1 Modeling and Simulation

A. Modeling and Simulation, a short review

A.1. How to Study a System

A system can be anything. It can be blood flow in the aorta, it can be the architecture of the internet, it can be the Golden Gate bridge, it can be the solar system. A system is an isolated part of reality[1] that we wish to study through scientific inquiry. According to Bernard Siegler 'a system is a potential source of data' [3]. We study the system by experimenting with it, where, according to François Cellier, 'an experiment is the process of extracting data from a system by exerting it through its inputs'. [4] This sounds very theoretical, but you can easily come up with examples of systems and experiments to study them (do this, write down some examples!). Figure 1 provides a schematic overview of ways to study a system.

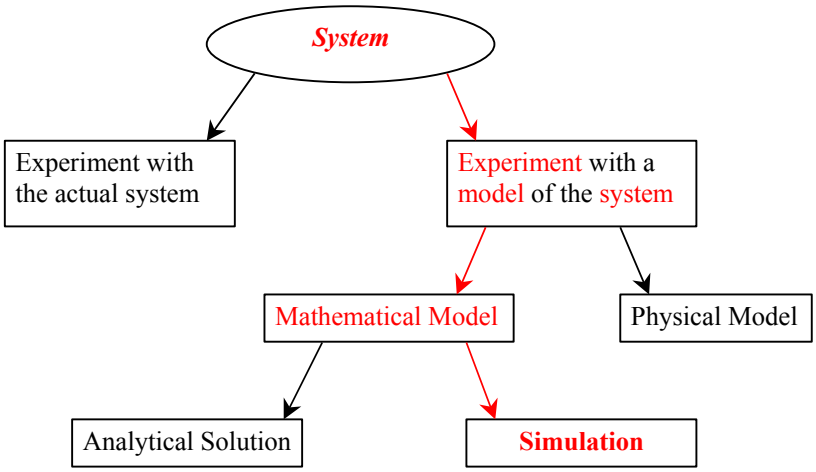

Figure 1: Several ways to study a system through experimentation.

First we may decide to experiment with the actual system. This may be possible for some systems (e.g. blood flow in the aorta may be measured with Magnetic Resonance Imaging, and we may study e.g. the influence of exercise of a person on the flow), but in other cases this may not be so obvious (how would you experiment with the solar system?). In many cases it is more natural to first construct a model of the system under study and next perform experiments on the model. So, you could study say the influence of a tornado on the mechanical vibrations in the Golden Gate bridge by actually building a small scale physical model of it and putting that in a wind tunnel to see what happens. However, you could also build a *mathematical model* of your system. For instance, for the Solar systems you could write down Newton's equation of motion for the main bodies in the solar system and then try to solve these equations.

In general, according to Marvin Minski, 'a Model (M) for a system (S) and an experiment (E) is anything to which E can be applied in order to answer a question about S'. [5] So, it should be clear by now that a model is not necessarily a computer program. However, in this lecture we will only study models that can be expressed as computer programs, so-

[1] Or may be even more than that, it may also be a construction of the mind, such as e.g. an abstract Mathematical theory.

called *Mathematical Models*. As the model replaces the original system we must always ask the question if this representation is accurate enough for our purposes. In other words, modeling always requires the act of *validation*, where model validation always relates to an experiment performed on the original system.

Now, mathematical models allow theoretical investigation, and in some (unfortunately not too many) situations it turns out that the mathematical equations can be solved analytically. We will encounter some situations of mathematical models that can be solved analytically later in this reader. However, this is rare, and in many cases one must resort to the final possibility, i.e. *Simulation*, where we may define simulation as 'an experiment performed on a mathematical model'. [6] Again note the importance of validation. It is easy to perform an experiment on a model in a range of parameters where the model is no longer valid, i.e. where the model is no longer an accurate representation of the system under study. In that case the simulation will produce results, but they have no meaning whatsoever (garbage in, garbage out).

So, within this framework, Modeling and Simulation provides a clear route through Figure 1 and currently is a well-established field in Science and Engineering. Let us now zoom in a little more on the modeling and simulation cycle, see Figure 2. A domain specific problem (a system) is mapped on a conceptual model, which is then mapped on a solver layer. The solvers can be of different types (*Numerical*, *Direct*, or *Natural*) and in the following chapters you will encounter these different types in much detail (remember, Scientific Computing was the glue between the model and the actually running simulation). Next, the solvers are mapped on a virtual machine model. This can be e.g. a message passing parallel computing model (e.g. SPMD in MPI, see [1]), or the data parallel model, or the bulk synchronous parallel model, etc. Finally, the virtual machine model is mapped onto the actual hardware where the simulation is executed.

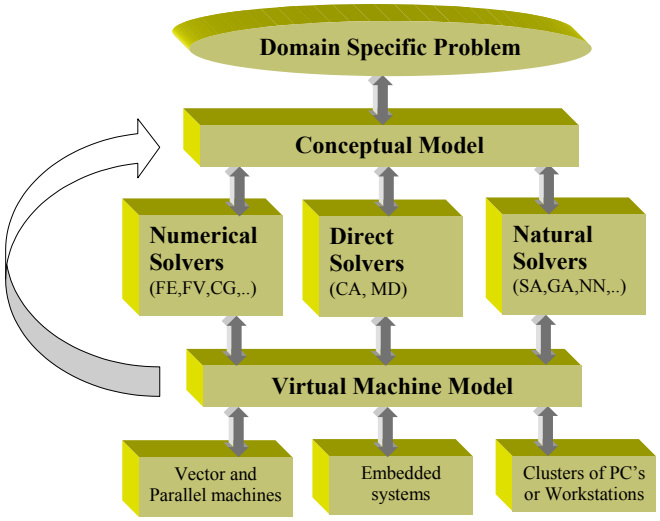

Figure 2: The Modeling and Simulation Cycle.

In this cycle many feedback loops will exist, but in this lecture the most interesting one is how a specific virtual machine model (read, the assumption of parallel computing in the SPMD paradigm) influences the models and the solvers. Or, is it possible to find models and solvers that are inherently parallel, and therefore allow a straightforward mapping on a parallel computer. In this lecture we will study a large collection of models and solvers, and constantly address this question.

A.2. Types of Mathematical Models

Although many ideas exist on how to distinguish between different mathematical models, and although certainly no clear consensus exists as of today, a generally well accepted distinction is in the following three types:

- *Continuous time models*;

- *Discrete time models*;
- *Discrete event models*.

The main distinction lies in the way the state of the model changes as a function of time, see Figure 3.

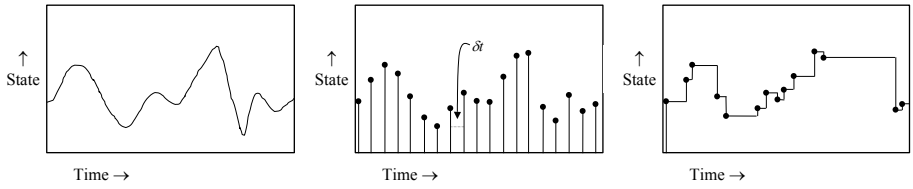

Figure 3: Trajectory of a continuous time - (left), discrete time - (middle), and discrete event model (right).

In continuous time models the state of a system changes continuously over time. These types of models are usually represented by sets of differential equations. With discrete-time models, the time axis is discretised. The system state changes are commonly represented by difference equations. These types of models are typical to engineering systems and computer-controlled systems. They can also arise from discrete versions of continuous-time models. The time-step used in the discrete-time model is constant. In discrete-event models, the state is discretised and "jumps" in time. Events can happen any time but only every now and then at (stochastic) time intervals. Typical examples come from "event tracing'" experiments, queuing models, Ising spin simulations, image restoration, combat simulation, etc.

B. Discrete Event Simulation

B.1. A Definition

We have seen that in continuous systems the state variables change continuously with respect to time, whereas in discrete systems the state variables change instantaneously at separate points in time. Unfortunately for the computational scientist there are but a few systems that are either completely discrete or completely continuous, although often one type dominates the other in such hybrid systems. The challenge here is to find a computational model that mimics closely the behavior of the system, specifically the simulation time-advance approach is critical.

If we take a closer look into the dynamic nature of simulation models, keeping track of the simulation time as the simulation proceeds, we can distinguish between two *time-advance* approaches: *time-driven* and *event-driven*.

1. *Time-Driven Simulation*

In time-driven simulation the time advances with a fixed increment, in the case of continuous systems. With this approach the simulation clock is advanced in increments of exactly Δt time units. Then after each update of the clock, the state variables are updated for the time interval $[t, t+ \Delta t]$. This is the most widely known approach in simulation of natural systems. Less widely used is the time-driven paradigm applied to discrete systems. In this case we have to consider whether:

- The time step Δt is small enough to capture every event in the discrete system. This might imply that we need to make Δt arbitrarily small, which is certainly not acceptable with respect to the computational times involved.
- The precision required can be obtained more efficiently through the event-driven execution mechanism. This primarily means that we have to trade efficiency for precision.

2. *Event-Driven Simulation*

In event-driven simulation on the other hand, we have the next-event time advance approach. Here (in case of discrete systems) we have the following phases:

Step 1 The simulation clock is initialized to zero and the times of occurrence of future events are determined.

Step 2 The simulation clock is advanced to the time of the occurrence of the most imminent (i.e. first) of the future events.

Step 3 The state of the system is updated to account for the fact that an event has occurred.

Step 4 Knowledge of the times of occurrence of future events is updated and the first step is repeated.

The nice thing of this approach is that periods of inactivity can be skipped over by jumping the clock from *event time* to the *next event time*. This is perfectly save since – per definition – all state changes only occur at event times. Therefore causality is guaranteed. The event-driven approach to discrete systems is usually exploited in queuing and optimization problems. The following sections will discuss in detail Discrete Event Simulation (DES) and Parallel Discrete Event Simulation (PDES).

C. References

1. Hoekstra, A.G.: Introduction Parallel Computing. Faculty of Science, University of Amsterdam, The Netherlands, (2001)
2. Sloot, P.M.A.: Simulation and Modelling. Faculty of Science, University of Amsterdam, The Netherlands, (2002)
3. Siegler, B.: Theory of Modeling and Simulation. (1976)
4. Cellier, F.: Continuous System Modeling. (1990)
5. Minski, M.: Models, Mind, Machines. (1965)
6. Korn, G.,Wait, J.: Digital Continuous System Simulation. (1978)
7. Sloot, P.M.A.: Computational Physics: Stochastic Simulation. Faculty of Science, University of Amsterdam, The Netherlands, (2002)
8. Hooper, J.W.: Strategy related characteristics of discrete event languages and models. Simulation **46** (1986) 153-159
9. Banks, J., Carson, J.S.,Nelson, B.L.: Discrete-Event System Simulation. Prentice Hall, (1999)
10. Chandy, K.M.,Misra, J.: Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. IEEE Transactions on Software Engineering **SE-5** (1979) 440-452
11. Chandy, K.M.,Misra, J.: Asynchronous distributed simulation via a sequence of parallel computations. Communications of the ACM **24** (1981) 198-205
12. Misra, J.: Distributed discrete event simulation. ACM Computing Surveys **18** (1986) 39-65
13. Fujimoto, R.M.: Performance measurements of distributed simulation strategies. Trans. Soc. Comput.Sim. **6** (1989) 89-132
14. Jefferson, D.R.: Virtual Time. ACM Transactions on Programming Languages and Systems **7** (1985) 404-425