1.2.3 Different Kinds of Simulations

There are a lot of ways to classify simulation models, but one useful way is along these three dimensions:

- **Static vs. Dynamic:** Time doesn't play a natural role in static models but does in dynamic models. The Buffon Needle Problem, described at the beginning of Section 1.3.1, is a static simulation. The small manufacturing model described in Chapters 2 and 3 is a dynamic model. Most operational models are dynamic; Arena was designed with them in mind, so our primary focus will be on such models.
- **Continuous vs. Discrete:** In a continuous model, the state of the system can change continuously over time; an example would be the level of a reservoir as water flows in and is let out, and as precipitation and evaporation occur. In a discrete model, though, change can occur only at separated points in time, such as a manufacturing system with parts arriving and leaving at specific times, machines going down and coming back up at specific times, and breaks for workers. You can have elements of both continuous and discrete change in the same model, which are called *mixed continuous-discrete models*; an example might be a refinery with continuously changing pressure inside vessels and discretely occurring shutdowns. Arena can handle continuous, discrete, and mixed models, but our focus will be on the discrete.
- **Deterministic vs. Stochastic:** Models that have no random input are deterministic; a strict appointment-book operation with fixed service times would be an example. Stochastic models, on the other hand, operate with random input—like a bank with randomly arriving customers requiring varying service times. A model can have both deterministic and random inputs in different components; which elements are modeled as deterministic and which as random are issues of modeling realism. Arena easily handles deterministic and stochastic inputs to models and provides many different probability distributions and processes that you can use to represent the random inputs. Since we feel that at least some element of uncertainty is usually present in reality, most of our illustrations will involve

1.3.2 Programming in General-Purpose Languages

As digital computers appeared in the 1950s and 1960s, people began writing computer programs in general-purpose procedural languages like FORTRAN to do simulations of more complicated systems. Support packages were written to help out with routine chores like list processing, keeping track of simulated events, and statistical bookkeeping.

This approach was highly customizable and flexible (in terms of the kinds of models and manipulations possible), but also painfully tedious and error-prone since models had to be coded pretty much from scratch every time. (Plus, if you dropped your deck of cards, it could take quite a while to reconstruct your "model.") For a more detailed history of discrete-event simulation languages, see Nance (1996).

1.3.3 Simulation Languages

Special-purpose *simulation languages* like GPSS, SIMSCRIPT, SLAM, and SIMAN appeared on the scene some time later and provided a much better framework for the kinds of simulations many people do. Simulation languages have become very popular and are in wide use.

Nonetheless, you still have to invest quite a bit of time to learn about their features and how to use them effectively. And, depending on the user interface provided, there can be picky, apparently arbitrary, and certainly frustrating syntactical idiosyncrasies that bedevil even old hands.

1.3.4 High-Level Simulators

Thus, several high-level "simulator" products emerged that are indeed very easy to use. They typically operate by intuitive graphical user interfaces, menus, and dialogs. You select from available simulation-modeling constructs, connect them, and run the model along with a dynamic graphical animation of system components as they move around and change.

However, the domains of many simulators are also rather restricted (like manufacturing or communications) and are generally not as flexible as you might like in order to build valid models of your systems. Some people feel that these packages may have gone too far up the software-hierarchy food chain and have traded away too much flexibility to achieve the ease-of-use goal.

1.3.5 Where Arena Fits In

Arena combines the ease of use found in high-level simulators with the flexibility of simulation languages, and even all the way down to general-purpose procedural languages like the Microsoft® Visual Basic® programming system or C if you really want. It does this by providing alternative and interchangeable *templates* of graphical simulation modeling-and-analysis *modules* that you can combine to build a fairly wide variety of simulation models. For ease of display and organization, modules are typically grouped into *panels* to compose a template. By switching panels, you gain access to a whole different set of simulation modeling constructs and capabilities. In most cases, modules from different panels can be mixed together in the same model.

Arena maintains its modeling flexibility by being fully *hierarchical*, as depicted in Figure 1-2. At any time, you can pull in low-level modules from the Blocks and Elements panel and gain access to simulation-language flexibility if you need to and mix in SIMAN constructs together with the higher-level modules from another template. For specialized needs, like complex decision algorithms or accessing data from an external application, you can write pieces of your model in a procedural language like Visual Basic or C/C++. All of this, regardless of how high or low you want to go in the hierarchy, takes place in the same consistent graphical user interface.

In fact, the modules in Arena are composed of SIMAN components; you can create your own modules and collect them into your own templates for various classes of systems. For instance, Rockwell Software (formerly Systems Modeling) has built templates for general modeling (the Arena template, which is the primary focus of this book), business-process re-engineering, call centers, and other industries. Other people have built templates for their company in industries as diverse as mining, auto manufacturing, fast-food, and forest-resource management. In this way, you don't have to compromise between modeling flexibility and ease of use. While this textbook focuses on modeling with the Arena template, you can get a taste of creating your own modules in Chapter 9.