ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

MSc in Computer Science and Engineering

Fancy Title

Thesis in:
SOFTWARE PROCESS ENGINEERING

Supervisor
Prof. Danilo Pianini

Candidate
Stefano Furi

IV Graduation Session
Academic Year 2024-2025

Abstract

Max 2000 characters, strict.

Contents

Abstract **iii**

1 Introduction **1**

1.1 Simulation 2

1.1.1 Discrete-Event System Simulation 3

1.1.2 The Stochastic Simulation Algorithm 4

9

Bibliography **9**

List of Figures

1.1 System, environment, and state evolution in discrete-event modeling. 2

List of Listings

introintro

Chapter 1

Introduction

The scientific method is characterised by the formulation of a *theory* which needs to be proven as false or true. This forms the first scientific pillar, which is the theoretical assumption and knowledge used to describe a certain phenomena, a property or a behaviour. The process of gathering evidence to support the validity of the theory is what *experiments* try to accomplish, namely a series of observations of the real world that confirm or refute the initial hypothesis. In the modern era of sciences, computer systems backs the vast majority of modern scientific discoveries, whether through the usage of the raw computing power (e.g. weather forecasting equations resolution through Massively Parallel Computers) of such systems or by exploiting virtual representations of the real world [PV05]. With the widespread adoption of simulation and simulators in the scientific method, researchers are now considering simulation as the *third pillar* of science which complements theory and experiments. Some argues that simulation cannot be compared to experimentation, while others claim that there is more in common between simulation and experimentation [Win10]. In the general scientific method framework, theory provides the model of the hypothesis, while through experiments some may validate those models against reality, while simulation can serve as sources of new insights or new hypothesis which will be tested experimentally. Therefore, simulation is essential in most scientific development workflows, and modern literature has plenty of successful applications of simulations in various fields from biology to physics and from chemistry to social sciences.

1.1 Simulation

A *simulation* is the imitation of the operation of a real-world process or system over time [BCNN10]. A simulation is built upon a *model* described through a set of mathematical, logical and symbolical relationships between entities in the system. The model is then simulated inside the system over time, making it possibile for observers to make analysis and prediction of model's impact on system performance. Moreover, the model is defined as a representation of a system for the purpose of studying that system, meaning that the set of characteristics of the model will include only those details useful in expanding or exploring the knowledge of the original problem. Simulation can also serve the purpose in giving insights on the model's performance with respect to the system, actively driving the design process of such model before it is even built in reality. A *system* in simulation terms, is defined as a set of objects which can interact with each other toward the accomplishment of some purpose. An important separation must be drawn in the boundary between the system and its environment, where the latter can influence the system (changes in the *system environment*) and viceversa.

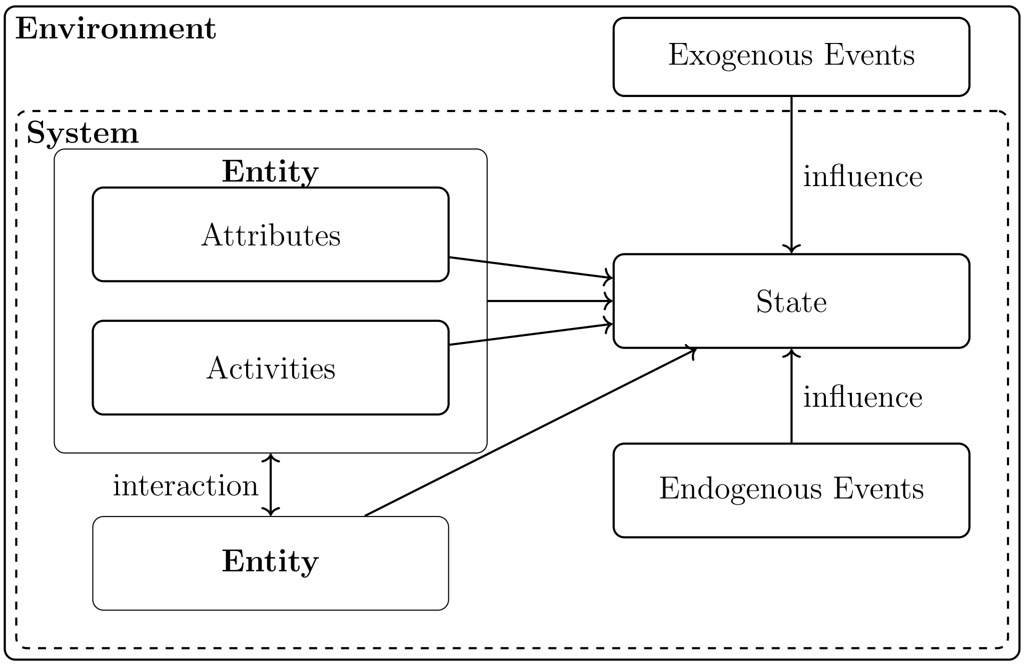

Figure 1.1: System, environment, and state evolution in discrete-event modeling.

Inside the environment, *entities* interact with each other and can be described by a set of *attributes* and their behaviour is represented by *activities* which last over time. The set of variables on a certain time instant during a simulation is used to define the system *state*, which in turn can be influenced by internal (*endogenous*) or external (*exogenous*) events. The interaction between the components is summarised in Figure 1.1.

Furthermore, a simulation could be characterised by the time span or the time instant in which a system is investigated, talking respectively about dynamic or static (Monte Carlo simulation) simulation models. In addition to that, some models may be represented by a deterministic set of input variables, whereas the vast majority of interesting problems may be expressed by means of stochastic models.

1.1.1 Discrete-Event System Simulation

Discrete-Event System Simulation is the modeling of systems in which the state variable changes only at a discrete set of points in time. The model is evaluated by means of *numerical methods*, i.e. system behaviours are approximated through computational procedures, rather than "solving" the equations representing system behaviours. Most applications of *Discrete Event Simulation* (DES) deals exclusively with dynamic, stochastic systems that change in a discrete manner, i.e. the state variable changes only at a discrete set of points in time. Those points are represented by events occurring. This means that a DES proceeds by producing a sequence of system snapshots that represent the evolution of the system through time including, for each t so that CLOCK $= t$, the state at time t along with the list of all activities in progress - the *Future Event List* (FEL).

Event Scheduling

Event scheduling and the advancement of time is a mechanism to advance simulation time, while granting that the chronological order of events is correct In the context of stochastic discrete simulations, scheduling an event means inserting an event in the FEL computing its duration by drawing or calculating a sample value from a statistical distribution. In this way at any given time the FEL is strictly ordered by event time (i.e. chronologically). After the state is updated at the t-th snapshot, the CLOCK advances at simulation time CLOCK $= t_1$, the next event is removed from the FEL and executed, effectively creating the snapshot for time t_1. This procedure repeats until the simulation is over.

While the event scheduling algorithm provides the logical framework for DES, its computational *efficiency* is governed by how the FEL is managed. In systems with a massive number of events such as biochemical systems, the overhead of searching, updating and keeping the FEL sorted becomes easily the primary bottleneck.

1.1.2 The Stochastic Simulation Algorithm

While general DES frameworks often rely on empirical distributions for event timings, the *Stochastic Simulation Algorithm* (SSA) [Gil77] provides a physically grounded method for simulating the time-evolution of chemical reaction networks. The "Direct Method" of the SSA relies on the *Markovian* property: the probability of a reaction occurring depends solely on the current molecular population (the state) rather than the history of the system.

In this framework, each reaction R_μ is assigned a *propensity function* a_μ. Assuming the system is well-stirred and at constant volume, the time to the next reaction τ is an exponential random variable with a cumulative rate equal to the aggregate propensity $\lambda = \sum_j a_j$. The *Probability Density Function* (PDF) of this timing is given by:

$$P(\tau) = \lambda e^{-\lambda\tau} \tag{1.1}$$

The specific reaction μ that occurs at that time is a discrete random variable chosen with probability:

$$P(\text{reaction} = \mu) = \frac{a_\mu}{\lambda} \tag{1.2}$$

To simulate this, two independent random numbers $r_1, r_2 \in (0, 1]$ are sampled from a uniform distribution. The time step is then computed via the *Inverse Transform Sampling* as $\tau = -\ln(r_1)/\lambda$, while μ is the smallest integer satisfying

$\sum_{j=1}^{\mu} a_j > r_2\lambda$.

Optimising SSA

The SSA can be then summarised in Algorithm 1. Right away some performance issues arise by looking at the pseudocode:

- Every propensity a_j is recomputed even if the reaction did not involve those species;
- All propensities have to be summed all together at every step;
- If reactions are stored in a list-like structure, finding the next reaction μ to execute is done with linear complexity.

Algorithm 1 Stochastic Simulation Algorithm (Direct Method)

```
 1: procedure SSA(initial state $\mathbf{x}$, time $T_{end}$)
 2:     $t \leftarrow 0$
 3:     while $t < T_{end}$ do
 4:         for each reaction $R_j$ do
 5:             Calculate propensity $a_j$ based on $\mathbf{x}$
 6:         $\lambda \leftarrow \sum a_j$
 7:         Sample $r_1, r_2 \sim \text{Uniform}(0,1)$
 8:         $\tau \leftarrow \frac{-\ln(r_1)}{\lambda}$                    ▷ Time to next event
 9:         Find smallest $\mu$ such that $\sum_{j=1}^{\mu} a_j > r_2\lambda$     ▷ Select event
10:         Update state $\mathbf{x}$ according to reaction $R_\mu$
11:         $t \leftarrow t + \tau$
```

In short, SSA complexity is $O(M)$ with M being the number of reactions in the system. In addition to that, the direct method takes time proportional to the number of reactions to calculate λ and to generate a random number according to **??**.

Optimising propensity updates It is possibile to optimise the execution of such algorithm by noting that, in most systems, chemical reactions are *sparse*, hence only a small subset of propensities may change from step to step. If a reaction A consumes species X, only those concentration that also use X will be affected. It is possible to model such influence relationships among reactions by means of a *Dependency Graph* $\mathcal{G} = (V, E)$ where:

- V is the set of reactions;
- An edge $e_{i,j} \in E$ exists from $R_i \in V$ to $R_j \in V$ if the execution of the reaction R_i changes the set of substances involved in R_j.

With this data structure we cut the cost of updating $M = |V|$ propensities at every iteration with just a small constant number of them.

Optimising scheduling As already said, a major bottleneck in simulating such systems is the FEL management: in both the direct method and in the *First Reaction Method* [Gil76] linear scans are performed to find the next reaction μ to execute, where in the latter the reaction with the smallest *putative time* τ_j should be chosen. Starting from this idea, in [GB00] the *Next Reaction Method* is presented by introducing four important properties:

1. The propensity function a_j is stored along with the putative time τ_j. If a_j is not recomputed, nor does τ_j;
2. A dependency graph $\mathcal{G} = (V, E)$ is stored which will help telling precisely which set of reactants to change when a given reaction is executed.
3. The list-like data structure backing the FEL is replaced with an *Indexed Priority Queue* (IPQ): it is a binary heap combined with a map, which maximises efficiency for sparse operations and a small number of updates, granting $O(1)$ access and a sorting cost of $O(\log M)$.
4. *Random number reuse*: This property addresses the computational cost of generating high-quality random numbers (could be the most heavy task in simulation). In the direct method, every step requires two new random numbers. In the Next Reaction Method, if a reaction R_j is *not* affected by the execution of R_μ, its absolute putative time T_j is simply kept in the FEL. However, if R_j is affected (its propensity changes from a_j to a_j^{new}), its new

putative time T_j^{new} is rescaled to maintain statistical exactness:

$$T_j^{new} = t + \frac{a_j}{a_j^{new}}(T_j^{old} - t) \tag{1.3}$$

This transformation preserves the Markovian property because the exponential distribution is memoryless. Therefore, a new random number is only required for the reaction that just fired, while all others are updated algebraically.

With the above elements, the *Next Reaction Method* transform a linear, brute-force search into a targeted, logarithmic update cycle. Through the Dependency Graph it is possibile to identify which reactions in the IPQ need to be modified when a reaction executes. With the latter it is possibile to perform a lookup of a given reaction in constant time, and update its putative time using **??** which drastically reduce the time spent in computing new random values. Finally, the heap structure of the IPQ restores the chronological order of the FEL in logarithmic time, ensuring the positioning of the next event at the root of the tree.

Algorithm 2 Next Reaction Method (Gibson-Bruck)

```
1: procedure NextReactionMethod(initial state $\mathbf{x}$, dependency graph $\mathcal{G}$)
2:     $t \leftarrow 0$
3:     Generate $a_j$ and putative times $T_j$ for all $R_j$    ▷ Initial expensive step
4:     Build IPQ containing all $T_j$
5:     while $t < T_{end}$ do
6:         $(\mu, T_\mu) \leftarrow$ IPQ.top()    ▷ $O(1)$ access to next event
7:         $t \leftarrow T_\mu$    ▷ Advance CLOCK to absolute event time
8:         Update state $\mathbf{x}$ according to reaction $R_\mu$
9:         Sample new $r$ and update $T_\mu = t - \frac{\ln(r)}{a_\mu}$    ▷ New time only for reaction that fired
10:        for each reaction $R_j$ in $\mathcal{G}$ affected by $R_\mu$ do
11:            if $j \neq \mu$ then
12:                $a_j^{new} \leftarrow$ update propensity using $\mathbf{x}$
13:                $T_j \leftarrow t + \frac{a_j^{old}}{a_j^{new}}(T_j - t)$    ▷ Reuse/rescale random numbers
14:                IPQ.update$(j, T_j)$    ▷ $O(\log M)$ heap maintenance
```

A simplified version of the algorithm is then presented in Algorithm 2, which shows that for each step we pay $O(\log M)$ for the heap updates and $O(\deg(\mathcal{G}))$ for the dependency graph traversal.

Bibliography

[BCNN10] Jerry Banks, John S. Carson, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation*. Prentice-Hall, 5 edition, 2010.

[GB00] Michael A. Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The Journal of Physical Chemistry A*, 104(9):1876–1889, 2000.

[Gil76] Daniel T Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976.

[Gil77] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[PV05] Douglass E. Post and Lawrence G. Votta. Computational science demands a new paradigm. *Phys. Today*, 58(1):35–41, January 2005.

[Win10] Eric Winsberg. *Science in the age of computer simulation*. University of Chicago Press, Chicago, 1 edition, 2010.