A Distributed, Object-Oriented Simulation System based on Hints

L. Böszörményi and A. Stopper

Institut für Informatik, Universität Klagenfurt, Universitätsstraße 65-67, A-9020 Klagenfurt, Austria, email: {laszlo, andreas} @ ifi.uni-klu.ac.at

ABSTRACT

A hint-based, distributed, discrete, object-oriented simulation system is described. In course of the design phase of the simulation model, explicit hints can be provided concerning dependencies and information flow inside the model.

The process of parallelization consists of two major steps. In the first step, the abstract model, enriched with user-supplied hints is mapped onto an arbitrary number of active units. In the second step, the active units are mapped onto a given number of physical nodes, characterized by their processing capacity and by the communication latency between them.

The distribution scheme may be dynamic, i.e. simulation objects can change their location in course of the simulation, in order to get better performance.

1. INTRODUCTION

Object-oriented simulation has its renaissance [1,2,3]. The advantages of classification, inheritance and polymorphism become obvious in modeling complex systems.

The need for large-scale, resource intensive simulations grows. A lot of effort have been done in investigating the parallelization of continuous and discrete event simulation models [4,5,6].

Much less effort has been invested in distributed object-oriented simulation. This is surprising somehow, because the idea of object-orientation is fundamentally connected to the idea of distribution. "The core of object-orientation is distribution of control" - states N. Wirth in [7].

Object-orientation has the following advantages in creating distributed, discrete simulations:

1. Through object-oriented modeling the state space gets a structure in the modeling phase already. In contrast to discrete event simulation, where we get one single event list (actually a long "spaghetti") we get a structure that displays parallelism which is inherent in the problem. In the process of mapping the simulation model on a parallel system we can take direct advantage of this information.
2. Objects communicate per definition by messages. Therefore they can be quite naturally distributed. The implementation must, nevertheless, take care of the actual distribution to ensure better performance. Passing a message to an object located in the same address space can be easily mapped onto a method call. Passing a message to an object located in a remote address space can be mapped e.g. onto a remote method call. Another possibility is to move the object to the caller's address space.

The architecture suggested in the paper makes the following assumptions.

1. The problem to be simulated is modeled by some object-oriented design method (we actually rely on OMT [8]).
2. The simulation model can be mapped onto and executed under the control of the simulation system.
3. The simulation model contains implicit information about inherent parallelism.
4. The simulation model is generally able to provide explicit hints to the underlying simulation system to approximate optimal distribution.
5. The simulation system provides a mechanism for dynamic scheduling of objects in a distributed environment.

The realization of the dynamic scheduling system relies on the following concepts.

1. The simulation is performed by a number of active units (active processing elements)
2. Every active unit contains a simulation engine (actually a local scheduler) and an object server
3. Based on run-time information about actual load, and on the hints given by the model, the simulation engines are able to move objects from one active unit to another one.
4. Objects may be passive or active. Passive objects can receive messages, execute a corresponding method and change their state. Active objects are passive objects with the additional capability of executing their own thread of control, in concurrency with the execution of their methods.

2. THE SIMULATION MODEL

According to system theory, a system consists of two parts

a) The elements of the system and their associations (static components)
b) The dynamic components, which affect the elements of the system and cause state-changes

2.1. The modeling steps

In the first step we build the model of the system to be simulated. This contains (according to OMT [8]) a static, a dynamic and a functional model. These models display among others the interdependencies between the different components. They do not contain, however, any information about the "strength" of the interdependencies. From a logical point of view it is irrelevant, whether an association is used only once a year or
once a microsecond. From the point of view of an efficient mapping onto a parallel architecture, exactly this information is of primary importance.

Therefore, in a second step we enrich the model with such kind of information.

We provide for every object class a supplement, which describes:

1. The degree of the dependency between the instances. Objects of a class may be independent, strongly or weakly dependent. In the case of weak dependency additional information is required to describe the scheme of the dependency exactly. (Such additional information may for example state that there is a dependency between objects numbered by even numbers and a dependency between objects numbered by odd numbers and the like.)

2. The kind of the class (static or dynamic). The number of instances in a static class is known in advance. Objects of a dynamic class are generated at execution time, their number lies in a given range or is infinite.

3. The extent of the class. For static classes this is the exact number of instances, for dynamic classes this is a (possible infinite) range.

The class supplements are represented by octagons, their general form is given in figure 1.

We provide for every association a supplement, which describes:

1. The direction of information flow.

2. The frequency of communication. It can be frequent or seldom.

The association supplements are represented by arrows (see figure 2). The black arrows describe associations between static components. The white arrows describe the flow of dynamic objects. The route of an object flow always follows a path consisting of static objects. The frequency of communication is given to each arrow.
As an example, on figure 3 a model of the entrance-management of a football bowl is given in OMT notation. The system consists of three static classes gates, ticket windows and control points. A number of ticket windows are statically assigned to a given gate. The gates may inform their corresponding ticket-windows if they stop visitor entrance. The fourth, dynamic class of the system is that of the visitors. A visitor first passes the gate (some security checks can be done), than buys a ticket (except he has a season ticket) and passes the control point.

The static OMT model is given in dotted lines. The hints, controlling the parallelization process are given in normal lines.

3. THE SIMULATION-SYSTEM

The architecture of the simulation system is shown on figure 4. Every node contains a number of active units. An active unit contains exactly one simulation engine that schedules a number of passive and active objects. The simulation engines have their own logical clock. The simulation system is able to move [9] objects to that simulation engine that process them fastest.
The implementation of the simulation system is based on Modula-3 [10,11] and Network Objects [12].

3.1. Steps of Parallelization

On the basis of these information we can distribute the object instances in two steps.

In the first step we map the set of simulation objects onto an arbitrary number of active units. The following rules apply:

1. Independent instances are placed in different active units.

2. Strongly dependent instances are placed in the same active unit.

3. Weakly dependent instances are clustered in different active units. The clustering is based on the additional information describing the scheme of weak dependency.

In the second step we map the active units onto a number of physical nodes. The capacity (speed and memory) of the nodes and the communication latency between them are considered in this step. The following rules apply:

1. Classes with seldom communication should be located on different physical nodes (if possible).

2. In the case of frequent communication an optimum must be computed as a function of the extension of the classes, the processing capacity of the nodes and the communication latency.

If we apply the parallelization rules on our example then we get the following results.

1. All static objects are placed into different active units (there is no strong dependency in the system). Thus, we get 28 active units (4 gates, 20 ticket windows and 4 control points).

2. The gates are located on the same physical node (no further information is given for clustering weak dependency).

3. The 4 groups of 5 ticket-windows, assigned to a gate can be located on 4 different nodes.

4. The control points can be located on 4 different nodes (these may be, however, the same ones as that of the ticket-windows, see figure 5)
4. CONCLUSION

A hint-based, distributed, object-oriented simulation system has been suggested. We get a highly dynamic and flexible simulation system, that is able to operate near to the optimum, taking advantage of information coming from "above" (from the model) and from "below" (from the run-time environment).

Due to the capability of moving objects, the scheme of execution scales well. In the case of low load all objects tend to move to the same active unit, in the case of high load they distribute themselves automatically.

REFERENCES