Acceleration of Distributed, Object-Oriented Simulations Using a Graph-Optimizing Approach

Andreas STOPPER, Laszlo BÖSZÖRMENYI,
Institut für Informatik, Universität Klagenfurt,
Universitätsstraße 65-67, A-9020 Klagenfurt, Austria,
email: {andreas, laszlo}@ifi.uni-klu.ac.at
http://www.ifi.uni-klu.ac.at/cgi-bin/staff_home?andreas/
http://www.ifi.uni-klu.ac.at/cgi-bin/staff_home?laszlo/

Keywords

Distributed, Object-Oriented Simulation

Abstract

An approach to accelerate distributed, object-oriented simulations is presented in this paper. It is based on the assumption that a higher acceleration can be achieved in an easier way, if the problem is alread tackled early at the modeling stage [STOP 95]. The user adds hints about the communication behavior and frequencies of object classes to the simulation model. Based on this information, an object graph is generated and distributed to a selected number of partitions. The distribution phase is fully automatic. As a result a distribution of the problem *nearby* the communication optimum is generated. In the next phase the distributed simulation program (code) is generated. In a final step the user only has to code the methods of the object classes and run the simulation. The major advantage of this approach is that the user is freed from the difficult task of finding a *good* distribution for the problem to be simulated, which is an important factor for the overall performance of the simulation. Another advantage is the possibility to vary model information (hints) about the communication, and get a new (quasi optimal) version of the simulation automatically generated.

1. Introduction

In distributed simulation the event-oriented view is predominant. There are the two main approaches - the conservative strategy [CHAN 81, MISR 86] and the optimistic strategy [JEFF 82, JEFF 83, JEFF 85]. A lot of effort has been spent in research to generate variants of these basic approaches to be faster under certain circumstances or to find a problem class for which the variant is most suitable.
Object-oriented concepts are well established in sequential simulation from the modeling to the execution stage. In distributed simulation these concepts are mainly used at the programming stage of the simulation cycle. There exist only a few really object-oriented distributed simulation languages [BARG 94, BAEZ 90]. The approaches which are chosen most frequently are the ones which extend object-oriented general purpose languages with language constructs suited for simulation or offer libraries of building blocks which enable the user to build simulation programs by composing them.

At the programming stage the user is still confronted with the problem of how to distribute the objects to be processed in a way that minimizes the interprocess communication, the most critical factor for performance. The larger the problem and the more meshed the communication between objects gets, the less are the chances to find good partitionings.

The key idea of our approach is to tackle the problem early at the modeling stage. The basic model consists of object classes and relations between them (Figure 1). It is extended with information about communication behavior between objects (instances of classes). In our models we distinguish between static objects (these are the components of the simulated system) and (streaming) dynamic objects (these are objects from outside which interact with static objects). The user adds the following information to each class:

1. the number of instances of a class and the basic workload of an instance

2. the communication behavior and pattern among the objects of one class (only for static objects)

3. the starting point streams of dynamic objects in terms of instances of static classes of objects through the system (only for dynamic objects)

Further the user has to add the following information about associations between different classes:

1. the communication of objects of different static classes, the cost of this communication and its frequency (these are static relations, Section 2)

2. the relation of dynamic objects to static objects is expressed by two numbers telling how many static objects of a class are contacted during the simulation, but not exactly which ones, the second number denotes the increase in workload for a static object, which is caused by a contact (dynamic relations, Section 2)

After the modelling stage the user can determine the number of partitions (processes) which should be generated or the maximum workload per partition. In the latter case the minimum number of partitions is generated, so that the maximum of the workload of all

partitions is less or equal to the maximum workload specified by the user.

The user model defines the basis from which the object graph is generated, in which the vertices are the static objects and the edges express the communication between objects. Vertices and edges have weights, this is the only way how workload and communication load and frequency can be specified (Section 3). The object graph is automatically partitioned by a graph partitioning software. The result of the partitioning is a number of subgraphs which form the input of the program (process)

the langauge representation of the graphical model of Figure 1. The communication structure and behavior of a system for the entrance management to a soccer stadion is modelled.

There are static classes *gates* with 4 instances (to control visitors), *ticket windows* with 20 instances where paying visitors get their tickets and *control points*, where seasonticket visitors are controlled and bought tickets are rendered invalid (line 3-5 figure 2). Objects of class *ticket window* share no communication (they are independent), the members of class *gate* communicate

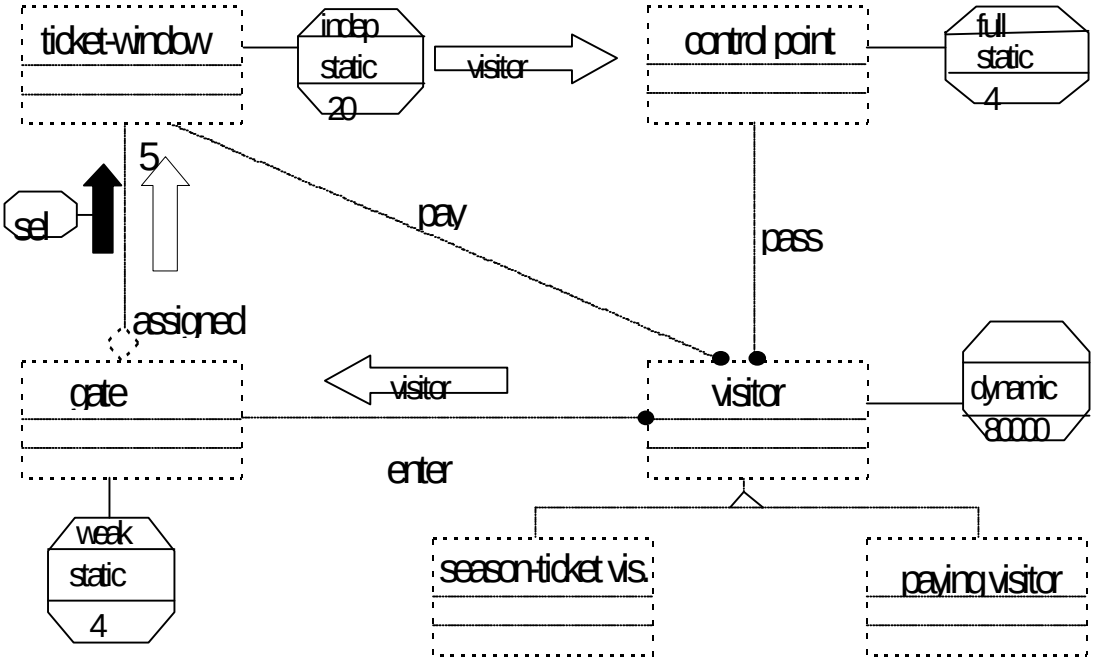

Figure 1: An omt-like simulation model extended with communication information

generation phase. In order to be able to link the objects together, the interfaces must fulfil some requirements. At last the user has to supply the implementations for those interfaces (the logic of the simulation) followed by a compilation step to get executable processes.

2. The Model Language

Figure 1 shows a graphical example of a possible simulation model which is extended with information about the communication structure [STOP 95]. We have not defined and implemented a graphic-based language. Our proposed language can be considered as a simple, typeless description-language which enables us to add the necessary topological information to any given model.

The main function of the language is to describe the communication behavior of all classes belonging to the model and their relations to each other. Classes are specified on an abstract level as a reference to an existing external definition which has to be supplied by the user. It is not the primary task of the language to specify internal and inheritance information of classes, only the communication structure is of interest. Figure 2 shows

partially and the instances of class *control point* are fully connected. For dynamic classes we only specify the number of instances and the starting point (which has to be a static class, in our case the *gate* objects, line 7-8 figure 2) where they enter the system. In the example there exists only one static relation between gates and ticket windows with no increase in workload (expressed by 0 in the range pattern, line 10-12 figure 2) and different edge costs between groups of nodes (expressed as ranges). Dynamic relations express the number of static objects to which a dynamic object has contact (line 14-18 figure 2). The first number in the relation expresses the workload for one static object which is contacted by the corresponding dynamic objects. As a consequence we increase the workload of each static object by the following formula ({number of all dynamic objects of one class} * {number of static objects of a class, contacted in this relation} / {number of all members of the static class} * {factor of workload}. Eg. for the relation in line 13 we get the following result

$$\text{workload} = \frac{3 * \textit{pvisitors} * 4}{4} * 1 = 60000$$

```
(* 1 *)  MODEL EntranceManagement
(* 2 *)   SCONST ticketwindows = 20; controlpoints = 4;
(* 3 *)   SCLASS gate [4]  2 ([1..1] <4> [2..4] [2..2] <3> [3..4] [3..3] <3> [4..4]) END;
(* 4 *)   SCLASS ticketwindow  [ticketwindows]  INDEPENDENT 1 END;
(* 5 *)   SCLASS controlpoint [controlpoints] FULL  4 <4> END;
(* 6 *)   SCONST pvisitors = 20000;
(* 7 *)   DCLASS seasonticketvisitor [3 * pvisitors]  PATHBEGIN gate  END;
(* 8 *)   DCLASS payingvisitor [pvisitors] PATHBEGIN gate  END;
(* 9 *)   SCONST gatecomm = 3;
(* 10 *) SRELATION  gate ticketwindow
(* 11 *)     0 ([1..1] <gatecomm> [1..5] [2..2] <gatecomm> [6..10] [3..3] <gatecomm> [11..15]
(* 12 *)     [4..4] <gatecomm> [16..20]) END;
(* 13 *) SCONST gateservice = 1; payservice = 7 * gateservice; controlservice = payservice;
(* 14 *) DRELATION seasonticketvisitor gateservice <4> gate END;
(* 15 *) DRELATION payingvisitor gateservice <4> gate END;
(* 16 *) DRELATION payingvisitor payservice <5> ticketwindow END;
(* 17 *) DRELATION seasonticketvisitor controlservice <4> controlpoint  END;
(* 18 *) DRELATION payingvisitor controlservice <4> controlpoint  END;
(* 19 *)
(* 20 *) END.
```

Figure 2: A model for the entrance management to a soccer stadion

This means that the overall workload caused by all members of the dynamic class is distributed evenly to all members of the static class (regular distributed workload maximizes the overall performance).

Based on the model the object graph for the problem is generated. The object graph O(V,E) consists of a set of vertices and a set of edges and is defined by the following rules:

(1) $\forall v \in V: W_v(v)$ is the weight of vertex v

(2) $\forall e \in E: W_e(e)$ is the weight of edge e

(3) edges are bidirectional, self edges are not allowed (same starting and end vertex)

(4) multiple edges between two nodes are not allowed (multiple edges can be expressed by one edge and by the sum of the single edge weights)

The weight of a vertex (workload) is interpreted as the cost of its work. Edge weights are the way to express frequency and the amount of data exchanged between vertices.

In Figure 3 the resulting object graph of the example modelled in Figure 2 is shown. The power index in a vertex means the workload (for reasons of clearness we have it specified for only one instance per class because in this example all instances of a certain class have the same workload).

3. The Optimization Phase

In section 2 the basic structure of *object graphs* has been defined. The criteria of the optimization strategy in our approach are to maximize local communication, minimize external communication, and load balance the work among processors of a distributed environment. An equivalent problem in graph theory is known as PARTITION and is defined for a Graph G(V,E) as follows:

- $G = G_1 \cup G_2 \cup ..... \cup G_n$
- $\forall G_i, G_j, i \neq j: G_i \cap G_j \equiv \Phi$
- $\sum_{v \in G_i} W_v(v) \approx \sum_{w \in G_j} W_v(w)$,
- $Ex \subset E: |\mathrm{Ex}| =$ minimal , Ex contains edges with vertices ending in different partitions

It was proved that this problem is NP-complete [GARE 76] and so heuristic approaches with polynomial runtime with solutions *nearby* the optimum are used to solve this problem.

In our approach a spectral bisection partitioner with a local refinement policy based on the Kernighan/Lin algorithm for the optimization phase is used [KERN 70, HEND 93, HEND 95].

If the user has developed the model there are two possibilities to control the optimization process:

1. Specification of the number of partitions to be created (this should correspond to the number of processors in the users distributed environment)

2. Specification of the upper bound of workload for partitions; based on this information the number of partitions matching this constraint is automatically generated

To be able to estimate the quality of a distribution we have defined the following measures:

For a partition we define the following functions:

- $Sv: P \rightarrow N \cong \sum_{i=1}^{|P|} Wv(i),\ P \subseteq V,$
  the workload of a partition
- $Si: P \rightarrow N \cong \sum_{j \in EP} We(j), P \subseteq V, EP \subseteq E,$
  $\{j: j = [x, y] \wedge x, y \in P \wedge x \neq y\}$, the amount of internal communication
- $Se: P \rightarrow N \cong \sum We(k), P \subseteq V, Q \subseteq V, P \cap Q \equiv \Phi,$
  $\{k: k = [x, y] \wedge x \in P \wedge y \in Q\}$, the amount of external communication
- $Sp: P \rightarrow N \cong Sv + Si + Se$, the full weight of a partition
- $SQ: N \rightarrow R \cong \frac{Se}{Si}$, $Si > 0$, *or* 0 if $Si = 0$, the communication ratio of external and internal communication

For the whole model we define:

- $V = \bigcup_{i=1}^{n} Pi,\ \forall Pi, Pj, i \neq j, 1 \leq i, j \leq n:$
  $Sv(Pi) \approx Sv(Pj)$

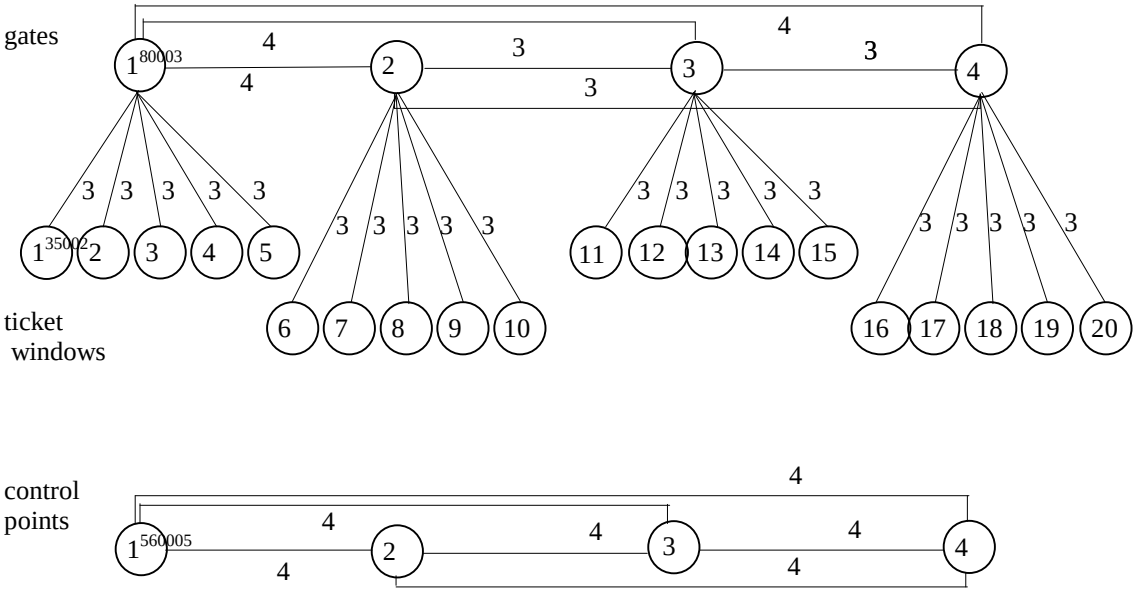

Figure 3: Object graph for the entrance management model

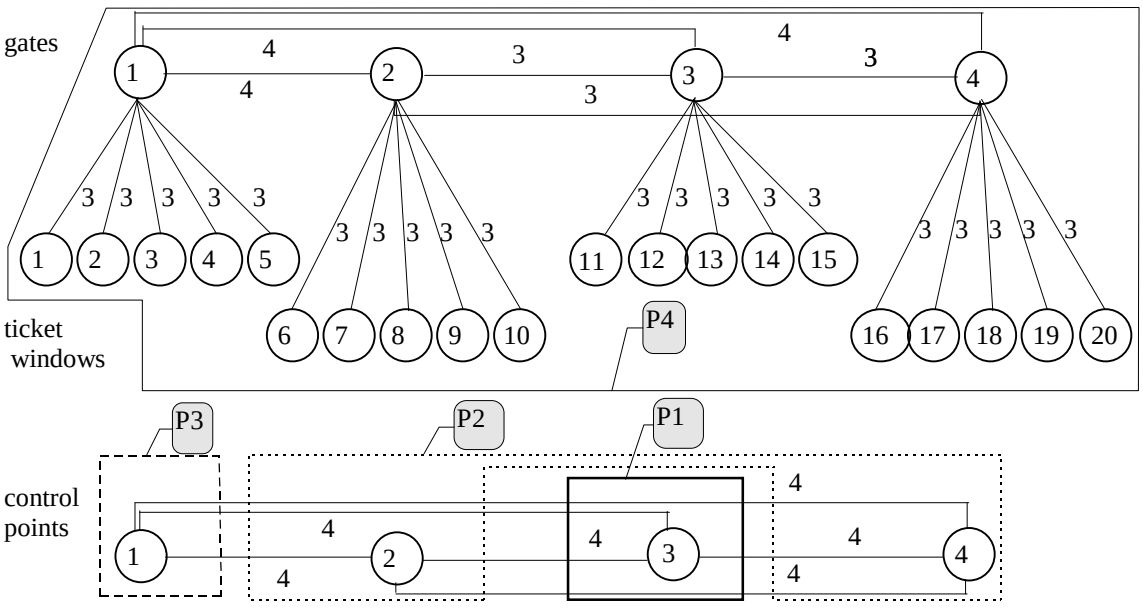

Figure 4: object graph decomposed into 4 partitions

- $MS_P: V \rightarrow N \cong \sum_{i=1}^{n} S_P(P_i)$, the whole weight of the model should be a minimum
- $MS_Q: R \rightarrow R \cong \frac{1}{n} \sum_{i=1}^{n} S_Q(P_i), n = |P_i|, P_i: S_i(P_i) \# 0$ the average partition communication ratio, the theoretical optimum for this measure is 0 (fully independent processes)

The results of partitioning the example object graph with the instruction to generate 4 partitions are shown in Figure 4, the numerical computations for this distribution are presented in Table 1.

	P1	**P2**	**P3**	**P4**
$\mathbf{S_v}$	560005	1120010	560005	1020052
$\mathbf{S_i}$	0	4	0	81
$\mathbf{S_e}$	12	16	12	0
$\mathbf{S_P}$	560017	1120030	560017	1020133
$\mathbf{S_Q}$	0	4	0	0

Table 1: Results of the optimization process

For the whole model we get:

MS_P: 3260197 (horizontal sum of line 4 in Table 1)
MS_Q: 4 (average of line 5 in Table 1)

The results for this small model are somewhat distorted regarding the balance of workload between partitions but this distortion stems from the small size of the model. We have only 28 vertices and 32 edges. The global minimum MS_P is truly a minimal value. We did also computations on larger models eg. 330 vertices and 7535 edges distributed over 8 partitions and we got an imbalance between the maximum and minimum workload which is less then 5 percent which conforms to the assumption of an even distributed workload.

4. Conclusions

In this paper an approach for the acceleration of distributed simulations has been presented. The approach uses graph-optimizing methods as optimization strategy. The mapping of a model onto an object graph with an optimization step has been shown. The key advantages of the presented approach are the reduction of development time, and a quasi optimal distribution of large models to distributed systems. We have developed a Compiler for the language which checks the syntax and semantics of models, computes the formulas of section 3 and generates an import and export table for objects in a certain partition. Exported objects are true instances in a given partition and imported objects are handles to objects which are created and exported by other partitions and activated by RPC. We have defined the structure and naming conventions for interfaces which are linked to the distributed program.

Currently we are dealing with the generation of the distributed program code. As the target language for our compiler we have chosen Modula-3 [NELS 91]. The communication between partitions (running on different hosts) is done by Network Objects [BIRR 94] - a powerful communication package which is also written in Modula-3. Our approach is independent from the underlying simulation environment, currently we are using a simulation package which implements the conservative strategy [MISR 86]. This package is part of a M.S. thesis. It is process-oriented and also written in Modula-3.

References

[BAEZ 90] D. Baezner, G. Lomow, B.W. Unger: „*Sim++: The transition to distributed simulation*“, Proc. of SCS Multiconference on distr. Simulation, San Diego, CA, 1990

[BARG 94] R.L.Bargodia, W.T.Liao:“*Maisie: A language for design of efficient discrete-event simulations*“, IEEE Transactions on SW-Engineering, April, 1994.

[BIRR 94] A. Birrell, G. Nelson et al: „Network Objects“, Digital Systems Research Center, Palo Alto, CA, 1994

[CHAN 81] K.M. Chandy, J. Misra: „*Distributed simulation via a sequence of parallel computations*“, Comm. of the ACM, Vol 24, No. 4, 1981

[GARE 76] M. Gare, D. Johnson, L. Stockmeyer: „Some simplified NP-complete graph problems“, Theoretical Computer Science, 1 (1976)

[HEND 93] B. Hendrickson, R. Leland: „*Multi- dimensional spectral load balancing*“, Proc. 6th SIAM Conf. Parallel Processing for scientific Computing, 1993

[HEND 95] B. Hendrickson, R. Leland: „*An improved spectral graph partitioning algorithm for mapping parallel computations*“, SIAM J. Sci. Computing, Vol 16, No. 2, 1995

[JEFF 82] D.R. Jefferson, H. Sowizral: „*Fast concurrent simulation using the time warp mechanism, part I, local control*“, Rand Corporation, Santa Monica 1982

[JEFF 82] D.R. Jefferson, H. Sowizral: „*Fast concurrent simulation using the time warp mechanism, part II, global control*“, Rand Corporation, Santa Monica 1983

[JEFF 85] D.R. Jefferson, H. Sowizral: „*Fast concurrent simulation using the time warp mechanism*“, Proc. of the SCS Distributed Simulation conference, San Diego, 1985

[KERN 70] B. Kernighan, S. Lin: „*An efficient heuristic procedure for partitioning graphs*“, Bell System Technical Journal 29, 1970

[MISR 86] J. Misra: „*Distributed discrete-event simulation*“, Computing Surveys, Vol. 18, No.1, 1986

[NELS 91] G. Nelson: „Systems Programming with Modula-3“, Prentice Hall Inc, Englewood Cliffs, NJ, 1991

[STOP 95] A. Stopper, L. Böszörmenyi: „*A Distributed, Object-Oriented Simulation System Based on Hints*“, Proc. of the EUROSIM Congress 95, Vienna, 1995