2 Modeling and Simulation Method

2.1 INTRODUCTION

Modeling and simulation (M&S) projects vary widely according to the nature of what is represented, the simulation paradigm employed, the modeling purpose, and the size of the modeling effort. Therefore, modeling processes and methodologies employed can be very different across projects. For example, a continuous model of traffic patterns might be constructed within a few days by a single modeler to illustrate the effects of a road maintenance project over a two-year period. On the other hand, a large discrete event model for managing automobile manufacturing supply chains might be developed and maintained by a large modeling team.

Simulation is used to represent the behavior of systems, processes, or scenarios. A system view is concerned with inputs to a system boundary, their transformation, and outputs across the system boundary. A process view is concerned with a series of steps, each step having inputs, performing a transformation, and producing an output. Scenarios in the context of training or gaming describe an environment with situational factors and actions performed. This book is primarily concerned with modeling and simulating real systems and processes, though it is not meant to exclude scenarios. In this book, a reality represented in a model may be referred to as a system, a system or process, a subject target, or a subject.

Understanding behavior is the usual reason for M&S. Even a simple process can have complex behavior, but simulation models often become complex because they represent systems or processes composed of many elements having many relationships and the overall behaviors are complex. Real complexity can complicate the modeling unnecessarily unless a disciplined approach is used.

The basis for a disciplined approach is the sequence of modeling stages discussed in this chapter with the caveat that modeling is not a strictly sequential process of engagement, specification, construction, verification and validation, experimentation, and reporting shown in Figure 2.1. Rather, a modeler undertakes these stages iteratively. For example, during engagement, a modeler may realize that a particular part of a system would be most difficult to represent, and so may specify that part at a high level of abstraction for discussion with the sponsor. The modeler may even construct a small model to take back to the sponsor for elucidation of the system workings and validation of modeling scope. Elaboration of the model would proceed with further iterations that respect the sequence of the stages.

Despite the wide variety of modeling projects, a general method for modeling and simulation can be described. As it is described, the need for specialized knowledge and skills will become apparent, therefore these are discussed later in this chapter.

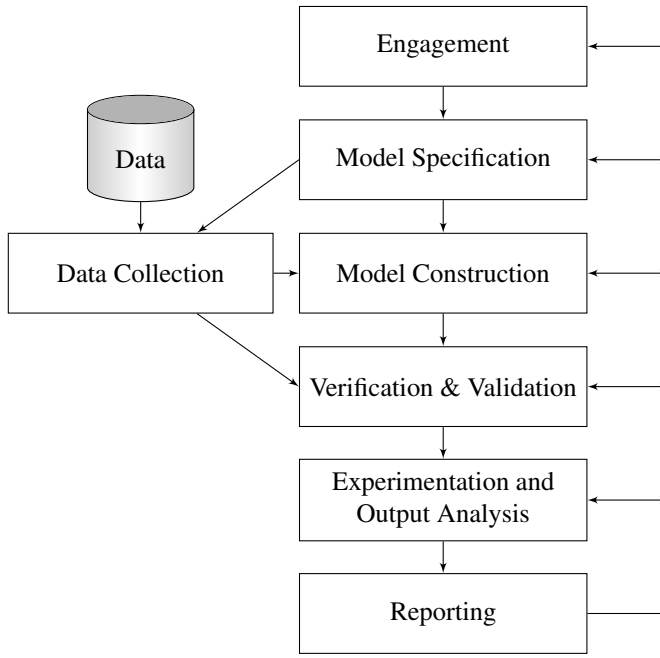

FIGURE 2.1: Modeling Process

2.2 ENGAGEMENT

When engaging with a customer for a modeling project, the customer may not have been exposed to M&S studies and so may not know how an M&S study can help. In this case, the modeler must spend time with the customer discussing what problems the customer perceives, why the customer thinks an M&S study may help, what can be represented, and how an M&S study can actually help. The customer is also likely to want to know about the modeler's experience and the cost and benefits of previous studies. If the customer is familiar with M&S studies, then the customer may have realistic expectations of the possibilities and the initial conversation can proceed directly to a particular problem, which must be important enough to the customer to warrant an M&S study.

Once a modeler and customer decide to engage in a modeling study, several points are important. The first of these is identifying the stakeholders, including sponsors (those who manage the funding), process or system owners (those responsible for managing the real system or process and the people in it), and the process or system participants (the people who do the work). Beyond a cognizant engineering specialist, stakeholders may include other engineers and scientists in related disciplines using the simulation results, managers and executives deciding on product options, providers of input data, system or product testing, marketing and sales, finance, hu-

man resources, investors, government and regulatory agencies, or others impacted depending on the modeling context.

Each stakeholder may have a different perspective on the reality to be represented and the modeler's task is to reconcile their viewpoints using a model. The modeler will have to facilitate discussions that elicit information about the reality in order to represent its structure, its inputs, and its expected and actual outputs. The modeler must also negotiate a model purpose, project budget and schedule, access to each group of stakeholders, access to data, and a model specification including model scope and level of abstraction.

In cases where a modeler has extensive experience in the domain that he/she is modeling, then he/she may not have to rely heavily on domain experts. Usually, though, a modeler who is engaging with a customer must draw heavily upon the experts in the customer organization. The best experts for an M&S study can be those who directly manage the people participating in a system or process because they are the most informed. In an engineering organization, these experts are often technical leads or first-level managers.

Beyond initial sessions in which the modeler elicits information about the system or process, the modeler must ensure that access is available for follow-up discussion of questions that arise during model construction, verification and validation, and experimentation. At the same time, the modeler must keep stakeholders informed of the study and its progress, mindful that stakeholders unfamiliar with M&S may not appreciate the complexity of their own systems and challenges in representing them.

The preceding discussion can be summarized in the following list of questions that must be answered when engaging for an M&S project or study:

- Who are the (groups of) stakeholders and what is the primary interest of each with regard to the modeling project?
- What are the roles of those involved in the project?
- Who will ensure that resources are available for the project or study?
- Who needs to be kept informed of progress and issues?
- Who will the modeler depend on for providing information and quantitative data about the real system or process? What is their availability?
- What sources of data are available?
- How much time is available for the M&S project or study?
- What budget is available for the M&S project or study?

2.3 MODEL SPECIFICATION

Model specification usually begins with identifying a problem and describing its context as a basis for specifying a modeling purpose. The modeling purpose will include the intended audience for the modeling results, the desired set of policies to be simulated, and the kinds of decisions the results are intended to support.

Modeling purposes can be characterized in two dimensions, utility and generality. Three categories of utility are distinguished here.

1. A model can provide an object for discussion as a concept is being defined and social constructs are developed. Conceptual definition models may not go through all the modeling stages, particularly verification and validation, because they serve as conceptual illustrations and support identification of variables that can strongly influence system behavior. Models developed for this first purpose have a very limited life, as they can be thrown away once they have served their purpose, or they may become a basis for more extensive modeling.
2. Models are also built to address a specific problem or question. For example, an organization may want to know how many servers should be specified for a system designed to handle an expected volume of transactions. A model of this kind may be produced for a specific system development project and its usefulness ends with the project, though it can be archived for a later time when a similar project might be undertaken.
3. Models may be developed and maintained for long-term usage. Flight simulators fall into this category. Also in this category are models that represent operations and provide ongoing decision support for operational planning.

In specifying the purpose of a model in the first two categories, it is helpful to pose the single most important question that the model will be used to address. It is also very helpful to establish the graphic that will be used to answer the question, the single most important chart that the customer wants to see. For example, it could be a cumulative distribution function for the time to complete a project, or a quantity distribution function for the number of widgets produced. Whatever it is, it needs to be specified early so that the simulation can be made to produce that chart.

The model may answer questions beyond the most important one originally posed for the model, but the one question helps to focus the model purpose and provides a criterion for model scope decisions. Though a model may have a single, well-defined purpose, it may well provide information on related issues. For example, a model constructed to answer a question about the duration of a process may also indicate what resources are constraining the process. Chapter 7 relates an example illustrating this point.

Modeling purposes in the third category lead to large endeavors that require experience gained from conceptual and limited use models. These types of modeling projects can address multiple purposes difficult to express in a single question. In these cases of larger utility, managing model purpose, scope, and level of abstraction require considerably more coordination and discipline.

Modeling purposes may also be distinguished by their degree of specificity or generality. A model may be built to represent a specific system or process, or a class of systems or processes. General models have much larger audiences than specific models. Very specific models require data from only one source, but general models require data from many sources. Verifying and validating specific models can also require less effort than general models. When specifying the purpose of a model, a modeler should keep in mind that more work comes with greater utility and with greater generality.

The primary modeling purpose question is:

- What question(s) should the modeling results answer?

Model scope may be defined by the modeling purpose, but it is clarified by the boundaries defined for the model. This means that the model must include elements sufficient to generate the outputs of interest. For example, if a customer is interested in resource utilization, then all the demands on the resources should be included. Modeler and stakeholders have to agree on what is included and excluded from the model. Maintaining model scope can be challenging. Stakeholders will have expectations that may go beyond the model scope, so the modeler must remind them of scope and capability. Model scope and boundaries can grow from one model version to the next, but scope should be maintained and clearly communicated for each version.

Modelers face a temptation against good definition. It is the desire to satisfy all stakeholders by providing a model that represents all their perspectives rather than a model that represents a common concern. For example, one stakeholder may want a model that reflects resource usage for a process in order to determine what the capacity of the process is, but another may want a model that includes all the personnel in their organization so that they can see where all their labor is spent, including that spent on the first stakeholder's process of interest. In that case, the modeler may need to use his negotiation skills to facilitate an agreement among the stakeholders for the sake of producing an adequate model within budget and schedule constraints.

The following are model scoping questions:

- Where should the model "start"? Where should it "end"?
- What system elements should be included?
- What resources are to be included?
- For each model component, will inclusion significantly affect the results? If so, will the influence on results be optimistic or pessimistic?
- For each model component, will inclusion strongly affect the model's credibility?
- For each model component, is data available or obtainable? How accurate is it?

Every model is an abstraction of reality and the modeler must choose an appropriate "level" of abstraction for a model. Choosing levels of abstraction can be challenging for a modeler, but it also allows the modeler to control how much detail goes into a model. A good guideline is to model at the level of abstraction necessary to answer the questions. Simple is typically better than complicated. When in doubt, by default start with as simple a model as possible and add detail later as necessary. One technique used successfully by expert modelers is to start with a simple prototype or proof-of-concept model early in the project to work out potential issues with the project, to improve the accuracy of the project time and resource estimates, and to obtain support from the stakeholders.

As an example of level of the abstraction, a model that simulates satellite communications can be very detailed if used to design communication scheduling algorithms, but can be very high level if used to confirm a decision on the number of satellites required in an orbit. The choice of abstraction level is also qualified by an M&S project budget and schedule. A modeler working within a small budget and tight schedule can choose to produce a smaller, more abstract model that requires less work to build and verify than a large, high-fidelity model. In fact, this approach is recommended: if the customer sees that results of a small model are useful, then addition of details for a better fidelity can be negotiated.

The following are model abstraction questions:

- Which of the following factors are influencing the modeling level of abstraction: modeling objective, model scope, performance measures of interest, alternatives to be examined, examination details for alternatives, data availability, execution time, modeling budget and schedule, animation, modeling toolset and language support, and modeler skillset? Is the degree of influence of each factor commensurate with the modeling purpose?
- How should each element be represented and to what level of detail?
- What simplifying assumptions can be made?
- Will any of the assumptions bias the modeling results either pessimistically or optimistically?
- What alternatives are to be explored?
- What scenarios are to be considered?

As a model is specified, important model parameters are discussed. These discussions typically start with output variables that address the modeling purpose and input variables that domain experts recognize as influential. With the discussion of input parameters, sources of quantitative data are discussed. Obtaining data for model parameters can be the most time-consuming part of a modeling project for several reasons. Data for model parameters may not exist and it will need to be elicited or collected. If data does exist, it may be proprietary and 1) upper management will need to be convinced that it should be made available for the modeling project, and 2) a process for acquiring the data will need to be executed. When data is provided, reviewing it and scrubbing it may require substantial effort.

Due to potentially long lead times in obtaining data, discussion of data sources should be undertaken during model specification so that data can be available once a model is constructed. The modeler requires data for model parameters but also for actual outcomes of the subject reality for model validation. Setting up a data collection system may not be feasible within the schedule and budget of the modeling project. If that is the case, then data is limited to whatever is available or whatever estimates can be elicited from domain experts.

The following are data acquisition questions:

- What sources of data are available for model inputs?
- If data is lacking for expected model inputs, can estimates be elicited from domain experts?

- What system performance data can be provided for validating the model?

When modeling a specific system or process, one might attempt to learn the real system or process through documents and telephone interviews. However, trying to model from a distance is difficult and inhibits fair representation of the reality. The best opportunity for good modeling occurs when making site visits, meeting the stakeholders in person, and walking through the subject of the model.

The foregoing discussion can be summarized in the following questions that should be answered when specifying a model:

- What is the question to be answered by a model study?
- What is the intended utility of the model: conceptual definition, solution generation, or long-term support?
- Is the model to represent a specific case or a class of cases?
- What kinds of decisions should be supported by the study?
- Who is the intended audience?
- What set of policies should be modeled?
- What is the model scope?
- What is the model level of abstraction?
- What model outputs are necessary?
- What kind of output data analysis will be necessary?

2.4 MODELING

2.4.1 MODELING PARADIGMS

One of the first decisions to be made in modeling is choosing the modeling paradigm. This may be dictated by the modeler's expertise in a particular paradigm or the paradigms supported by a simulation software package used in the modeler's organization. However, the different paradigms have advantages and limitations for various types of modeling. Continuous modeling, discrete event simulation, and agent-based simulation are discussed here.

Continuous simulation and discrete event simulation are distinguished by their representation of time. In continuous simulation, time is represented in regular intervals. Flows are represented as passing through a continuous model into and out of containers (also called stocks or levels). Flow rates can be increased and decreased based on the variable relationships. With each tick of the simulation clock in a continuous model, a fixed amount of simulated time passes and the values of all model variables, flow rates, and container contents are updated.

In a discrete event model, events are scheduled and the simulation clock moves to each event in the schedule (see the next event time advance approach in Section 3.2.1). Entities move through a model, obtaining resources, spending time in activities, and releasing their resources. Entities can have attributes so that each entity can have its own characteristics. Attribute values are used to set activity durations and to route entities through a model. Variable values are updated with each event.

Because they represent flows, continuous models tend to be more abstract than discrete event models. Where a continuous model treats work as a continuous flow, a discrete event model treats work as individual items passing through activities. Consequently, discrete event model diagrams can be easier to understand for audiences that think of entities being changed through activities performed on them. The ability to characterize entities in discrete event models also adds to representational capability.

Agent-based models represent interacting entities, or agents. Agents have their own characteristics and can initiate actions, communicate with one another, and react to one another. Like discrete event simulation, agent-based models generally treat time as a series of discrete events.

Table 2.1 summarizes characteristics of the three modeling paradigms in terms of system characteristics and modeling goals. Use continuous modeling, such as system dynamics modeling, if a global view is desired and aggregated entities are sufficient, whereby information on individual entities isn't necessary or possible to model. Discrete event modeling is useful for investigating system-level process behavior with visibility into individual entities. Agent-based modeling is also useful for disaggregated modeling whereby individual object behavior can be described and the modeler is interested in collective behavior that emerges from the interactions of the agents.

TABLE 2.1
Model Type Selection Criteria

Criteria	**Continuous**	**Discrete Event**	**Agent Based**
Perspective	Global view with feedback	System-level processes	Individual interacting objects
Entities	Aggregated entities	Disaggregated entities with attributes	Disaggregated agents with behaviors

Having acknowledged the distinctions for modeling paradigms, it is also important to know that current simulation programs are blurring these distinctions, particularly between continuous and discrete event simulation. The programs support both continuous and discrete event timing in the same model and is sometimes called hybrid modeling.

The choice of a modeling paradigm may depend on the simulation capabilities required as well as the modeler's expertise. These three modeling paradigms are discussed in more detail in Chapter 3.

2.4.2 MODEL CONSTRUCTION

Once a modeling paradigm and supporting software have been selected, decisions must be made about representing the real subject of the model. For continuous models, anything in the real system that changes is represented as a flow, so one must decide what flows to represent, how they are modified, and how they interact. For discrete event models, one must decide how to represent system elements using entities and resources, and how entities are routed. For agent-based models, agents are identified, their characteristics are specified and rules for their behavior are written.

In making decisions about model structure, multiple representational possibilities may be available. Consider, for example, modeling of a workflow process in a discrete event model. Entities may be used to represent work items and people are treated as resources, or entities may represent people who perform work and the work items are treated as resources. The choice of representation may depend on the modeling purpose. If the stakeholders are interested in changes in the people as they work, then the people can be characterized using attributes and attribute values that are updated to reflect changes in personal characteristics. On the other hand, if the stakeholders are interested in changes in work products and work capacity, then work items can be represented as entities with attributes, and people are the resources. When representational choices become apparent, simple trial models of each should be produced and evaluated.

Another structural decision in modeling is the division of the model into parts. Small models may not need partitioning, but larger models are managed best in parts. A number of factors may influence decisions about partitioning a model. The real system may logically be partitioned, suggesting the way in which a model of it should be partitioned. Also, the different flows or entity types or agents identified for a model may suggest partitions. Previous models of the subject reality may also suggest advantageous or disadvantageous ways to partition a model. In some cases, partitions are created to add new purpose and scope to an existing model.

As construction proceeds, a modeler should distinguish between design and implementation. For those who use graphical simulation software, the difference between design and implementation can be blurred because part of a model can be a working implementation while another part is being designed. A temptation faced by all modelers, especially novices, is to create an entire design of a large model in a graphical program with the expectation that the model runs as intended. However, the result can be an unwieldy model that is difficult to debug. When such a model has been created, it is recommended that it be treated as a design, set aside, and start a new model in which small sections can be implemented and tested, one section at a time. Of course, this iterative approach can and should be used from the outset, designing and implementing in pieces. An iterative approach with successive elaborations (“build a little, test a little”) is best.

As a model is constructed, it should have only enough detail to address its purpose(s) and no more. Modelers face several temptations that can lead to models that are larger than necessary:

- The desire to have the most comprehensive model in the field, one that represents all relevant phenomena known on the subject.
- The desire to satisfy all stakeholders, who may be asking questions such as "Does it include ...?" and "Can it show ...?"
- The desire to build and maintain only one model that incorporates all subsystems or incorporates all system variations.
- The desire to avoid representational abstractions about which one is unsure. It can seem easier to add many details and be sure that the representation is correct rather than try more abstract representations that require less data and testing.

As these temptations arise, it is best to face each one and acknowledge the trade-offs that come. As models grow, so does the amount of work in constructing, testing, fixing, and managing them.

Depending on how a model is to be used, performance requirements may be necessary. Iterative construction also facilitates assessment of model performance as the model is being built. For example, if many thousands of runs are needed to produce the necessary output data, then the simulation software must be able to record all the runs without using all available computer memory. The model must also be able to run fast enough to produce the required number of runs in a reasonable time period. Performance requirements and desires should be documented and performance should be assessed periodically during model construction to determine whether computing resources are sufficient.

Iterative construction also facilitates configuration management of a model. As each addition is made to a model, the change can be documented either in the model file or a separate document. The model file can be copied to a new file and a new edition can be started. This management of configurations is especially helpful when experimenting with model constructs and wanting to branch from a previous version, when multiple modelers are working on the same model, while debugging and looking for the source of a problem, or when stakeholders are requiring results and a stable version is needed for producing them.

Modeling decisions can reflect trade-offs between multiplicity and behavioral complexity. For example, suppose a factory has multiple product assembly lines that have much in common but differ from one another in a few processing steps. A graphical model of these lines might depict each line separately or it might abstract from the separate lines by combining them into a single line with processing exceptions to represent the differences. Combining the lines produces a simpler static view of the model. However, all the items passing through one processing line can make tracing more difficult than tracing fewer entities through individual lines. In the end, one may decide to use a graphical depiction of multiple lines because the diagram is easier to use when discussing the model with stakeholders familiar with the factory layout.

2.4.3 DATA COLLECTION AND INPUT ANALYSIS

Parameter creation is part of model construction. Parameters for inputs and outputs should be readily identifiable to system managers and participants. Input parameters should be created with the expectation that data can be collected for them and output parameters should be created with the expectation that data from system outcomes is available for validation.

Data must be relevant and of good quality. Relevant data comes from measurements that have the same meaning and units as the model parameters for which the data will be used. Good quality implies that data is complete and does not contain anomalies, such as inaccurately recorded values. Obtaining good-quality, relevant data for input parameters and for validation can be challenging. Although some types of simulation models are built to use qualitative data, simulation models ordinarily use quantitative data.

Quantitative data can be deterministic or stochastic. Deterministic data can be either constant values or variable values computed as a function of time or other independent variables. Stochastic data reflects variation that cannot be characterized as a deterministic function. The uncertainty in values can be characterized in a random distribution. When recorded data is provided for input parameters, it can be analyzed to determine whether it should be used in a model as constant, functionally variable, or randomly variable. However, when recorded data is not available for input parameters, the modeler faces a question as to whether data should be collected or elicited.

If data collection is undertaken, the costs of developing a measurement system and collecting data are incurred. These include specifying the data to be collected, specifying the collection methods and instruments, providing for data storage, training data recorders, and reviewing the collected data for completeness and anomalies. The time and expense required for data collection may not be feasible for a modeling project, especially for initial models of a system. In these cases, elicitation of parameter values from domain experts should be considered.

Occasionally domain experts can offer constants or functions for parameter values, but usually recalling values from experience will produce estimates with variation. When consulting experts in a system or process, they can be asked for a most likely value of a parameter, a minimum value, and a maximum value. The definition of minimum and maximum can lead to debates as to whether these mean extreme values (the lowest and highest values either seen or possible) or "90% values." One way to facilitate this discussion is to ask for low and high values as usually occurring and then ask whether extremes beyond the usual lows and highs can appear. The following series of questions is helpful for eliciting data for each parameter.

- What is the lowest value of the parameter most of the time?
- What is the highest value of the parameter most of the time?
- Is a value between these two more likely than either one? If so, what is it?
- If the parameter has a most likely value, can values occur that are less than the lowest value or more than the highest value?

Once data is obtained for model inputs, it must be analyzed to produce input values. Input analysis can involve a number of statistical tools, and it is the subject of Chapter 5. It will elaborate more on input analysis methods, including using the answers to the preceding questions. It is sufficient to say that input analysis is another step involved in model iteration. Much effort can be spent choosing input values including input distributions. The best approach in many cases is to apply just enough effort to obtain input values that are good enough to test a model with runs for a sensitivity analysis. The sensitivity analysis varies the input values systematically so that the relative influence of inputs can be estimated. With these estimates, a modeler can decide which inputs deserve further effort for refinement.

2.5 MODEL ASSESSMENT

When a model has been produced, stakeholders call upon the modeler(s) to explain the model's credibility. The modeler must establish his/her own confidence in the model and then convey that confidence to stakeholders and peers, usually through sharing results of Verification and Validation (V&V) exercises. Verification exercises determine whether or not a model is built correctly (error-free) and represents the intended behavior according to the model specification. Validation exercises determine whether the model provides an adequate representation of the real system for the model's stated purpose and addresses the sponsor's problem. The importance of model assessment cannot be understated: stakeholders must have confidence in a model in order to use its results well.

Most textbooks on M&S cover model assessment. Richardson and Pugh, in their text on system dynamics modeling [25], present a model assessment scheme summarized in Table 2.2 that distinguishes exercises for structural assessment and behavioral assessment. They also add model evaluation exercises to the V&V exercises in their scheme outlined below. This assessment scheme is also covered in more detail in modern treatments of system dynamics including [29] and [18].

The details of these assessments are listed below:

- Verification of structure
  - Equation review is a modeler check of every equation for correctness. The parameter dimensions of each equation are analyzed for potential errors. Each equation is checked for the effects of extreme values, including division by zero.
  - Structural adequacy review is a modeler review to ensure that the elements included in the model and the level of abstraction are sufficient to address the model's stated purpose and specification.
- Verification of behavior
  - Traces of specific entities are essential to verifying a model's logic and the correctness of its implementation. Tracing should cover each path and test execution of each condition.

TABLE 2.2
A Model Assessment Scheme (adapted from Richardson and Pugh [25])

	Structure	**Behavior**
Verification	Equation review Structural adequacy review	Parameter variability testing Structural insensitivity review Traces
Validation	Face validity review Parameter validity review	Output comparisons Outputs discrimination check Case replication Case prediction
Evaluation	Model appropriateness review	Sensitivity analysis Unexpected behavior review Anomalous behavior review Extreme inputs review Process insights review System improvement test

    - Parameter variability testing is testing that model outputs reflect changes to model inputs in the same way that the real system outputs reflect changes to real system inputs.
    - Structural insensitivity review is an expert review of model outputs and behavior against its structure to ensure that the level of abstraction does not inhibit the modeling purpose.
- Validation of structure
    - Face validity is assessed by expert review of the model structure to confirm that, for the modeling purpose, it is an adequate representation of the real system.
    - Parameter validity is assessed by expert review of parameters to ensure that they are recognizable and not contrived, and that parameter values represent the best available information about the real system.
- Validation of behavior
    - Comparisons of model outputs and system outputs are made statistically. System output values should fall within confidence intervals on the mean model outputs.
    - Experts comparing system outputs and model outputs should not be able to discriminate between the two.
    - A case replication demonstrates the model's ability to imitate real system results.
    - A case prediction demonstrates the model's ability to forecast system behavior accurately. This may be a statistical test in which a prediction

interval is constructed and system output is tracked to see whether it falls within the interval.

- Evaluation of structure
  - Members of the intended audience review the model for appropriate levels of abstraction/detail and simplicity/complexity.
- Evaluation of behavior
  - Sensitivity analyses can be run using designed experiments to identify the relative influence of input variables on each outcome, as well as interactions between inputs. The results should be useful in explaining some phenomena in the real system.
  - A modeler reviews any unexpected behavior found during model usage and performs traces to explain the behavior. If possible, the behavior should be observed in the real system.
  - If the model exhibits anomalous behavior, it should be reviewed against the real system to ensure that it does not conflict with system behavior.
  - When extreme values are input to the model, model behavior should imitate system behavior when the system is subjected to the same inputs.
  - Review model usage and record any insights gained about system behavior.
  - A system improvement test considers whether the model identifies any system improvements.

Most, but not all, of the foregoing exercises are necessary for building confidence in a model. The verification exercises are essential for the modeler's confidence that a model works correctly. The tracing is especially important so that a modeler can explain the inner workings when called upon to do so. Fortunately, verification exercises can be run iteratively during model construction so that they are not all left until afterward (when stakeholders are calling for modeling results they can use).

Validation and evaluation of structure starts with the first time a modeler presents a draft model to stakeholders, explains it to them, and elicits their feedback. This feedback usually helps correct and refine model structure as well as facilitate further data collection. In such a meeting, the model specification can be refined, data collection can continue, and model structure can be validated.

Validation and evaluation of behavior exercises provide a number of opportunities for demonstrating what a model can do. The exercise that stakeholders request most often is replication of a case of actual results. Therefore, early in the modeling process, a modeler should be looking for a case of recent system behavior that the model can be calibrated to imitate. In the course of such a calibration, if the modeler cannot reproduce one or two primary system outcomes, then the modeler must reexamine data and assumptions, sometimes returning to system experts. For example, it can happen that elicited values for a parameter were inaccurate and that trying new values allows the model to reproduce system outputs. If the experts agree that the initial values may have been erroneous and that new values are better, then the modeler has leeway for calibration.

The foregoing example also illustrates occasions in which a model is like a puzzle: the modeler obtains all the pieces and puts them together. If they do not fit (i.e., outcomes do not imitate real system behavior), then the modeler must figure out which piece(s) do not fit well and scrutinize them with experts. They may find that they tried to insert the wrong piece into the puzzle.

2.5.1 EXPERIMENTATION AND OUTPUT ANALYSIS

As a model is assessed, experimentation and output analysis can begin in earnest. Good evaluation exercises usually signal the start of experimentation and provide the model with insights for further experimentation. Unfortunately, too many models are not used often enough for experimenting, sometimes because the modeler lacks the imagination to create experiments or statistical knowledge and skills to conduct experiments. Modelers are encouraged to put as much effort into using a model as they put into making it.

One school of thought on simulation is that M&S is a learning tool for investigating how systems work. They measure success from what one can learn through M&S. Another school of thought views M&S primarily as a statistical tool that produces quantitative data to support decision making. The two schools of thought are not mutually exclusive, as indicated in the discussion on model specification. Models can have a range of utility and a particular model can evolve from a simple learning model to a decision support model.

Like input analysis, output analysis usually involves statistical analysis of output data and presentation of the data in formats accessible by the stakeholder audience. Analyzed outputs should be used to tell a story and provide information that addresses the problems about which stakeholders are thinking.

2.6 MAKING RECOMMENDATIONS AND REPORTING

As a modeler exercises a model, generates output data, and analyzes it, model behaviors are revealed and he/she comes to understand how a model works. This information provides a basis for discussions with system participants who can apply their knowledge of system workings. System participants often know about the behavior a model describes but the modeler can clarify and quantify the behavior. When they find behavior that is counterintuitive or issues that are unexplained, the modeler is provided opportunities for investigation by running model scenarios.

Recommendations can also emerge from these conversations. Participants often have ideas that they think should be implemented for improving the way a system or process works, and the modeling results may indicate which ideas are most useful. In exploring alternative scenarios, the modeler may also see improvement opportunities that participants had not seen.

Insights and recommendations should not come as surprises to stakeholders. Those surprised by recommendations may be reluctant to accept them. Ongoing discussions should lead up to the final report and briefing with its recommendations.

Good documentation of an M&S study is important for facilitating understanding of the results, especially among those not directly involved in the study, and promoting their implementation. Though a model may not be used by stakeholders, making it accessible confirms transparency of the study. an M&S study report should include a clear description of the problem and the appropriate view of the system, the objectives of the study, modeling specification and assumptions, the modeling process, an overview of the model, the model assessment activities and results, and conclusions with recommendations. The following outline may be used as a checklist for an M&S report.

1. Introduction
   - Problem statement and description
     - Extent and history of the problem, causes of the problem, possible solutions and their obstacles
   - Purpose of study
   - Purpose of model building
   - Executive summary of key results
2. Background
   - System description
     - Diagrams that illustrate the system configuration
   - System reference behavior
     - Narrative description, tabular data, graphs
   - Assumptions and the underlying rationale for each
3. Model Development
   - Modeling process
     - Modeling approach and sequence of events
   - Model evolution
   - Data acquisition
     - Source of data, method of collection, problems, solutions, analysis, etc.
4. Model Description
   - Time frame, spatial boundaries, entities, attributes, key assumptions
   - Process flow and model structure
     - Flowcharts, model diagrams, etc.
   - Key logic and equations, sources of uncertainty, probability distributions, etc.
   - Model Overview
     - Sources of data and input analysis
     - Flowcharts, flow networks, block diagrams, etc.
   - Assumptions and other model details
   - Approach for verification and validation
5. Model Verification and Validation
   - Testing results, extreme values, sensitivity analysis
   - Comparison to reference behavior, expert review, etc.
   - Statistical analyses

    - Other testing and V&V methods
6. Model Application and Transition
    - Experimentation and analysis of results
    - Tabulated data, plots, bar graphs, histograms, pie charts, statistical analyses and other reports, etc.
    - Interpretation of model results
    - Limitations of the model and future enhancements
    - Next steps
    - Model transfer issues
        - Availability of model user documentation and of model developer documentation
7. Conclusions and Recommendations
    - Real-world system
        - Policy suggestions
    - Future model usage and development
    - Modeling process notes
    - Modeling tool(s) and platform used
    - Process improvement scenarios
    - Future research
8. Appendices (if necessary)
    - Supplementary model details
    - Model run output
    - Selected computer outputs, related letters, technical articles, etc.
    - Additional data analysis as needed

2.7 KNOWLEDGE AND SKILLS FOR MODELING AND SIMULATION

The discussion in this chapter should have made clear that M&S requires knowledge and skills, so this last section discusses these areas explicitly. A modeler must possess knowledge and skills in four areas. First, one must understand modeling and simulation concepts, processes, paradigms, and tools as described in this chapter. M&S tools are numerous and are evolving, and full descriptions are beyond the scope of this book. Reviews and surveys of simulation software are readily available online. An example survey of tools kept updated is at `http://www.orms-today.org/surveys/Simulation/Simulation.html`.

The second area in which a modeler requires knowledge is that of the domain being modeled. The domain might be distinguished in two degrees of relevance: specific systems and fields. A modeler who is familiar with specific systems may also be a domain expert capable of modeling with minimal reliance on other system experts. A modeler who is expert in a field, for example product development, and is asked to model a specific development program must rely heavily on experts in the development program to acquaint him/her with its specific structure and data. In this case, a modeler brings familiarity with the structure and dynamics of product development as well as expertise in representing them. Sometimes a modeler is requested

for his/her M&S skills despite a lack of familiarity with a field. In this case, he/she may learn from published modeling work in the field, but is still more reliant on domain experts to learn about the field and the subject system or process in sufficient depth for modeling.

Engagement skills are a third area in which a modeler should be well-versed. This area includes human relations, organizational, and project management skills that enable a modeler to meet with potential customers; assess their needs and relate them to possible M&S solutions; develop reasonable expectations for an engagement; negotiate a program of study; work with participants to specify a model, collect data, and validate the model; keep sponsors informed; and report useful results.

The fourth M&S knowledge and skill area for modelers is statistics. Models are essentially quantitative tools executed on computers and the success of most M&S studies depends on quantitative analysis. Input values are usually the result of statistical analysis, such as distribution fitting. Model experimentation and sensitivity analysis often require designed experiments that are analyzed with analysis of variance (ANOVA). Outputs may also need to be analyzed with hypothesis testing, regression analysis, confidence intervals, and prediction intervals. The modeler uses statistics to understand system behavior as represented in a model and relate a story of that behavior to stakeholders. See statistics details in Chapters 4, 5, and 6.

In summary, keys to a successful M&S project include a well-defined and achievable goal, complementary skills on the project team, an adequate level of user participation, selection of an appropriate simulation toolset or language, and effective project management.

2.8 SUMMARY

Modeling and simulation (M&S) projects vary widely, so modeling processes and methodologies can be employed very differently across projects. The nominal modeling stages are engagement, specification, construction, verification and validation, experimentation, and reporting. But the modeling process does not proceed through these stages in a strictly sequential manner because a modeler undertakes combinations of these stages iteratively. A modeler needs specialized knowledge and skills to conduct a modeling process well.

First, one should identify the stakeholders including sponsors, process or system owners, and the process or system participants who do the actual work. Stakeholders may also include other engineers and scientists in related disciplines using the simulation results. The modeler must always keep stakeholders informed of the study and its progress.

Model specification begins with identifying a problem and describing its context for a modeling purpose. The purpose will include the intended audience for the modeling results, the desired set of policies to be simulated, and the kinds of decisions the results are intended to support. The modeler must choose an appropriate level of model abstraction necessary to answer the questions. By default start with a simple model and add detail as necessary.

One of the first decisions in modeling is choosing the modeling paradigm. The different paradigms of continuous modeling, discrete event simulation, and agent-based simulation have respective advantages and limitations for various types of modeling.

Continuous models are usually more abstract than discrete event models. Use continuous modeling if a global view is desired and aggregated entities are sufficient. Discrete event models can be easier to understand for audiences that think of entities being changed through activities performed on them, and useful for investigating system-level process behavior with visibility into individual entities. Agent-based modeling is also useful for disaggregated modeling whereby individual object behavior can be described and if one is interested in collective behavior that emerges from their interactions. The choice of modeling paradigm may also depend on required simulation capabilities and the modeler's expertise.

During model construction, decisions must be made about representing the real-world subject. For continuous models, one must decide what flows to represent, how they are modified, and how they interact. For discrete event models, one must decide how to represent system elements using entities and resources, and how entities are routed. For agent-based models, agents are identified, their characteristics are specified, and rules for their behavior are written.

Another structural decision in model construction is the partitioning of a model. An iterative approach with successive elaborations ("build a little, test a little") is best to support these modeling decisions. Iterative construction also facilitates assessment of model performance as it is being built, and configuration management of the ongoing model changes.

For data collection and input analysis, the input parameters should be created such that the data can be collected and outputs defined that will be available for validation. The costs of data collection should be considered, and elicitation of parameter values from domain experts may be warranted.

Model assessment addresses 1) verification to determine whether or not a model is built correctly without errors and represents the intended behavior according to the model specification, and 2) validation to determine whether the model provides an adequate representation of the real system for the model's stated purpose and addresses the sponsor's problem. V&V covers both structural and behavioral aspects.

Experimentation and output analysis can begin when a model is being assessed. Like input analysis, output analysis involves statistical analysis of output data and presentation of it to the stakeholder audience such that it addresses their problems.

Good documentation of an M&S study is important for facilitating understanding of the results, especially among those not directly involved in the study, and promoting their implementation.

A modeler must possess the right knowledge and skills to conduct a good simulation study. One must understand M&S concepts, processes, paradigms, and tools; have knowledge of the application domain; engagement skills to work with people; and statistics skills for the quantitative analysis.

The overall keys to a successful M&S project include a well-defined and achievable goal, complementary skills on the project team, an adequate level of user par-

ticipation, selection of an appropriate simulation toolset or language, and effective project management.

3 Types of Simulation

This chapter presents an overview of the major types of simulation models with simple illustrations. The systems and/or their models can be discrete, continuous, or a combination. The essential difference in modeling is how the simulation time is advanced. Continuous systems modeling methods such as system dynamics always advance time with a constant delta (dt), a fixed time slice. Since variable values may change within any time interval in a continuous system, the delta increment is very small and time-dependent variables are recomputed at the end of each time increment. The variable values change continuously with respect to time. In contrast, in discrete modeling, the changing of variable values is normally event-based. State changes occur in discrete systems at aperiodic times depending on the event happenings. The simulation time is advanced from one event to the next in a discrete manner.

All classes of systems may be represented by any of the model types. A discrete model is not always used to represent a discrete system and vice versa. The choice of model depends on the specific objectives of a study.

Simulation software may be oriented for specific model types or allow hybrid approaches. In general, the software tools must implement the following functions:

- Initialization of system variables
- Time clock for flow control
- For continuous systems:
  - Time advancement in equal small increments, with all variables recomputed at each time step
- For discrete systems:
  - Event calendar for timing
  - Time advancement from one event to the next, with no changes in variables between events
  - Event processing routines
- Statistics collection
- Output generation

Additional features beyond the above may include a graphical user interface, animation, a variety of probability distributions for random variates, design of experiments, optimization, interfaces with other software or databases, and more. See Appendix A for some available simulation tools to support continuous, discrete event, agent-based, and hybrid modeling applications.

Models may be deterministic (having no probabilistic variables) or stochastic (including probabilistic variables). Few engineering applications are wholly deterministic. When a model has probabilistic variables, each run can have a different outcome constituting an estimate of the model characteristics. Therefore, many runs

must be made to characterize output variables. See Chapter 4 on handling randomness in models, Chapter 5 for stochastic inputs, and Chapter 6 for output analysis of stochastic systems to address uncertainty.

3.1 CONTINUOUS

Continuous systems models consist of differential or algebraic equations to solve variables representing the state of the system over time. A continuous model shows how values change as functions of time computed at equidistant small time steps. These changes are typically represented by smooth continuous curves. Continuous simulation models may also be applied to systems that are discrete in real life but where reasonably accurate solutions can be obtained by averaging values of the model variables.

Continuous models perform a numerical integration of differential equations over time. A practical and simple solution is to use Euler's method. It approximates the derivative of the function $y(t)$ by using the forward difference method. It computes the following at each time increment:

$$y(t_{n+1}) = y(t_n) + \frac{dy}{dt}(t_n)\Delta t \tag{3.1}$$

where

y is a function over time
$\frac{dy}{dt}$ is the time derivative of y
n is the time index
Δt is the time step size to increment t_n to t_{n+1}

The Runge-Kutta method uses additional gradients within the time divisions for more accuracy. See Appendix A for simple programs for continuous systems demonstrating Euler's method and the Runge-Kutta method. These programs use integration methods for computing variable values at each time step within a time loop.

System dynamics is the most widely used form of continuous simulation. System dynamics refers to the simulation methodology pioneered by Jay Forrester, which was developed to model complex continuous systems for improving management policies and organizational structures [5] [6]. Improvement comes from model-based understandings.

System dynamics provides a very rich modeling environment. It can incorporate many formulations including equations, graphs, tabular data, or otherwise. Models are formulated using continuous quantities interconnected in loops of information feedback and circular causality. The quantities are expressed as levels (also stocks or accumulations), rates (also called flows), and information links representing the feedback loops.

Levels represent real-world accumulations and serve as the state variables describing a system at any point in time (e.g. the number of cars on a road, number of manufacturing defects, height of water in a dam, work completed, etc.). Rates are

the flows over time that affect the levels. See Table 3.1 for a description of model elements.

The system dynamics approach involves the following concepts [24]:

- Defining problems dynamically, in terms of graphs over time
- Striving for an endogenous (caused within) behavioral view of the significant dynamics of a system
- Thinking of all real systems concepts as continuous quantities interconnected in information feedback loops and circular causality
- Identifying independent levels in the system and their inflow and outflow rates
- Formulating a model capable of reproducing the dynamic problem of concern by itself
- Deriving understandings and applicable policy insights from the resulting model
- Ultimately implementing changes resulting from model-based understandings and insights, which was Forrester's overall goal

A major principle of system dynamics modeling is that the dynamic behavior of a system is largely a consequence of its structure. Thus, system behavior can be changed not only by changing input values but by changing the structure of a system. Improvement of a process thus entails an understanding and modification of its structure. The structures of the as-is and to-be processes are represented in models.

The existence of process feedback is another underlying principle. Elements of a system dynamics model can interact through feedback loops, where a change in one variable affects other variables over time, which in turn affects the original variable. Understanding and taking advantage of feedback effects in real systems can provide high leverage.

3.1.1 CONSERVED FLOWS AND THE CONTINUOUS VIEW

In the system dynamics worldview, individual entities are not represented as such. Instead they are abstracted into aggregated flows. The units of flow rate are the number of entities flowing per unit of time. These flows are considered material or physical flows that must be conserved within a flow chain.

Information links connect data from auxiliaries or levels to rates or other auxiliaries. Information connections are not conserved flows because nothing is lost or gained in the data transfer.

A physical example of a real-world level/rate system is a water network, such as a set of holding tanks connected with valved pipes. It's easy to visualize the rise and fall of water levels in the tanks as inflow and outflow rates are varied. The amount of water in the system is conserved within all the reservoirs.

The continuous view does not track individual events, rather flowing entities are treated in the aggregate and systems can be described through differential equations.

There is a sort of blurring effect on discrete events. The focus of continuous models is not on specific individuals or events like discrete event approaches (described in Section 3.2), but instead on the patterns of behavior and on modeling average individuals in a population.

3.1.2 MODEL ELEMENTS

System dynamics model elements are summarized in Table 3.1. Their standard graphical notations are shown in the next example model. The same notations with similar icons are used consistently in the modeling tools for system dynamics listed in Appendix A.

TABLE 3.1: System Dynamics Model Elements

Element	**Description**
Level	A level is an accumulation over time, also called a stock or state variable. It can serve as a storage device for material, energy, or information. Contents move through levels via inflow and outflow rates. Levels represent the state variables in a system and are a function of past accumulation of rates.
Source/Sink	Sources and sinks indicate that flows come from or go to somewhere external to the system or process. Their presence signifies that real-world accumulations occur outside the boundary of the modeled system. They represent infinite supplies or repositories that are not specified in the model.
Rate	Rates are also called flows; the actions in a system. They effect the changes in levels. Rates may represent decisions or policy statements. Rates are computed as a function of levels, constants, and auxiliaries.
Auxiliary	Auxiliaries are converters of input to output, and help elaborate the detail of stock and flow structures. An auxiliary variable must lie in an information link that connects a level to a rate. Auxiliaries often represent score-keeping variables.

(*continued*)

TABLE 3.1: System Dynamics Model Elements (*continued*)

Element	Description
Information Link	Information linkages are used to represent information flow (as opposed to material flow). Rates, as control mechanisms, often require connectors from other variables (usually levels or auxiliaries) for decision making. Links can represent closed-path feedback loops between elements.

Elements can be combined together to form larger infrastructures with associated behaviors. Using common existing infrastructures in models can save a lot of time and headache when reusing them. An infrastructure can be easily modified or enhanced for different modeling purposes. See [18] for a taxonomy of infrastructures with examples that apply to engineering contexts.

3.1.3 MATHEMATICAL FORMULATION OF SYSTEM DYNAMICS

This section explains the mathematics of system dynamics modeling for general background, and is not necessary to develop or use system dynamics models. It illustrates the underpinnings of the modeling approach and provides a mathematical framework for it. Knowing it can be helpful in constructing models. An elegant aspect of system dynamics tools is that systems can be described visually to a large degree, and there is no need for the user to explicitly write or compute differential equations. The tools do all numerical integration calculations. Users do, however, need to compose equations for rates and auxiliaries, which can sometimes be described through visual graph relationships describing two-dimensional (x-y) plots.

The mathematical structure of a system dynamics simulation model is a set of coupled, nonlinear, first-order differential equations per Equation 3.2.

$$x'(t) = f(x, p) \tag{3.2}$$

where

x is a vector of levels
p is a set of parameters
f is a nonlinear vector-valued function

State variables are represented by the levels. As simulation time advances, all rates are evaluated and integrated to compute the current levels. Runge-Kutta or Euler's numerical integration methods are normally used. These algorithms are described in standard references on numerical analysis methods or provided in technical documentation with system dynamics modeling tools. Also see the software examples in Appendix B.

Numerical integration in system dynamics determines levels at any time t based on their inflow and outflow rates per Equation 3.3, where the dt parameter is the chosen time increment.

$$Level(time) = Level(time - dt) + (inflow - outflow) * dt \tag{3.3}$$

Describing the system with high-level equations spares the modeler from integration mechanics. Note that almost all tools also relieve the modeler of constructing the equations; rather, a diagrammatic representation is drawn and the underlying differential equations are automatically produced.

Example: System Dynamics Model for Resource Allocation

Engineering management needs to allocate personnel resources based on relative needs among competing tasks. The model in Figure 3.1 contains resource allocation infrastructures to support this strategic decision making. It is implemented with the iThink tool (see Appendix A). The resource allocation model has levels, shown as boxes, for the tasks (in work units) and task resources (number of people). The two primary flows represent two streams of work: each set of work enters a backlog, is performed at a rate illustrated by the valves, and then enters a completed state. Auxiliary variables are designated as circles (e.g. *Task 1 Productivity*), sources and sinks are the clouds, and information links are the single arrows connecting elements.

In each task flow chain, the rate of work accomplished is represented as a generic production process shown in Figure 3.2. This is an example of a well-established infrastructure. The flow rate drains the task backlog level and accumulates into completed work. The production rate of work completion is computed by multiplying the resources (# of people) by their average productivity (tasks per person per time unit).

Tasks with the greatest backlog of work receive proportionally greater resources ("the squeaky wheel gets the grease"). The allocation infrastructure will adjust dynamically as work gets accomplished, backlogs change, and productivity varies. The dynamic behavior is driven by the equations for the continuously adjusted proportion of resources allocated for a task per Figure 3.3. Resources are allocated dynamically based on each workflow's backlog. More resources are allocated to the workflow of tasks having the larger backlog. As work is accomplished, backlogs change and productivity varies.

Productivity serves as a weighting factor to account for differences in the work rates in calculating the backlog effort. Auxiliaries are used to help compute the estimated effort for each task by adjusting for productivity. The full model equations are shown in Figure 3.4. They include the standard graphic icons for system dynamic elements of rates, levels, and auxiliaries. Graphic curves show defined time-varying input functions (also listed with their table values).

This type of resource policy is used frequently when creating project teams, allocating staff, and planning their workloads. With a fixed staff size, proportionally more people are dedicated to the larger tasks. For example, if one task is about twice as large as another, then that team will have roughly double the people. The work

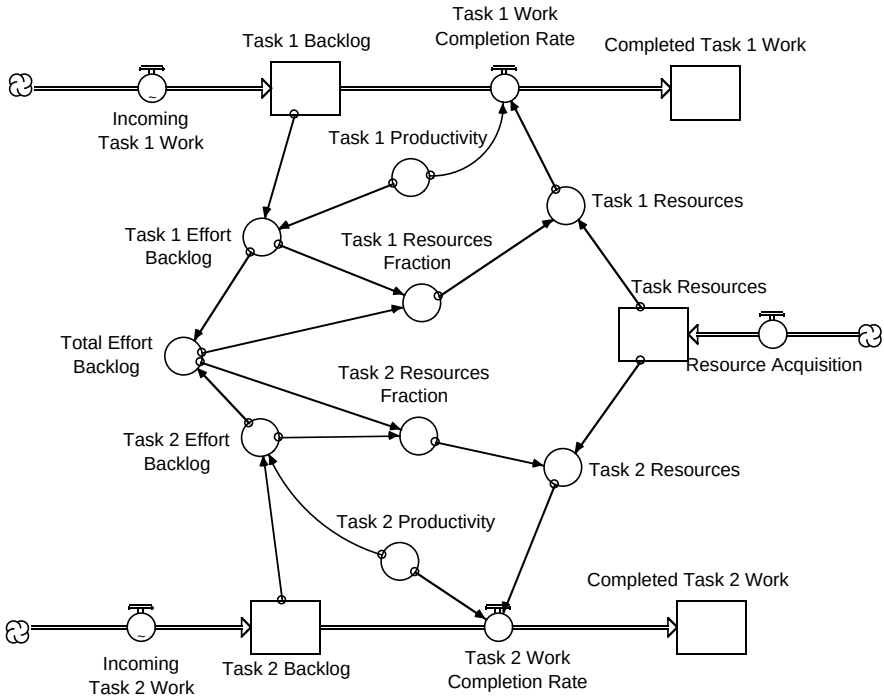

FIGURE 3.1: Resource Allocation Model

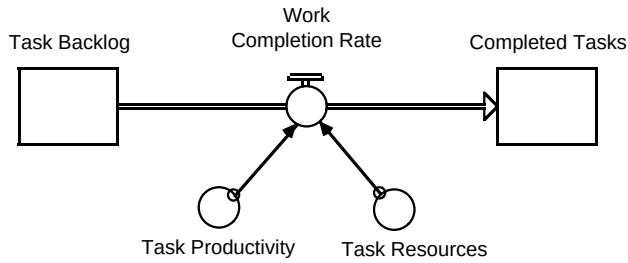

FIGURE 3.2: Task Production Structure and Rate Equation

backlogs are monitored throughout a project, and people re-allocated as the work ratios change based on the respective estimates to complete. This example models the allocation between two tasks, but it can be easily extended to cover more competing tasks. Many more variations of the resource allocation infrastructure are also possible.

Figure 3.5 shows the results of a simulation run where both tasks start with equal

```
Task_1_Effort__Backlog = Task_1_Backlog/Task_1_Productivity
Task_2_Effort__Backlog = Task_2_Backlog/Task_2_Productivity
Total_Effort__Backlog = Task_1_Effort__Backlog+Task_2_Effort__Backlog
Task_1_Resources__Fraction = Task_1_Effort__Backlog/Total_Effort__Backlog
```

FIGURE 3.3: Task Resource Allocation Equations

```
Completed_Task_1_Work(t) = Completed_Task_1_Work(t - dt) + (Task_1_Work__Completion_Rate) * dt
INIT Completed_Task_1_Work = 0
    INFLOWS:
        Task_1_Work__Completion_Rate = Task_1_Resources*Task_1_Productivity
Completed_Task_2_Work(t) = Completed_Task_2_Work(t - dt) + (Task_2_Work__Completion_Rate) * dt
INIT Completed_Task_2_Work = 0
    INFLOWS:
        Task_2_Work__Completion_Rate = Task_2_Resources*Task_2_Productivity
Task_1_Backlog(t) = Task_1_Backlog(t - dt) + (Incoming__Task_1_Work - Task_1_Work__Completion_Rate)
* dt
INIT Task_1_Backlog = 10
    INFLOWS:
        Incoming__Task_1_Work = GRAPH(time)
        (1.00, 1.00), (2.00, 1.00), (3.00, 1.00), (4.00, 1.00), (5.00, 1.00), (6.00, 1.00), (7.00, 1.00), (8.00, 1.00),
        (9.00, 1.00), (10.0, 1.00), (11.0, 1.00), (12.0, 1.00), (13.0, 1.00)
    OUTFLOWS:
        Task_1_Work__Completion_Rate = Task_1_Resources*Task_1_Productivity
Task_2_Backlog(t) = Task_2_Backlog(t - dt) + (Incoming__Task_2_Work - Task_2_Work__Completion_Rate)
* dt
INIT Task_2_Backlog = 10
    INFLOWS:
        Incoming__Task_2_Work = GRAPH(time)
        (1.00, 1.00), (2.00, 1.00), (3.00, 1.00), (4.00, 1.00), (5.00, 1.00), (6.00, 2.02), (7.00, 1.98), (8.00, 2.00),
        (9.00, 1.00), (10.0, 1.00), (11.0, 1.00), (12.0, 1.00), (13.0, 1.00)
    OUTFLOWS:
        Task_2_Work__Completion_Rate = Task_2_Resources*Task_2_Productivity
Task_Resources(t) = Task_Resources(t - dt) + (Resource_Acquisition) * dt
INIT Task_Resources = 4
    INFLOWS:
        Resource_Acquisition = 0
Task_1_Effort__Backlog = Task_1_Backlog/Task_1_Productivity
Task_1_Productivity = .5
Task_1_Resources = Task_Resources*Task_1_Resources__Fraction
Task_1_Resources__Fraction = Task_1_Effort__Backlog/Total_Effort__Backlog
Task_2_Effort__Backlog = Task_2_Backlog/Task_2_Productivity
Task_2_Productivity = .5
Task_2_Resources = Task_Resources*Task_2_Resources__Fraction
Task_2_Resources__Fraction = Task_2_Effort__Backlog/Total_Effort__Backlog
Total_Effort__Backlog = Task_1_Effort__Backlog+Task_2_Effort__Backlog
```

FIGURE 3.4: Resource Allocation Model

backlogs and 50% allocations due to equal incoming work rates. At $time = 6$ more incoming work starts streaming into the backlog for Task 2 as seen on the graph. The fractional allocations change, and more resources are then allocated to work off the excess backlog in Task 2. The graph for *Task 2 Resources Fraction* increases at that point and slowly tapers down again after some of the Task 2 work is completed.

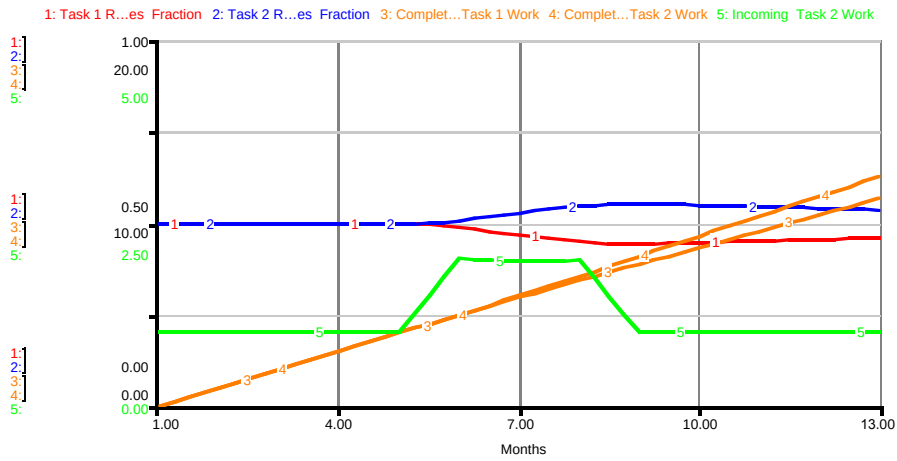

FIGURE 3.5: Resource Allocation Model Output

3.2 DISCRETE EVENT

Discrete event models consist of flows of individual entities, characterized by the values of their respective attributes. The characteristics of the entities can change during a simulation run. These changes occur instantaneously as the simulated time lapses. The entities move through a system represented as a network of nodes, perform activities by using resources, and create events that change the state of a system. Standard discrete event model elements are summarized in Table 3.2. They are described in more detail in subsequent sections.

TABLE 3.2: Discrete Event Model Elements

Element	**Description**
Create node	Generator of entities to enter a system.
Terminate node	Departure point of entities leaving the system.
Activity node	Locations where entities are served and consume resources.
Entity	An object in a system whose motion may result in an event.
Resource	Commodities used by entities as they traverse in a system.

(*continued*)

TABLE 3.2: Discrete Event Model Elements (*continued*)

Element	Description
Path	Routes that entities travel between nodes.
Batch/Unbatch Nodes	Where entities are combined into groups or uncombined.
Information link	Data connections between model elements for logical and mathematical operations.

3.2.1 NEXT EVENT TIME ADVANCE APPROACH

This section describes the discrete event-based approach for a queue/server system and generalizes the introductory example calculations in Section 1.6.1. It illustrates the logic used for time stepping through an event-based simulation, updating of statistics, etc. The next-event time-advance approach for a single-server queueing system will be described using the following notation and illustrated in Figure 3.6. It is the standard procedure used in discrete event modeling. This description (derived from [17]) will step through the computation sequence of a simulation including the generation of random numbers as normally performed by a simulation tool. The nomenclature used is:

t_i = time of arrival of the ith customer
$A_i = t_i - t_{i-1}$ = interarrival time between $(i-1)$st and ith arrivals of customers
S_i = time that server spends serving ith customer
D_i = delay in queue of ith customer
$c_i = t_i + D_i + S_i$ = time that ith customer completes service and departs
e_i = time of occurrence of ith event

The variables are random numbers and we assume that the probability distributions of the interarrival times A_1, A_2,... and the service times S_1, S_2,... are known. Their cumulative distribution functions can be used to generate the random variates per the methods described in Chapter 4.

The event times, e_i's, are the values the simulation clock takes on after $e_0 = 0$. At $time = 0$ the server is idle. The time t_1 of the first arrival is determined by generating A_1 and adding it to 0. The simulation clock is then advanced from e_0 to the next event time e_1. In Figure 3.6 the curved arrows represent advancing the simulation clock.

The first customer arriving at time t_1 finds the server idle and immediately enters service with a delay in queue of $D_1 = 0$. The status of the server is changed from

idle to busy. The time c_1 when the customer will complete service is computed by generating the random variate S_1 and adding it to t_1.

The time of the second arrival, t_2, is computed as $t_2 = t_1 + A_2$, where A_2 is another generated random arrival time. If $t_2 < c_1$ as shown in Figure 3.6, the simulation clock is advanced from e_1 to the time of the next event, $e_2 = t_2$. If c_1 was less than t_2, the clock would be advanced from e_1 to c_1.

The customer arriving at time t_2 finds the server already busy. The number of customers in the queue is increased from 0 to 1 and the time of arrival of this customer is recorded.

Next the time of the third arrival, t_3, is computed as $t_3 = t_2 + A_3$. If $c_1 < t_3$ as depicted in the figure, the simulation clock is advanced from e_2 to the time of the next event, $e_3 = c_1$, where the first customer completes service and departs. The customer in the queue that arrived at time t_2 now begins service. The corresponding delay in queue and service completion times are computed as $D_2 = c_1 - t_2$ and $c_2 = c_1 + S_2$ while the number of customers in the queue is decreased from 1 to 0.

If $t_3 < c_2$ the simulation clock is advanced from e_3 to the next event time $e_4 = t3$ as shown in Figure 3.6. If c_2 was less than t_3, the clock would be advanced from e_2 to c_2, etc.

With the sequence of events per Figure 3.6, it can be seen that the server is idle between times c_2 and t_3 before the third customer begins service.

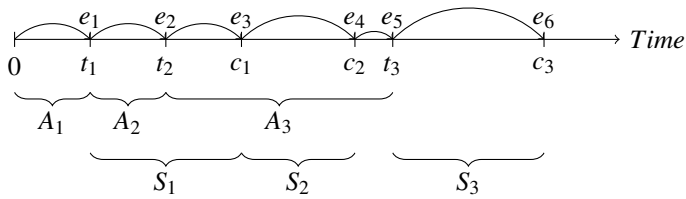

FIGURE 3.6: Next Event Time Advance Approach

3.2.2 COMMON PROCESSES AND OBJECTS IN DISCRETE SYSTEMS

Discrete event simulation views systems and processes as interconnected event-based flows of entities through queues and activities. This view corresponds well with intrinsic, measurable, real-world phenomena. Using this view, discrete event simulation represents system behavior using a set of common constructs, described in this section primarily following the concepts and descriptions in [16]. Capabilities to represent these are to be expected of discrete event modeling tools.

In graphical modeling tools these processes may be represented as visual icons called nodes, blocks, modules, or similar names. They may be termed functions or modules in a modeling language. In all cases they represent a model piece or construct with associated behavior logic.

The moving objects in a system whose motion results in events are typically called entities or items. Interconnected nodes represent entity movement routes. The broad similarity of tools is based on movement and accumulation of entities along these connecting paths through the nodes. The modeler logically connects the nodes to represent the entity flows in a system. Model construction entails selection and linking of appropriate model pieces and specifying their parameters.

Knowledge of common processes helps the modeler understand system behavior as well as expected features of modeling tools. The modeler can better identify desirable modeling capabilities for specific problems. Later in this section are graphical and programmed examples of these capabilities with representative and widely used tools in the field. Additional references for more details on discrete event simulation can be found in [17], [2], and [16].

Entity Movement

Dynamism in discrete systems is caused by the movement of entities which results in the occurrence of events that change the system state over time. In most systems the entities enter the system as inputs through the system boundary, move between the components within the system, and may leave the system boundary in the form of output from the system. In other cases the system may initially contain some entities that move through the system and possibly leave. Some entities may never leave the system boundary.

Entity Creation

To create entities entering a system (e.g. cars arriving for service or manufacturing pieces arriving at a facility) a process for the entity source is required. The simulation module or node type representing this process may be called a *source*, *create*, *generate*, etc. A creation node generates entities with desired timings. It should allow for setting the time of creation of the first entity and the times between entity arrivals as either constant or random variables. For the latter a random variate generator provides samples from probability distribution functions by specifying values of its parameters.

Entities may be assigned names when they are generated. This attribute may be used for entity routing through branches on the basis of the entity type, or for specialized treatment of different entities.

Starting and stopping criteria for creation of entities can be dictated. The creation module should allow for stopping the simulation either after a certain amount of time has elapsed or after a specified number of entities. It is also desirable to allow for the batch creation of entities, i.e., the creation of more than one entity at each arrival time.

One may need to initialize the system with entities already existing at accumulation points. Nodes or modules that correspond to the accumulation points (queues) should provide for initialization of the desired number of entities in each queue.

Entity Termination

Entities that enter a system may also leave. A provision for entity departure may be called a *terminate*, *sink*, *depart*, *out*, or similar module name in modeling tools to represent the path end for an entity. A simulation may end with some, all, or none of the entities remaining in the system depending on creation and termination parameters. Entity termination may be a simulation stopping criteria. One may specify the number of entity terminations required to end the simulation run.

Entity Traversal

Entities traverse through paths connecting system components. The paths may have associated delay times or their capacities may be limited. An explicit delay object or node may used to generate a delay corresponding to the travel time of an entity from one node to the other. The delay times may be specified as a constant, a random variable, a user variable, another expression, or be a function of an entity attribute. Path routes can be selectable based on logical conditions, probabilities, or the status of alternative queues on different paths.

Entities traversing a system may also undergo multiplication into more entities, or several entities join to or from a single entity. Carriers can be used to transport batched entities through the network. Grouped items may traverse a carrier entity (e.g. public transit vehicle) and subsequently ungroup. Previously batched entities may become ungrouped, e.g. the unloading of packages from a carrier. Entities that are grouped or assembled together may lose their individual attributes. There may be grouping conditions based on the number of entities or their attributes. An example of different entity types joining would be the assembling of manufactured items from smaller heterogeneous pieces.

Entity Use of Resources

Resources are commodities used by entities as they traverse a system. They represent servers, operators, machines, etc. and have associated service times. They provide services where entities remain while being processed. Arriving entities must wait until the server is free. Resources (sometimes called facilities) reside at fixed system locations and are made available to entities as needed. Resources usually have a base stock with a fixed level of resource units (e.g. serving capacity).

At event times the entities seize resources, use them with an associated delay, and the resources generally become available again for the next entity. Entity priority schemes are possible, such as special entities that jump to the head of a queue or interrupt another in service. There may also be parallel servers available for entities, each possibly with different service characteristics.

Entity Accumulation (Queues)

Entities move through the system and may wait at accumulation points. In discrete systems these accumulation points are called *queues* as the entities line up in order.

Queues form when entities need resource facilities that may be busy or otherwise unavailable. A queue represents a buffer before a resource or facility.

Priority schemes may be specified for leaving the queue to be served (e.g. First-In-First-Out (FIFO), Last-In-First-Out (LIFO) and others). Multiple queues may be formed for parallel servers. There may be a queue waiting-area capacity limit constraint, or threshold waiting times may be specified for entities to depart a queue.

Entities may also accumulate and form queues at logical gates or switches on paths where entities require permission to proceed. Rules may be specified before entities enter service, such as a required matching with another entity type. An example is an assembly operation for incoming entities that are synchronized and integrated to form new entities.

Auxiliary Operations

Auxiliary operations used in discrete event modeling include handling and utilization of variables, manipulation of entities (e.g. transfer, delete, copy), file operations and others. Custom user variables may be specified along with default system variables.

Statistics specification provides for measuring and assessing system performance data. Statistics include both observation-based (e.g. waiting times) and time-based statistics (e.g. utilization) as covered in Sections 1.6.1 and 6.5.1. Capturing of entity event times for arrival, departure, etc., allow for measures such as waiting time or queue length. Server-based statistics include resource utilization measures. For all of these it may be possible to specify only transient or steady-state data collection.

Visual tracing of entities along routes and other animation are additional useful capabilities. These may be helpful to the modeler in debugging and model verification, or external validation and presentation with stakeholders.

Example: Discrete Processes and Model Objects

Figure 3.7 shows a graphical ExtendSim [13] model of a single server queue demonstrating basic capabilities for modeling discrete processes with model blocks. Entities are created, they traverse the system, accumulate, use a resource, and terminate. An operation for generating random numbers for event timing is also included.

Entities are generated in the *Create* block and transition to the *Queue* block awaiting service in the *Activity* block. The service time is dictated by a random number sampled from the *Random Number* block and the entities leave the system through the *Exit* block. See further annotations in the figure. Also see Chapter 7 for an applied case study using ExtendSim for modeling discrete processes.

Example: Car Charging Station Model with SimPy

Discrete event modeling and simulation will be illustrated in detail for the electric car charging station scenarios. This example is elaborated here and in subsequent chapters to demonstrate basic discrete event model components, key statistics and illustrate the overall modeling process steps. The examples will implement modeling

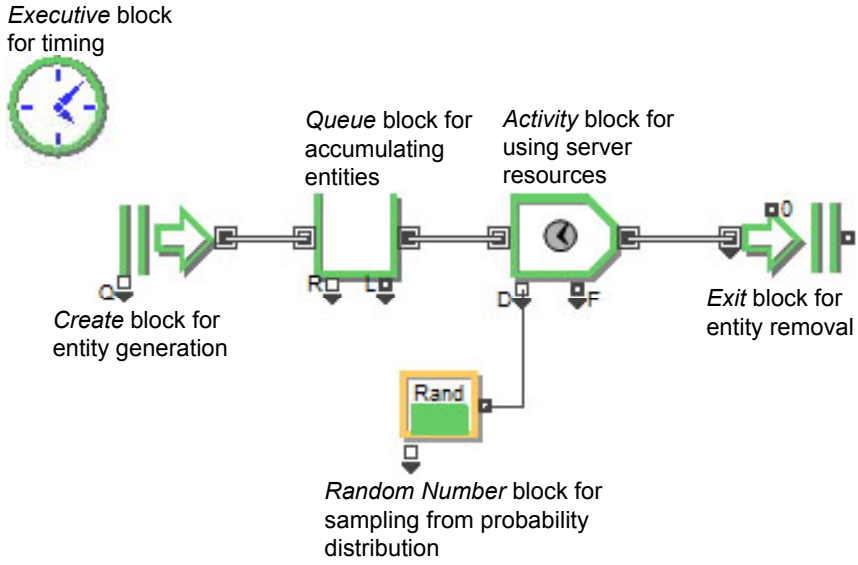

FIGURE 3.7: ExtendSim Model for Single Server Queue with Common Process Blocks

constructs using the SimPy discrete event simulation library written in the Python language. It is open source and accessible to readers to reproduce these examples, extend the source code provided in this chapter and Appendix B, and use for other applications. See [27] for additional details and downloading of SimPy. An independent technical overview of discrete event simulation using SimPy can also be found in [20].

SimPy is a process-oriented discrete event simulation library, so the behavior of active entities like cars or customers is modeled with *processes*. All processes live in an *environment*. They interact with the environment and with each other via *events*.

Processes are described by simple generators that declare functions that behave like an iterator, i.e., they can be used in a loop. During their lifetime, the processes create events and *yield* them waiting to be triggered. When a process yields an event, the process gets suspended. SimPy resumes the process when the event occurs (gets triggered).

A model for the charging station will require instantiating the SimPy simulation environment, defining an electric car process (generator), and a resource for the charging station. We will illustrate building up the model from the beginning and incrementally add complexity. In the subsequent code examples, any text following the "#" character are embedded comments.

Initializing the SimPy simulation environment can be done as in Figure 3.8. The first statement in the source code imports the SimPy library in order to use its func-

tionality. Creating an instance of the `simpy.Environment` will be needed for a car process and thus will be passed into the car process function. The resources used by car entities will also be attached to it.

```
import simpy

# instantiate execution environment for simulation
environment = simpy.Environment()
```

FIGURE 3.8: Initializing Simulation Environment

The execution environment is a class object with associated methods that will be used. The methods use a dot notation, e.g. `environment.now` refers to the current simulation time.

The electric car objects will first need a resource for the charging station. The resource can be defined per Figure 3.9 with the `simpy.Resource` method specifying the resource environment and characteristics. The resource is parameterized by its capacity (e.g. `capacity=1` refers to a single charging bay). This capacity will be varied later for simulation experiments with different operating scenarios.

```
# charging station resource
charging_station = simpy.Resource(environment,
    capacity=1)
```

FIGURE 3.9: Defining Charging Station Resource

The electric car (event) behavior must also be defined before creating actual car instances. The cars will arrive, use the charging resource, and leave. The car process generator is listed in Figure 3.10 defining the events and is attached to the previously defined charging station resource.

This function in Figure 3.10 generates cars during the simulation with their instrinic process behaviors described in the code for arriving, charging, and departing events. Other actions for statistics collection or output statements can also go in the generator. The car process generator also requires a reference to the named `environment` in order to create new events in it.

The first `yield` statement suspends execution until the arrival time occurs, and then triggers the event for the charging station resource. The resource is requested via the `yield request` statement, and the car will enter service when the charging resource is available. If the server is free at arrival time, then it can start service immediately and the code moves on to the next `yield` for the charging time end event. If the server is busy, the car is automatically queued by the charging resource.

```
# electric car process generator
def electric_car(environment, name, charging_station,
    arrival_time, charging_time):
     # trigger arrival event at charging station
     yield environment.timeout(arrival_time)

     # request charging bay resource
     with charging_station.request() as request:
         yield request
         # charge car battery
         yield environment.timeout(charging_time)
```

FIGURE 3.10: Defining Electric Car Process

When the resource eventually becomes available the car begins service, which ends after the charging time elapses. The car process will pass the control flow back to the simulation once the last `yield` statement is reached.

Next the car arrival times and charging times must be specified before a simulation commences. This first illustration will use the fixed times in Figure 3.11, initially demonstrated in Section 1.6.1. Note that in a typical simulation with randomness, the times are generated within the simulation, and are not a priori constants (random times will be demonstrated next).

```
interarrival_times = [2, 8, 7, 2, 11, 3, 15, 9]
charging_times = [11, 8, 5, 8, 8, 5, 10, 12]
```

FIGURE 3.11: Defining Fixed Event Times

Next a loop to create all the events must be specified, which is shown in Figure 3.12. The code defines a fixed number of car processes to be created. Finally, the simulation is started after the car generation loop by calling the run method `environment.run()`. Note that the simulation duration may be specified in other ways such as a fixed ending time vs. the fixed number of car processes.

Running the program will produce the event output and summary statistics in Figure 3.13 (after adding some output print statements and statistics collection logic listed in Appendix B). The numerical results are identical to the manual calculations in Chapter 1.

Few changes are necessary to model random event times. After importing a random number generation library, the event generation loop simply replaces the fixed times with exponentially distributed random times per Figure 3.14. The provided val-

```
# simulate car processes
for i in range(7):
     arrival_time += interarrival_times[i]
     charging_time = charging_times[i]
     environment.process(electric_car(environment, '
         Car %d' % i, charging_station, arrival_time,
         charging_time))

environment.run()
```

FIGURE 3.12: Event Generation Loop and Simulation Start

ues are reciprocals of the distribution means for the interarrival and charging times respectively (i.e., the interarrival time mean = 6 and charging time mean = 5).
The full Python code for these examples with additional print statements, data initializations, and statistics computation is listed in Appendix B.

3.3 AGENT BASED

Agent-based modeling uses autonomous decision-making entities called *agents* with rules of behavior that direct their interaction with each other and their *environment*. An agent must possess some degree of autonomy and be distinguishable from its environment. It must perform behaviors or tasks without direct external control reacting to its environment and other agents. Transitions move agents between *states* similar to how rates move entities between levels in continuous models.

Agents modeled may include many types of interacting individuals or groups such people, vehicles, robots, cells, data packets on a network, teams, organizations, and many others. Agents can represent entities that do not have a physical basis but are entities that perform tasks such as gathering information, or modeling the evolution of cooperation. Agent behavior can be reactive, e.g. changing state or taking action based on fixed rules, or adaptive after updating internal logic rules via learning.

Agents interact in environments, which are the spaces in which they behave. The environment may be discrete, continuous, combined discrete/continuous, or characterized by networks. Agent-based modeling can thus be combined with other simulation methods used in engineering sciences. For example, when simulating the interaction with the environment, the environment may be represented by a discrete or continuous field.

Agent-based simulations are a suitable tool to study complex systems with many interacting entities and non-linear interactions among them. Emergent behaviors can result, which are patterns generated by the interactions of the agents, which are often unexpected.

Agent-based modeling is relatively new without an extensive history of engineer-

```
Time  Event
2 Car 0 Arriving at station
2 Car 0 Entering a bay
10 Car 1 Arriving at station
13 Car 0 Charged and leaving
13 Car 1 Entering a bay
17 Car 2 Arriving at station
19 Car 3 Arriving at station
21 Car 1 Charged and leaving
21 Car 2 Entering a bay
26 Car 2 Charged and leaving
26 Car 3 Entering a bay
30 Car 4 Arriving at station
33 Car 5 Arriving at station
34 Car 3 Charged and leaving
34 Car 4 Entering a bay
42 Car 4 Charged and leaving
42 Car 5 Entering a bay
47 Car 5 Charged and leaving
48 Car 6 Arriving at station
48 Car 6 Entering a bay
58 Car 6 Charged and leaving

Summary
# of cars served = 7
Average waiting time: 3.9 minutes
Utilization: 0.948
Average queue length: 0.466
```

FIGURE 3.13: Program Output

ing usage like discrete event and continuous modeling. Its usage is increasing however. Agent-based models have been primarily developed for socio-economic systems simulating the interactions of autonomous agents (both individual or collective entities such as organizations or groups) to assess their effects on the system as a whole.

There are also many scientific and engineering applications to model groups of actors and their interactions based on behavioral rules. Urban systems in civil engineering to study people patterns, communities, vehicles, and traffic patterns are a natural fit. Example applications to-date in civil engineering include water resources [3], civil infrastructure [26], construction management [28], and more. They have been used to model complex healthcare systems and disaster response to simulate the movement of many individual people, and the consequent behavior of the crowd,

```
# simulate car processes
for i in range(7):
      arrival_time += random.expovariate(0.16)
      charging_time = random.expovariate(0.2)
      environment.process(electric_car(environment, '
          Car %d' % i, charging_station, arrival_time,
          charging_time))

environment.run()
```

FIGURE 3.14: Event Generation with Random Times

as they make their way out of the buildings and find transport to medical facilities.

Modeling dynamically interacting rule-based agents is ideal for transportation systems with emergent phenomena, when individual behavior is nonlinear and changes over time with fluctuations like traffic jams. It can be more natural to describe individual behavior activities compared to using processes in discrete event models.

Agent-based modeling is relevant for the engineering of systems of systems. Systems engineers can investigate alternative architectures and gain an understanding of the impact of the behaviors of individual systems on emergent behaviors. Other examples of engineering applications include supply chain optimization, logistics, distributed computing, and organizational and team behavior.

3.3.1 AGENT-BASED MODEL ELEMENTS

Agent-based modeling is typically implemented with an object-oriented programming where data and methods (operations) are encapsulated in objects that can manipulate their own data and interact with other objects. The behaviors and interactions of the agents may be formalized by equations, but more generally they may be specified through decision rules, such as if-then statements or logical operations.

Table 3.3 lists typical agent-based model elements. These definitions are consistent with the implementation in [8] and may vary slightly in other modeling tools.

TABLE 3.3: Agent-Based Model Elements

Element	Description
Population	An interacting group or collection of agents during a simulation described by the type of agents and the population size.
Agent	An individual actor in a population governed by its own rules of behavior.
State	An on/off switch for an agent. When the state is active, the agent is in that state and vice versa. One or more sets of states may be placed in an agent.
Transition	Transitions move agents between states in a model. When a transition is activated, an agent moves from one state to another. They are configured by what triggers the transition. Some different types of triggers may include a timeout, a probability, or a condition based on logical relationships to other agents or events.
Action	An action manipulates agents during a simulation. They are defined by what triggers the action and what the action does when triggered. An action can be to move an agent, to change a primitive's value, to add a connection to an agent, or other.

Example: Agent-Based Model for Car Behavior

Figure 3.15 shows an agent-based model for electric cars acting as interacting agents. This is a web-based simulation model using Insight Maker (see Appendix A). Consistent with the elements in Table 3.3, the population of cars is signified by the cloud at the top. Cars are defined as the agents that move between the states of *Waiting*, *Parked*, *Impatient*, and *Driving*.

This model is essentially a queuing process like the previous car charging example, but a variation with respect to the discrete event model is the consideration of cars leaving after waiting too long. Note the car behavior is actually dictated by a human agent inside and could alternatively be termed a driver rather than a vehicle.

Transitions are indicated by arrows in the model for moving between states. These are triggered by the decision to start looking for a charging space, the existence of an empty space, giving up after waiting too long, and completion of charging to

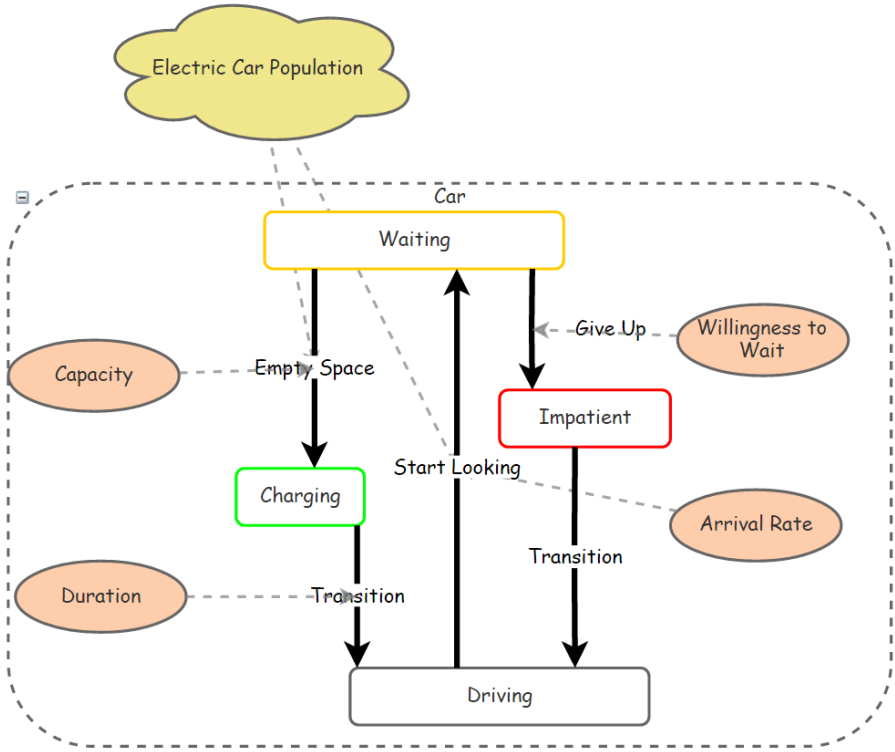

FIGURE 3.15: Example Agent-Based Model for Electric Car Charging

start driving again. The ovals are other model variables used in the logical rules of behavior. Dashed lines represent information links passing values.

A population of cars start off in a *Driving* state. At each cycle, according to a Poisson distribution defined by *Arrival Rate*, some cars transition to *Waiting* as they wait for an empty charging space according to:

```
RandPoisson(Round([Arrival Rate]))/Count(FindState([
    Electric Car Population], [Driving]))
```

The number of cars in the *Waiting* state represents a queue. If an empty bay is available for a car, then the state transitions to *Charging* per:

```
[Parking Capacity] > Count(FindState([Electric Car
    Population],[Charging]))
```

The cars remain in a *Charging* state according to a normal distribution and then transition back to *Driving* as they leave, modeled as:

```
RandNormal([Duration], [Duration]/2)
```

The model uses a timeout for the impatience behavior. If a car is in the *Waiting* state for a period longer than *Willingness to Wait*, then the state times out and transitions to *Impatient* and immediately transitions to *Driving* again. Figures 3.16 and 3.17 show a simulation timeline and state transition map for a simulation run.

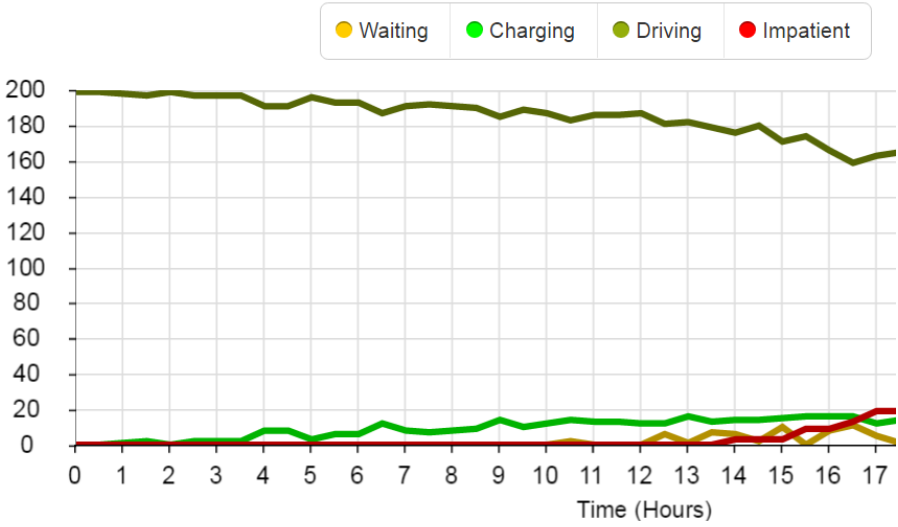

FIGURE 3.16: Simulation Timeline

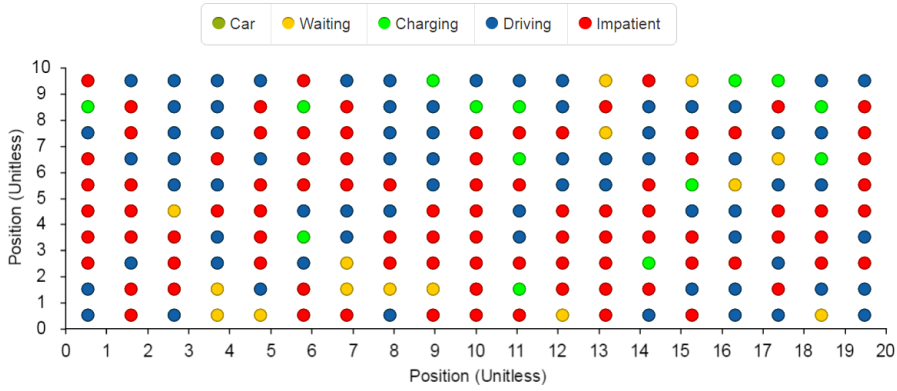

FIGURE 3.17: State Transition Map

3.4 SUMMARY

Systems and/or their models can be discrete, continuous, or a combination. All classes of systems may be represented by any of the model types. In continuous

models the variables change continuously with respect to time. Time advancement is in equal small increments, with all variables recomputed at each time step. Discrete system models are event based, whereby an event calendar is used for timing. State changes occur in discrete systems at aperiodic times depending on the event times. Time advancement is from one event to the next, with no changes in variables between events.

Before choosing a modeling method for engineering applications, an important question that should be answered is, "What is continuous and what is truly discrete?" Are there discrete aspects that need to be preserved for the purpose of the study or is an aggregate view sufficient?

System dynamics is the most widely used form of continuous simulation. In the continuous view, process entities are represented as aggregated flows over time. The approach does not track individual events; rather, flowing entities are treated in the aggregate and systems can be described through differential equations. System dynamics provides a rich modeling environment incorporating many types of formulations for this.

System dynamics model elements are levels, sources and sinks, rates (flows), information links, and auxiliary variables. Levels represent real-world accumulations and serve as the state variables describing a system at any point in time. Rates are the flows over time that affect the levels. Rates may represent decisions or policy statements. Rates are computed as a function of levels, constants, and auxiliaries. Sources and sinks indicate that flows come from or go to somewhere external to the system. Models are built using conserved flow chains where the levels store the flowing entities.

Discrete event models consist of flows of individual entities, characterized by the values of their respective attributes. The characteristics of the entities can change during a simulation run. These changes occur instantaneously as the simulated time lapses. The entities move through a system represented as a network of nodes, perform activities by using resources, and create events that change the state of a system. The next-event time-advance approach is the standard procedure used for computing in discrete event modeling.

Discrete event simulation represents system behavior using a set of common constructs that correspond well with intrinsic, measurable, real-world phenomena. Standard discrete processes include entity creation, entity movement and traversal, entity use of resources, entity accumulation (queues), and entity termination. In tools they are used to represent the movement and accumulation of entities along paths through system nodes. Modeling tools also provide auxiliary operations and statistics specification for measuring and assessing system performance data.

Agent-based models contain agents that are autonomous decision-making entities with rules of behavior that direct their interaction with each other and their environment. Agents may include many types of interacting individuals or groups. Agents interact in environments, which are the spaces in which they behave. The environment may be discrete, continuous, combined discrete/continuous, or characterized by networks.

Agent-based model elements include agents, their populations, states, transitions, and actions. Individual agents are governed by their own rules of behavior and populations store agents during a simulation. Agents have internal states, and transitions are used to move agents between states in a model. Actions manipulate agents during a simulation and are defined with event triggers.

This chapter has demonstrated simple, illustrative models for continuous, discrete event and agent-based types of modeling and simulation. The reader is encouraged to look further in the references and upcoming chapters, see Appendix B for modeling tools, experiment with the tools and the provided models in this book.