannot be emphasized enough.

7 DISCUSSION AND CONCLUSIONS

One of the major benefits of the incremental development approach is the risk reduction. Since a simulation study is divided into stages, with implementation in between, results are secured and the simulation study can be aborted without loosing too much money. If the project has to be aborted for some reason at least the stages performed have resulted in actual implementation. If an all-in-one-big-chunk project has to be aborted there are no results implemented, only costs. In these days of continuous migration of companies and fast shifts in strategies there is reason to believe that this risk reduction is appreciated.

The reduced lead-time of the successive stages reduces the probability of changes in the studied system and project organization during a stage.

The incremental development methodology allows learning as the project goes on. The implementation phase and the simulation analyzes will give ideas and estimations for the next development stage. Experiences from the stages performed adds considerable knowledge for use in the succeeding stages. The result is a more controlled development of the simulation model. An example of this was the paint shop where the modeling effort could be reduced in the performed stage and there was no need to develop this module further in a later stage.

The traceability inherent to configuration management reduce the time spent on maintenance efforts considerably. Traceability and several developers also reduces the impact of personnel turnover.

One of the drawbacks of incremental development is that the total amount of data needed can be larger than if the entire model was performed in one single stage. Each level of detail needs data at different levels. However, it isn't necessarily so that all the modules have to be built, as mentioned above. The simulation study can stop at a certain level of detail if there is no purpose of adding more detail, which reduces or removes the effect of the drawback.

The incremental approach has more flexibility than the all-in-one-big-chunk approach. The goals of the simulation study can change at each new stage. Thus the focus of the simulation studies can shift with the shift of focus of strategies and goals of the company.

Continuous improvements of a manufacturing system utilizing incremental simulation model development yields long term results and doesn't overload the organization at discrete points of time.

Some simulators seem well suited for modular development while others have a structure that makes modular design difficult. A simplified generalization of this statement would then be that simulators are less suitable and simulation languages are more suitable for modularized development. The ideal would then be an object oriented simulation tool.

A prerequisite for the promised lead-time reduction when development is performed concurrently is that the developers can work on their own and communication should only concern the interfaces. A CM tool facilitates concurrent development and is recommended.

ACKNOWLEDGMENTS

The work of Lars Randell and Lars Holst is funded by the Swedish National Board for Industrial and Technical Development (NUTEK), the CONSENSUS for VSOP project, and PROPER, a program funded by the Swedish Foundation for Strategic Research

REFERENCES

Babich, W. A. 1986. *Software Configuration Management*. Addison-Wesley, Reading, Massachusetts.

Ball, P. D. 1994. *Design of an expandable manufacturing simulator through the application of object-oriented principles*. PhD dissertation, The University of Aston in Birmingham.

Ball, P. D. 1996. Introduction to discrete event simulation. In *Proceedings of the 2nd DYCOMANS workshop on Management and Control : Tools in Action*, 367–376. Algarve, Portugal.

Banks, J., J. S. Carson, and B. L. Nelson. 1996. *Discrete-event system simulation*. 2nd ed. Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458.

Bersoff, E. H., V. D. Henderson, and S. G. Siegel. 1980. *Software Configuration Management: An Investment in Product Integrity*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Boughton, N. J. 1995. *Modelling manufacturing planning and control systems: The application of object-oriented principles and discrete-event simulation*. PhD Dissertation, The University of Aston in Birmingham.

Dart, S. 1990. Spectrum of functionality in configuration management systems. Technical Report CMU/SEI-90-TR-11 ESD-90-TR-212, Carnegie Mellon, Software Engineering Institute. Available online from `http://www.sei.cmu.edu/legacy/scm/abstracts/abscm_spectrum_of_func_TR11_90.html` [accessed June 22, 1999].

Law, A. M., and D. W. Kelton. 1991. *Simulation Modeling and Analysis*. 2d ed. McGraw-Hill Inc.

Pidd, M. 1996. Five simple principles of modelling. In *Proceedings of the 1996 Winter Simulation Conference*, 721–728.

Pidd, M., and R. B. Castro. 1998. Hierarchical modular modelling in discrete simulation. In *Proceedings of the 1998 Winter Simulation Conference*, 383–389.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. 1991. *Object-Oriented Modeling and Design*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632.

Shewchuk, J. P. and T. C. Chang. 1991. An approach to object-oriented discrete-event simulation of manufacturing systems. In *Proceedings of the 1991 Winter Simulation Conference*, 302–311.

AUTHOR BIOGRAPHIES

LARS RANDELL is a Ph.D. student in the Department of Mechanical Engineering at Lund University. He received a M.Sc. in Mechanical Engineering from Lund University 1996.

LARS HOLST is a Ph.D. student in the Department of Mechanical Engineering at Lund University. He received a M.Sc. in Mechanical Engineering from Lund University 1998.

GUNNAR BOLMSJÖ received the M.Sc. and Ph.D. degree in Mechanical Engineering from Lund University in 1981 and 1986 respectively. From 1986 he was a research associate at the Dept. of Mechanical Engineering, Lund University, and is since 1987 Professor in Robotics at the same department. He is a member of IEEE, SIRES, AAATE and JOM.