Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

TOWARD A STANDARD PROCESS: THE USE OF UML FOR DESIGNING SIMULATION MODELS

Hendrik Richter
Lothar März

Fraunhofer Institut für Produktionstechnik und Automatisierung
Nobelstrasse 12, D–70569 Stuttgart, GERMANY

ABSTRACT

Designing complex simulation models is a task essentially associated with software engineering. In this paper, the Unified Modeling Language (UML) is used to specify simulation models. It is shown that, similar to the "Unified Process" in software engineering, such a methodology forms a sound base for developing complex simulation models. An example is provided to illustrate how this approach supports the design process.

1 INTRODUCTION

Discrete–event simulation is frequently used to evaluate and optimize production processes. Sometimes, such simulations can be performed by standard programs using a tool box paradigm. If, on the other hand, a more complex scenario is to be described such tool boxes are not powerful enough and one has to rely on simulation programs with open architecture. So, most complex simulation projects need to be considered essentially as a task of software engineering.

Recently, a new approach to software engineering has been developed and established, the so–called "Unified Process" (Royce 1998, Booch, Rumbaugh and Jacobson, 1999). Its main features are an iterative and incremental development process instead of a waterfall life cycle, promotion of a component–based architecture, and the use of a graphical modelling language, the Unified Modelling Language (UML). In the following, we attempt to adopt this approach to specify simulation models. By using the UML, all essential features of structure and dynamics of the simulation model to be built can be described. Thus, before implementation starts, requirements on the simulation model can be easily formulated. In addition, users as well as developers obtain a specification and a documentation parallel to the process of building the simulation model.

The remainder of the paper is organized as follows. Section 2 gives a brief overview about object–oriented modelling and documentation using UML and its use to specify simulation models. In Section 3, an evolutionary algorithm is considered. This example shows how the proposed methodology can be applied to specify all essential features of implementing such an algorithm in an object–oriented simulation system. Finally, in Section 4 conclusions are drawn.

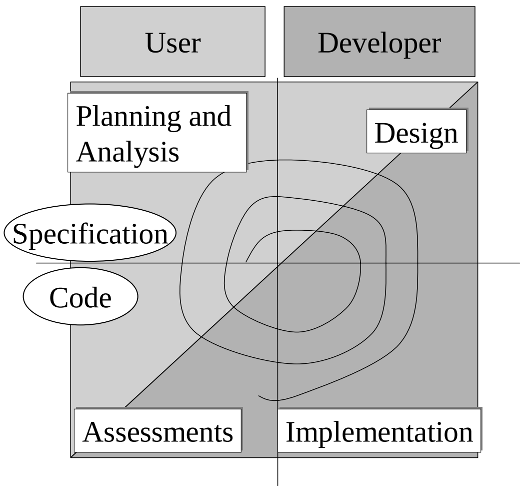

Figure 1: Software Development Process

2 OBJECT–ORIENTED MODELLING AND DOCUMENTATION WITH UML

As mentioned above, designing and building a simulation model is often associated with software engineering.

Software development proceeds from planning and analysis through design, implementation, and assessment. It has been shown that it is useful to proceed in an iterative and incremental process as shown in Figure 1 (Gemma et al. 1995, Royce 1998). Such an approach to software engineering is particularly supported by the Unified Modeling Language (UML), (Fowler and Scott, 1998, Booch, Rumbaugh and Jacobson, 1999). Meanwhile, the UML is an industrial standard for object-oriented analysis and software development. It is a mainly graphical model of the object under study for which a software solution is to be developed.

Through UML we obtain, for the whole development process, a description of requirements and performance features of the simulation model regarding its structure and dynamics. To that end, different types of UML diagrams can be employed. Thus, interactions to user (through use cases), structure (through class diagrams) as well as dynamics (through interaction diagrams and state diagrams) of a simulation model can be described in a comprehensible way.

In addition, by UML diagrams we obtain detailed requirements on a simulation model. So, the user as well as the developer have a common language for communication. The UML can serve not only as a bridge between user and developer, but also between specification and code (see Figure 1). This is particularly helpful in large simulation projects, where the success depends highly on finding common grounds for all participants.

Describing simulation models, as with software in general, requires to allow different views. Crucial for understanding is interaction with users, static structure, and dynamic behaviour. It has been shown that such an UML-based documentation is more sound than comment on the code alone. By using the modeling language UML, the documentation is built parallel to the development process. Moreover, such a documentation of the principles underlying the simulation model allows the building of reusable components.

It can be seen that the proposed methodology has a number of advantages. These are in particular:

1. Establishing a general standard for modelling and documentation of simulation models.
2. Independence from the used simulation software, i.e., the way the model is encoded.
3. Definition of a methodology for developing simulation models through the use of object-oriented modelling and documentation.
4. Visualization of concepts, structures and dynamics of a simulation model through UML.
5. Identification of reusable components in simulation models.
6. Building a framework for project management for simulation studies.

In the next section, the given approach to building simulation models is illustrated by an example.

3 ILLUSTRATVE EXAMPLE: AN EVOLUTIONARY ALGORITHM

In this section, we illustrate the proposed methodology by an example: an evolutionary algorithm used to optimize parameters of a production process specified by an external simulation model. The algorithm has the following structure (Goldberg, 1989, Michalewicz, 1999):

Algorithm:

(i) Calculate the objective function using parameters obtained from the external simulation model.
(ii) Select individuals for which the value of the objective function is above average.
(iii) Create a new individual from individuals chosen in (ii).
(iv) Perform random alterations in the individuals created in (iii).
(v) Loop to (i) unless a maximal value of the objective function or another termination criterion is reached.

The given algorithmic structure will now be realized in a simulation system. Thereby, the following requirements are to be considered:

– Standard coupling to the external simulation model, that is, simulation model and optimizer communicate by input and output parameters and are otherwise completely separated (encapsulated).
– Specific optimizer components, e.g. fitness function, recombination algorithm are changeable in a simple way.
– General documentation of the model.

Documentation of the algorithm is done in the modeling language UML using the software package Rational Rose. In general, there are two types of classes: manager classes and function classes. The job of manager classes is to coordinate and call function classes. They schedule the steps of the algorithm and determine which function class is called. Function classes perform different tasks, e.g., calculating the fitness or doing recombination.

In Figure 2, manager classes are depicted in an interaction diagram. It shows how the manager classes are called in their temporal order. The EA manager calls the manager responsible for a particular step according to the step given in the algorithm above. This manager now calls the function classes required to perform the task. The called function class informs the manager about having completed its job.

As soon as all function classes have performed their task, the manager calls back the EA manager in order to give him the information to go on by calling the manager class for the next step.

This hierarchical structure can be pursued. After doing the steps selection, recombination and mutation, and passing the termination criterion, the EA manager calls the simulation manager to calculate the parameters used for the fitness function. This simulation manager carries out the

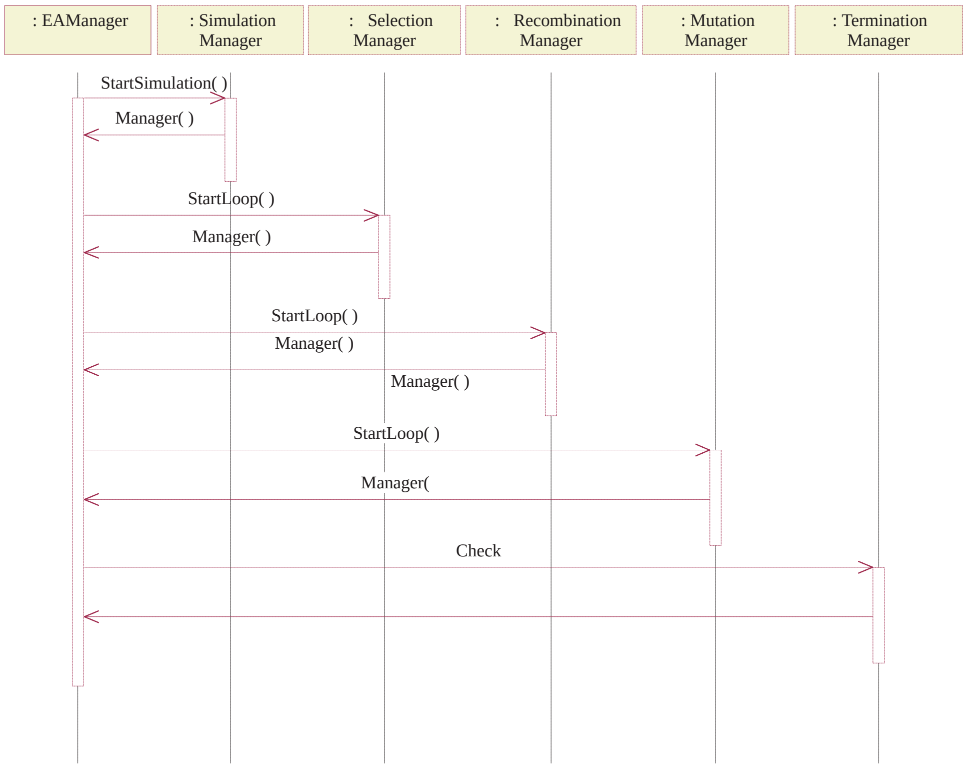

Figure 2: Interaction Diagram for the Implementation of an Evolutionary Algorithm

simulation of the production system under study and the next iteration starts.

Another way to describe dynamics of a simulation model is a state diagram as shown in Figure 3. A state diagram is used to show dynamics within one class, e.g. in which order operations of a class are called.

In contrast to the dynamic behaviour shown in the interaction and state diagram, a class diagram shows the static structure and the relationships between classes, see Figures 4 and 5. Such a diagram visualizes, for example, reading and writing access on attribute and operations. In addition, the general structure of the architecture of the simulation model to be designed is depicted. Figure 4 shows the coarse structure of the manager classes introduced in the interaction diagram with their operations.

For each manager class we have a more detailed class diagram, see Figure 5. Here, the subclasses of the selection manager are shown. It works the following way: the selection manager selects two individuals from SelectionPopulation (using GetPairOfIndividuals), calculates the fitness for each individual and chooses the one with higher fitness for the whole population. The underlying dynamics can be assigned by an interaction diagram not shown here. In this way, we obtain a description for static structure and dynamic behaviour as precise as required. In the course of building a simulation model, basic architecture should be designed in structure and dynamics at first. Then, the model can be built in detail, following an iterative process. Finally, it should be mentioned that for each class, attribute or operation, a separate documentation can be inserted which can be as detailed as comments on the code.

4 CONCLUSION

In this paper, we have discussed the issue of using the UML to design and build simulation models. UML diagrams can be used to assign different aspects of interest, e.g., the interaction with users, the static structure and the dynamic behaviour. In this way, we obtain a graphical description of the simulation model which can be used as a specification and a documentation.

REFERENCES

Booch, G., Rumbaugh, J., Jacobson, I. 1999. *The unified modeling language user guide.* Addision Wesley, Reading, MA.

Fowler, M., Scott, K. 1998. *UML distilled. Addision Wesley*, Reading, MA.

Gemma, E., Helm, R., Johnson, R., Vlissides, J. 1995. *Design patterns: Elements of reuseable object–oriented software*. Addision Wesley, Reading, MA.

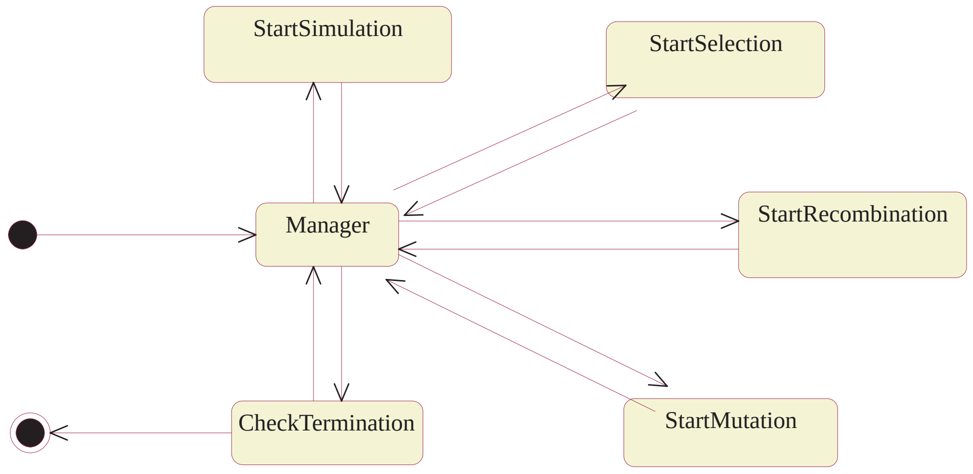

Figure 3: State Diagram of the Class EAManager

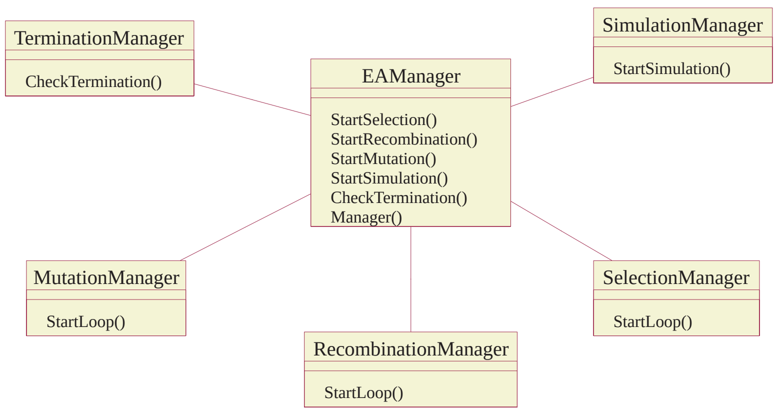

Figure 4: Class Diagram for the Implementation of an Evolutionary Algorithm

Goldberg, D.E. 1989. *Genetic Algorithms in search, optimization, and machine learning*. Addision-Wesley, Reading, MA.

Michalewicz, Z. 1999. *Genetic algorithm + data structure = evolution programs*. 3rd ed. Springer-Verlag, Berlin.

Royce, W. 1998. *Software project management: A unified framework*. Addision Wesley, Reading, MA.

AUTHOR BIOGRAPHIES

HENDRIK RICHTER is researcher at the Fraunhofer Institut für Produktionstechnik und Automatisierung (Manufacturing Engineering and Automation) in Stuttgart, Germany. He holds a masters degree from Technische Hochschule Leipzig, Germany and a Ph.D. degree from Technische Universität Dresden, Germany. He is currently engaged in research in the area control and optimization of complex system and simulation and its software engineering. His email address is <hir@ipa.fhg.de>.

LOTHAR MÄRZ is project manager at the Fraunhofer Institut für Produktionstechnik und Automatisierung (Manufacturing Engineering and Automation) in Stuttgart, Germany. He holds a masters degree from Technische Hochschule Darmstadt, Germany and Ecole Centrale de Lyon, France. His primary research interests are discrete event simulation, modelling and optimization. His email address is <lom@ipa.fhg.de>.

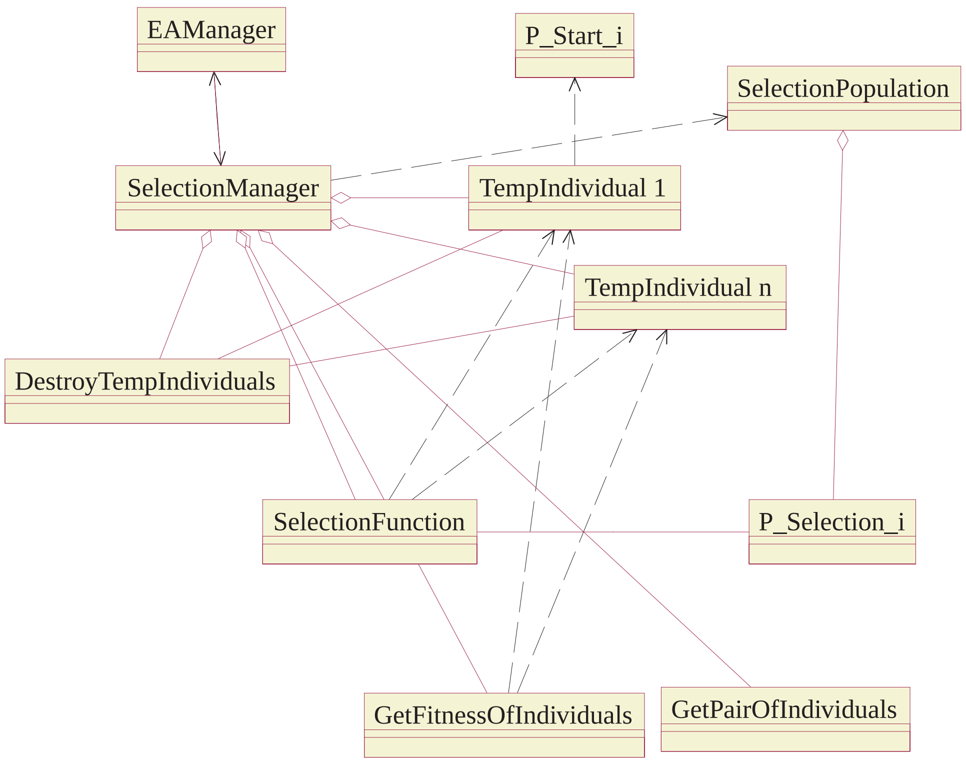

Figure 5: Class Diagram of the Class SelectionManager