Object oriented parallel discrete event simulation : the PROSIT approach

Lionel Mallet, Philippe Mussi

Object Oriented Parallel Discrete Event Simulation: The **PROSIT** Approach

Lionel Mallet and Philippe Mussi

N° 2232

Avril 1994

PROGRAMME 1

Architectures parallèles,

bases de données,

réseaux et systèmes distribués

ISSN 0249-6399

Object Oriented Parallel Discrete Event Simulation: *The* **PROSIT** *Approach*

Lionel Mallet and Philippe Mussi [*]

Programme 1 — Architectures parallèles, bases de données, réseaux
et systèmes distribués
Projet Mistral

Rapport de recherche n˚2232 — Avril 1994 — 15 pages

Abstract: The *Prosit* project, launched in 1991, is devoted to the development of a new discrete event simulation system. This future system is currently designed with two main concerns in mind:

- performance on both conventional and multi-processor machines
- versatility and ease of use.

The project described in this report is a joint effort of Simulog, INRIA and GEC-Alsthom and has been partially supported by French Ministry of Research and Space. A version of this report has been presented at European Simulation Multiconference, Lyon (France), June 1993.

[*] {Philippe.Mussi},{Lionel.Mallet}@sophia.inria.fr

In particular, *Prosit* user programs (models and experiment descriptions) should need only a new compilation so as to run efficiently on either a sequential workstation, or a multi-processor machine.

In this paper, we focus on the design philosophy of *Prosit* , which is based on the object paradigm. First, we present the modeling process in *Prosit* , then we describe the model architecture, statistics collection and parallelism integration, and we give an insight into class hierarchy. Finally, *Prosit* future developments and prospects are discussed.

Key-words: Discrete event simulation, Distributed Simulation, Object Oriented Simulation

(Résumé : tsvp)

Simulation à événements discrets répartie orientée objets: *l'approche* PROSIT

Résumé : Le projet **PROSIT** , entamé en 1991, est consacré au développement d'un nouveau système de simulation à événements discrets basé sur le paradigme objets. L'orientation objets a été choisie pour répondre à deux critères principaux:

- la performance, à la fois sur des machines conventionnelles et sur des architectures multi-processeurs
- la versatilité et la facilité d'utilisation.

En particulier, les programmes-utilisateur *Prosit* (modèles et expériences) ne doivent nécessiter qu'une recompilation pour assurer une exécution efficace sur une station de travail classique, un *cluster* de stations de travail ou une machine multi-processeurs.

Ce rapport est centré sur la philosophie de conception de *Prosit* , qui est basé sur le paradigme objets. Nous présentons d'abord le processus de modélisation en *Prosit* , puis l'architecture des modèles, la collation de statistiques et l'intégration du parallélisme, et nous donnons un aperçu de la hiérarchie de classes. Enfin, nous présentons les développements futurs et les perspectives du projet.

Mots-clé : Simulation à événements discrets, Simulation répartie, Simulation orientée objets

1 Introduction

It is a well known fact that Discrete Event Simulation and Object Oriented Languages have a long common history. Many authors even consider Simula[Lam88], one of the first simulation languages, as the first real object oriented programming language. However, that common story did not stop back in 1967, as shown by simulation environments like ModSim2 or Sim++ [Ros92].

The authors' organisations have acquired a great experience in discrete event simulation by developing and enhancing the QNAP2[VP85] package, especially in the field of queueing networks. This package has incorporated, over the years, many up to date features, including object oriented enhancements, but its overall architecture made its extension to distributed simulation unlikely.

The *Prosit* project, described in this paper, is devoted to the development of a brand new discrete event simulation system, designed from the grounds up with distributed simulation in mind. Its design is based on the object paradigm, from which naturally derive several interesting features:

1. Modularity and reusability, allowing both *Prosit* programmers and end-users to develop high-level model libraries. These libraries may include sub-models for high-level subsystems, or highly optimised simulation classes for commonly used sub-systems.

2. Target independence: distributed simulation (in both optimistic and conservative variants) and parallel replication is implemented in such a way that application programmers do not have to take it into account. Their simulation classes inherit *parallel methods* only if needed, and the choice of sequential or parallel implementation will only be made at the final compilation stage. Furthermore, simulation classes and user programs are independent of the simulation method used.

3. Extensibility: various tools may be incorporated, with little or no changes in basic or user written classes. These tools include statistics gathering and processing, automatic load-balancing, animation, sub-model aggregation, analytical solvers, etc. . .

Chapter 2 focuses on the design of the *Prosit* system. The modeling process is presented first. Then we explicit the model architecture, and describe how measurements and statistics gathering and parallelism are integrated.

In chapter 3 are given some ideas of short-term and long-term evolution of the *Prosit* project, including its planned first implementations.

2 Design Issues

This section deals with *Prosit* major design issues. By *design issues* we intend simulator design as well as model design, i.e., our point of view as well as the user's point of view. We first describe the *simulator generation process*, and go on with describing the model architecture. Following is a discussion about statistics collection and parallelism, i.e. the simulator architecture on multi-processor environment.

2.1 Modeling Process

The key choice of *Prosit* was the will to use an existing language for the simulator. We did not want to design a new language for the purpose of simulation. By the way, this choice forced the modeling process to be a compilation process. Furthermore, our wish to use an object-oriented language to benefit from the well known advantages of such languages completely defined the modeling process as being an assembly of class instances, using dedicated class libraries and user-defined classes suited to the problem to model.

The whole process is described by Figure 1. The *Prosit* project will define a set of simulation and modeling classes (simulation classes are those dealing with the simulation phase whereas modeling classes are those used to build the model). These classes will be referred to as base classes. To mask the simulation paradigm to the final user of the simulator, a library programmer will define a set of classes for a specific field of application. These classes, gathered in a library, will allow to build a model at a higher description level than the one corresponding to the simulation paradigm, i.e., by manipulating *processors* instead of *FIFO servers*. This possibility to define high level libraries is crucial if we want the simulator to be usable by engineers instead of just mathematicians. In fact, the simulation paradigm itself is contained in the description and code of the modeling classes, and hence