An Integrated Approach to System Modelling using a Synthesis of Artificial Intelligence, Software Engineering and Simulation Methodologies*

Paul A. Fishwick
University of Florida.

Abstract

Traditional computer simulation terminology includes taxonomic divisions with terms such as "discrete event," "continuous," and "process oriented." Even though such terms have become familiar to simulation researchers, the terminology is distinct from other disciplines —such as artificial intelligence and software engineering— which have similar goals relating specifically to modelling dynamic systems. There is a need to unify terminology among these disciplines so that system modelling is formalized in a common framework. We present a perspective that serves to characterize simulation models in terms of their procedural versus declarative orientations since these two orientations are prevalent throughout most modelling disciplines that we have encountered. We used a sample dynamic system (e.g., two jug problem) found in artificial intelligence to highlight the connecting threads in system modelling within each discipline. Moreover, in teaching simulation students using this perspective, we have had considerable success in relating the field of modelling within computer simulation to other sub-disciplines within computer science. The result is that modelling in simulation can be more easily compared-with and contrasted-against other modelling approaches in computer science.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**] Requirements/Specifications - *Methodologies*; D.2.2 [**Software Engineering**] Tools and Techniques - *Computer-aided software engineering*; D.2.10 [**Software Engineering**] Design - *Methodologies, Representation*; I.2.0 [**Artificial Intelligence**] General - *Cognitive Simulation*; I.2.4 [**Artificial Intelligence**] Knowledge Representation Formalisms and Methods - *Representations*; I.6.1 [**Simulation and Modeling**] Simulation Theory - *Model classification, systems theory*; I.6.5 [**Simulation and Modeling**] Model Development - *Modeling methodologies*. I.6.8 [**Simulation and Modeling**] Types of Simulation - *Combined, Discrete event, Continuous*.
General Terms: Modeling.
Additional Key Words and Phrases: Multimodeling, Abstraction Levels.

*Author's Address: Paul A. Fishwick, Dept. of Computer and Information Sciences, University of Florida, Bldg. CSE, Room 301, Gainesville, FL 32611. This paper is an enhanced and comprehensive manuscript based on initial results [34] prepared for the Third Conference on AI, Simulation and Planning in High Autonomy Systems.

1 Introduction

Computer simulation is the creation and execution of dynamical models employed for understanding system behavior. Even though the literature base in simulation is quite large, many simulation textbooks —serving as archives for future researchers and students— cover simulation methodology using the classic taxonomies including terms such as "continuous," "discrete event" and "event-oriented." This type of taxonomy may seem comfortable and familiar; however, we will demonstrate that for the task of *modelling*, we need to strengthen ties with artificial intelligence (AI) and software engineering (SE) to provide a more uniform view of system modelling that is less focussed on one particular *simulation* method (such as discrete event simulation), and more attuned to modelling continuous, discrete models and spatial models using a unified framework. First, we stress that "modelling" and "simulation" are two different tasks and we will attempt not to use them interchangeably; one model may be simulated using several different simulation algorithms. When reviewing "world views" in discrete event simulation, it is sometimes tempting to confuse a method of modelling such as *process orientation* (i.e., a functional modelling approach) with a method of simulating or executing a model such as *event scheduling* (i.e., an approach of simulating parallelism of a model on a sequential architecture).

Our emphasis is on *modelling methodology* [89, 65, 66, 25, 24] rather than *analysis methodology*. Methodology in analysis has received a much more comprehensive treatment in the general simulation literature [54, 9] as compared with methodology in modelling. The art and science of modelling has a firm foundation within the computer science discipline, and consequently, we find several examples in computer science that serve to bolster our argument for a more integrated modelling approach that encapsulates many of the modelling methods in AI, SE and simulation. There are key facets of simulation modelling that are strongly related to parts of AI and SE, and these facets appear to be more fundamental to the nature of simulation modelling than are the current divisions along the lines of "discrete event," "continuous" and "combined" or "activity scanning," "event scheduling" and "process interaction" as frequently discussed in the simulation literature. For example, the data flow diagram (DFD) in SE and the block model in simulation and control engineering represent the same modelling technique; the DFD has elements including functional blocks, inputs, outputs, coupling and hierarchy; the same is true of the functional block model in SE. In the past, the two modelling camps could be considered completely separate entitites; however, with the onset of distributed computing and the encapsulation of code in physically separated —and, therefore, modelled— objects, software engineers are every bit as interested in modelling continuous and discrete data flows as are simulation modellers. In SE, a data flow diagram represents a modelling technique utilized for a variety of models. In simulation, we should also use this functional category of modelling to represent queuing networks, digital logic circuits and systems for control engineering; there are some differences in the data flow (discrete versus continuous), although this is a minor difference and does not represent a fundamental shift in modelling practice.[1]

Our hypothesis is that, while the current taxonomies for modelling in simulation have

[1] In terms of *closed form analysis*, it is natural to form a dichotomy between "discrete" and "continuous" since the solution methods are different. With system *modelling*, though, the differences are not as pronounced.

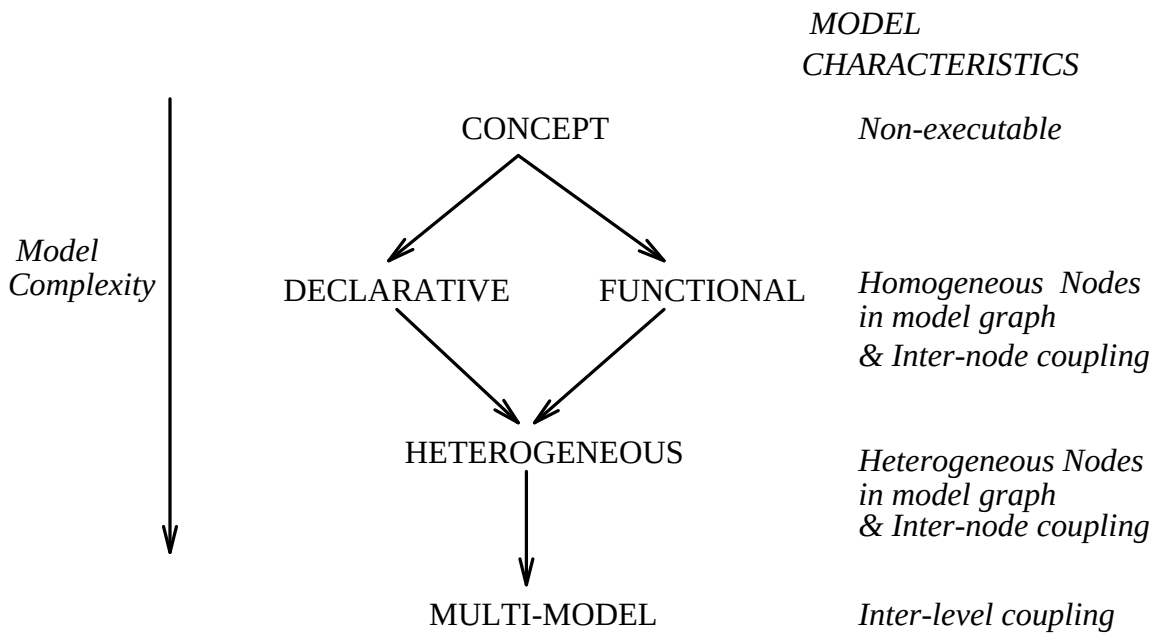


Figure 1: The proposed modelling paradigm.

served their purpose, we need to consider another taxonomy that fits more closely with related modelling efforts in the AI and SE communities. The modelling of dynamic systems has become a widespread phenomenon and is no longer specific to the simulation discipline; we need to embrace these competing areas to derive a common taxonomy or "world view" with regard to system modelling. This alternative taxonomy is based on a set of modelling approaches depicted in figure 1. In fig. 1, the first type of model is the *concept model* which corresponds to an object model within SE or a system entity structure [91]. The concept model is non-executable, and serves as a knowledge base for the system. From the concept model, one can derive either one of two basic modelling forms: *declarative* and *functional*. Declarative models emphasize state transitions, while functional models emphasize operational or event oriented modelling. Declarative and functional model forms are prevalent in all three disciplines. For instance, programming languages are often categorized into either declarative [1, 49] or functional [48, 6] types. The declarative programming languages (such as Prolog) emphasize changes in state where "states" are best coded as particular data structures — a simple variable being the most commonly used form for a state. Functional programming languages concentrate on functional composition while de-emphasizing side effects[2]. One may combine these forms to synthesize *heterogeneous* models where model graph nodes may be of different types. Finally, the *multimodel* [33, 57, 37] is the most comprehensive type of model that supports multiple models tied together with homomorphic mappings from one model to another [28]. The multimodel approach is a generalization of combined simulation modelling [18, 71] where models may be of many different types — not just a mixture of discrete event and continuous components. The proposed taxonomy is an extension to the object oriented modelling paradigm. While we adhere to the object model as a good basis for conceptual, non-executable models, we depart from the norm

[2]The "side effects" for software encoding dynamical system behavior are actually the states of the system.

when introducing how additional modelling techniques such as production systems, track-based animation and System Dynamics fit within the overall framework. In the usual object oriented modelling approach within SE [79, 16], specific modelling methods such as FSA modelling and DFDs are promoted. We treat these two types of models as instances of a class rather than a class by itself. Our approach is to stress the utility of having functional and declarative *classes* of models. In our declarative modelling class, for instance, we list several types of models including FSAs, production rules, and logic based models.

We are interested in overviewing the commonalities in the modelling process as well as asking fundamentally interesting historical questions such as "What are the causes or catalysts aiding in the relatively recent convergence in AI, SE and simulation modelling?" and "How, precisely, will modellers and decision makers benefit in this convergence?" The integrated style is being used by several simulation researchers [92, 60, 78, 66, 33, 3]; however, it has not yet penetrated the general simulation textbook literature as a more powerful paradigm for modelling dynamical systems. Although the introduction of new dichotomies, taxonomies and reorganizations is a philosophical issue, we have had first hand experience in teaching the proposed simulation modelling methodology to college seniors and first year graduate students at the University of Florida with much success. Students who take, for instance, courses in AI and SE can take a class in simulation that builds upon —rather than replaces— recently learned model forms.

Two key aspects of the view shown in fig. 1 are the *declarative* versus the *functional* perspectives. These two modelling views form a dichotomy in that most modelling methods in simulation are oriented toward one view rather than the other. System Dynamics models [74] and block models, for instance have functional orientations. The term "dichotomy" is used somewhat loosely, though, since there exist several kinds of modelling methods such as Petri nets [70] that have equal shares of declarative (i.e., place) and functional (i.e., transition) sub-representations.

We first overview why we chose these two categories in our attempt to synthesize system modelling techniques in AI, SE and simulation. Within the context of a two jug problem in AI, we then discuss how the jug system can be viewed from these two perspectives. Finally, the multimodel approach is illustrated to demonstrate how models of different types may be coupled together to form a *multimodel* or "model database." We close with some conclusions and research currently underway to extend these ideas.

2 Synthesis of Modelling Techniques

2.1 AI & Simulation Models

Within AI, one is concerned with how to model human thought and decision making. Often, the decision making is heuristically based, and there is incomplete knowledge about the domain [56]. This "incompleteness" should be programmed into the dynamical model where it is present. For the past decade, the interface area between AI and simulation has grown, and several papers and texts have appeared [63, 64, 84, 36, 43]. Simulation models have characteristically been composed of simple entities and objects; however, the introduction of autonomous agents within the queue into a model has suggested that simulationists use AI

models in places where autonomy is present. While a simple queuing network avoids the use of knowledge based simulation models, a more detailed model would include beliefs, plans and intentions of the autonomous agents —in the queue— at some level of detail. Trajectories of projectiles and three phase motors are comparatively simple to model because these objects operate in accordance with natural laws that are controlled through engineering; the laws and control methods for non-autonomous objects are fairly simple with regard to model complexity. Models of intelligent agents are much more complicated due to the sophisticated reasoning abilities of humans and of robots that are endowed with human-like reasoning.

Autonomy has, therefore, spawned the creation of knowledge based models that contain a variety of natural, artificial and intelligent objects interacting and reasoning in a complex environment. The fields of qualitative reasoning and qualitative physics [83, 13] are evidence of the AI interest in system modelling from the perspective of mathematical reasoning. In addition to autonomy playing a critical role, incomplete knowledge is ever present within models and AI researchers have suggested new ways of representing this type of knowledge. While simulationists have used probability theory for representing abstracted quantities, one can also use heuristic rules and constraint based modelling techniques. These types of techniques have been used for declarative and functional modelling [85]; however, they are most useful within conceptual models that serve to enhance our ability to diagnose symptoms, plan future actions and provide common-sense explanations of device behavior [83, 22].

2.2 SE & Simulation Models

Software engineers are pioneering new and novel methods for building system models; tools such as Foresight [5] provide the modeller with the capability of modelling dynamic systems using finite state automata and block modelling all under the umbrella of object organization. Software engineers have developed a keen interest in simulation; there is an apparent convergence between these two areas [76, 68, 7, 8, 61, 45, 46]. Some of our previous research [31, 30, 32, 37] has suggested the study of *model engineering* as a direct analog to software engineering. Within the simulation community, Zeigler presents a theory for modelling autonomous agents [92] while implementing a model engineering methodology. The historical reason for the convergence of the SE and simulation fields lies in the area of distributed and real-time design [72] and computing. Technology has seen the computer decrease in size and cost while increasing in power. This combination of circumstances naturally leads to the use of computers in almost every electro-mechanical device. When software engineers had to concern themselves with modelling only mainframe or workstation software, the modelling process dealt with functional decompositional methods: creating hierarchical routines and stubs while gradually expanding the size and complexity of the program. Now, however, with the ever expanding migration of the microprocessor, the structural components of software models are beginning to act and appear like the physical objects in which the processors live. Modelling distributed software, with its emphasis on communication protocols, is similar to modelling the physical objects for which the software is written.

Thus, the convergence of SE and simulation models stems largely from distributed computing. There are key differences, though, between the end results one obtains. Software engineers want an executable program, while simulationists want to model the performance and lumped behavior of the system. While these two avenues appear to diverge, there is

actually a confluence. The confluence is best seen at a higher level in the decision making process that oversees the use of software engineers and simulationists. That is, the decision maker who wants to create efficient and correct distributed software for his inter-bank transaction project, for instance, is also highly concerned with the efficiency of the entire distributed architecture. Whereas, a decade ago, a project might have involved simulation before or during actual construction of hardware, communications pathways and distributed software, now a project can be created while cleanly integrating the tasks of performance analysis with software development. The ultimate simulation is the actual creation of the software which is then executed; lumped statistical behaviors can easily be obtained when one has the lowest level performance traces. The ultimate software development tool is one that permits modelling the software at a variety of abstraction levels — the lowest level providing detailed behavior of the sort normally associated with program input and output traces. Given this view, simulation is the process of creating abstract versions of programs; the final most detailed simulation is simply the executable program.

3 Terminology

3.1 Using Systems Theory as a Starting Point

For the fundamental building primitives comprising models that represent time-dependent system behavior, we have found systems theory [69, 87, 52, 51] to provide the most mathematically consistent foundation. Systems theory has developed since the early 1960s into a field that made precise the core components of "systems" regardless of the specific discipline (i.e., computer science, biology, chemistry, physics, operations research) [12, 4]. The first formal theories for discrete event simulation [88, 89, 91] were founded upon systems theory, and much recent work has continued this trend. Even though our proposed paradigm is consistent with object oriented modelling, we were surprised at the lack of system theoretic formalism present in AI and SE general; the formal emphasis within SE seems to be within formal verification and correctness of programs. AI has many formalisms of which mathematical logic is the most prominent. On the other hand, simulation literature is relatively weak in its use of logic and in its emphasis on correct programs implementing models; however, recently there has been renewed interest within the use of logic to both verify and simulate models [62].

3.2 The Generic Model Structure

A deterministic system $< T, U, Y, Q, \Omega, \delta, \lambda >$ within classical systems theory [69] is defined as follows:

- T - is the *time* set. For continuous systems [19], $T = \mathcal{R}$ (reals), and for discrete time systems, $T = \mathcal{Z}$ (integers).
- U - is the *input* set containing the possible values of the input to the system.
- Y - is the *output* set.

- Q - is the *state* set.
- Ω - is the set of *admissible* (or *acceptable*) input functions. This contains a set of input functions that could arise during system operation. Often, due to physical limitations, Ω is a subset of the set of all possible input functions ($T \rightarrow U$).
- δ - is the transition function. It is defined as: $\delta : Q \times T \times T \times \Omega \rightarrow Q$.
- λ is the output function, $\lambda : T \times Q \rightarrow Y$.

A system is time invariant if its behavior is not a function of a particular interval of time. This simplifies δ to be of the following form: $\delta : Q \times T \times \Omega \rightarrow Q$. Here, $\delta(s, t, \omega)$ yields the state that results from starting the system in s and applying the input ω for a **duration** of t time units.

This formalism, although concise, is quite general. For structural reasons we employ techniques such as level coupling, inter-level coupling and hierarchy to make the overall system more manageable; however, we first identify some key pieces within the above definition. Q is also known as the *state space* of the system. An element $s \in Q$ is termed the *state* of the system, where the state represents a value that the state components can assume. The state of an ice hockey puck would be (x, y) whereas the state of a two-queue system would be (q_1, q_2) where q_i represents the number of entities in queue i. Normally the state has a structure of an n-tuple: $\{s_1, s_2, \ldots, s_n\}$ however, it is best generalized as a *data structure* — a tuple being one type of data structure. Superstates provide flexibility in describing some model forms [35]. A *superstate* is a subset of a state, therefore, "goal" is the subset of the hockey puck state representing the geometrical region where the goal net is located[3]. A pair consisting of a time and a state (t, s) where $s \in Q$ is called an *event*. Events are points in event space just as states are points in state space. Event space is defined as $Q \times T$. Events normally represent values of state that correspond to definite cognitive or lexical mappings [10, 11]. For instance, in a queueing model we identify an event as an "arrival" but we may not have words to represent the values of other states whose values do not correspond to a cognitive or lexical association. *State* and *Event* are critical aspects of any system and by focussing on one or the other, we form two different sorts of models: *declarative* models that focus on the concept of state, and *functional* models that focus on the concept of event. States represent a "snapshot" of a system, while events, even though they occur at points in time, are naturally associated with functions (i.e, routines, procedures). A change in state is associated with an event, and vice versa; so, there is a duality between states and events. Declarative and functional orientations are discussed widely in both AI and SE, and we propose that they share a common bond with simulation modelling as well.

Given basic elements such as states and events, we can structure models in various ways to aid us in better analyzing systems. By *coupling* functions or states, and by making *networks* and *hierarchies* we can simplify the overall model organization. We will call one model an "abstraction level" or a "perspective." Abstraction levels are discussed by many researchers in SE [58], AI [82] and simulation [28, 29]. A model can contain *homogeneous* or *heterogeneous* node types; simple modelling constructs are homogeneous (such as FSAs), and more complex constructs are heterogeneous (such as System Dynamics graphs, Petri nets or

[3]The concept of superstate in SE is discussed by Davis [21]

bond graphs). If the hierarchical organization is not purely representational (i.e., all levels can be collapsed into a single model) then we can have *multimodels* where many models are attached to one another via behavior-preserving homomorphic links.

4 Concept Models

Models whose components have not been clearly identified in terms of system theoretic categories such as state, event and function are called "concept models" [81]. Conceptual models are a logical first step to modelling; research in these models has many connections in AI, SE, systems science and simulation. At first, it would appear that non-executable models should play no part in simulation. Simulationists have historically not spent a considerable amount of time on "model engineering" (with exceptions noted in the previous section); even though the need for iterating through the modelling process is well understood and appreciated, there is little textbook methodology present to aid simulationists in this key area. Model engineering does not lend itself to a nice neat formalism, and therefore the engineering aspect of simulation modelling is often part art and part science [80]. Despite this difficulty in formalization, model engineering is a critical task and should be a central activity within the simulation field; we want to better understand the very nature of modelling including how and why we choose the models that we do during the course of systems analysis. The area of Systems Dynamics [42, 41, 74, 75] focuses, not only on the formalism for a dynamic model but also, on several key steps during the model building process: 1) causal model without signed information, 2) causal model with signed arcs and loops, 3) flow graph, and finally 4) equations (or program). These steps provide clues as to how we might engineer more generic models. This is not to suggest that we should abandon all of our modelling methods in favor of the Systems Dynamics approach. Instead, we note that one success of the approach is its clearly emphasized model engineering steps.

The first type of model that we want to create is a concept model that emphasizes a objects and their relations to one another. From such a model, we can gradually progress to more system theoretic constructs. The concept model in AI is termed a semantic network [86, 27, 17], while equivalent model in SE is the object model with attribute definition [79, 15, 16, 14]. Most work in simulation, concerning concept models, has been performed along the lines of model specifications [59, 68] and the system entity structure [89, 92]. Since simulation has its formal roots in systems theory and science, we find work relating to conceptual modelling in these areas as well [20, 52, 38]. The semantic network, even though it can serve as a rough cut of a simulation model, was often built as an end in itself, or to facilitate qualitative reasoning via link traversal. Semantic networks are traversed to answer simple questions about a system. Simulationists need such models —to augment their mathematical system models— since there is often a need to ask more than simply "predict when object XYZ reaches point B" or "give the mean idle time for the cashier." It would also be useful for simulationists to be able to ask more abstract questions about a system, and furthermore, to obtain abstract answers that serve in forming a causal explanation of system behavior [77]. The typical AI semantic network would be very weak at producing precise quantitative answers; however, similar claims can be made against an equational simulation model — it produces precise answers but cannot produce simple explanations of behavior in "close-to"

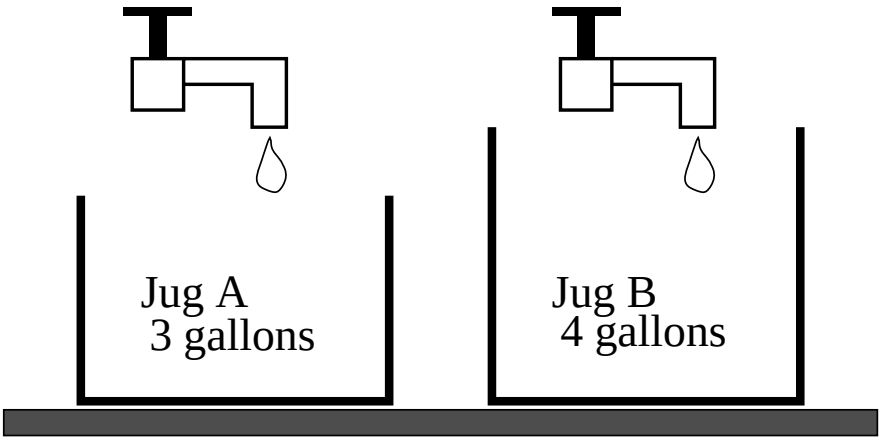


Figure 2: Two jug system.

natural language expression. Therefore, concept models not only provide a "specification" for a simulation model, they *are* simulation models at high levels of abstraction.

The object modelling approach in SE has produced the equivalent of the AI semantic network, but for a different purpose: to permit a software engineer to prototype a large software system. Our approach is similar to object oriented design principles espoused in SE: begin with an object representation and then produce declarative and functional model types. Let's consider the water jug problem in AI [53, 73]. In the water jug problem, there are two water jugs (one with a three gallon capacity, and the other with four gallons). Jug A is the three gallon jug and jug B is the four gallon jug (see figure 2). There are water spigots and no markings on either jug. There are three basic operations that can be performed in this system: emptying a jug, filling a jug, or transferring water from one jug to the other. A concept model can be created by concentrating on key objects, concepts and actions within the system. Rumbaugh et al. [79] present a useful checklist for deriving the object model:

1. Identify objects and classes.
2. Prepare a data dictionary.
3. Identify associations and aggregations among objects.
4. Identify attributes and objects and links.
5. Organize and simplify object classes using inheritance.

There are no hard rules for developing an object model; however, this checklist serves as a starting point by suggesting guidelines. The complete process would involve iterative refinement of the rules. Let's consider each guideline with respect to the two jug system. An object graph will be composed of vertices and arcs, so we start by defining the nodes. Objects will be considered either *classes*, which define broad categories, or *instances*, which define examples within classes. Nouns are often a good place to start when identifying classes. For instance, the noun "tap" is a good class since there are many different types of taps including the two used in the system. If we use the statement of the system description, we arrive at the following classes: *tap* and *container*. We consider tapA and tapB to be examples of

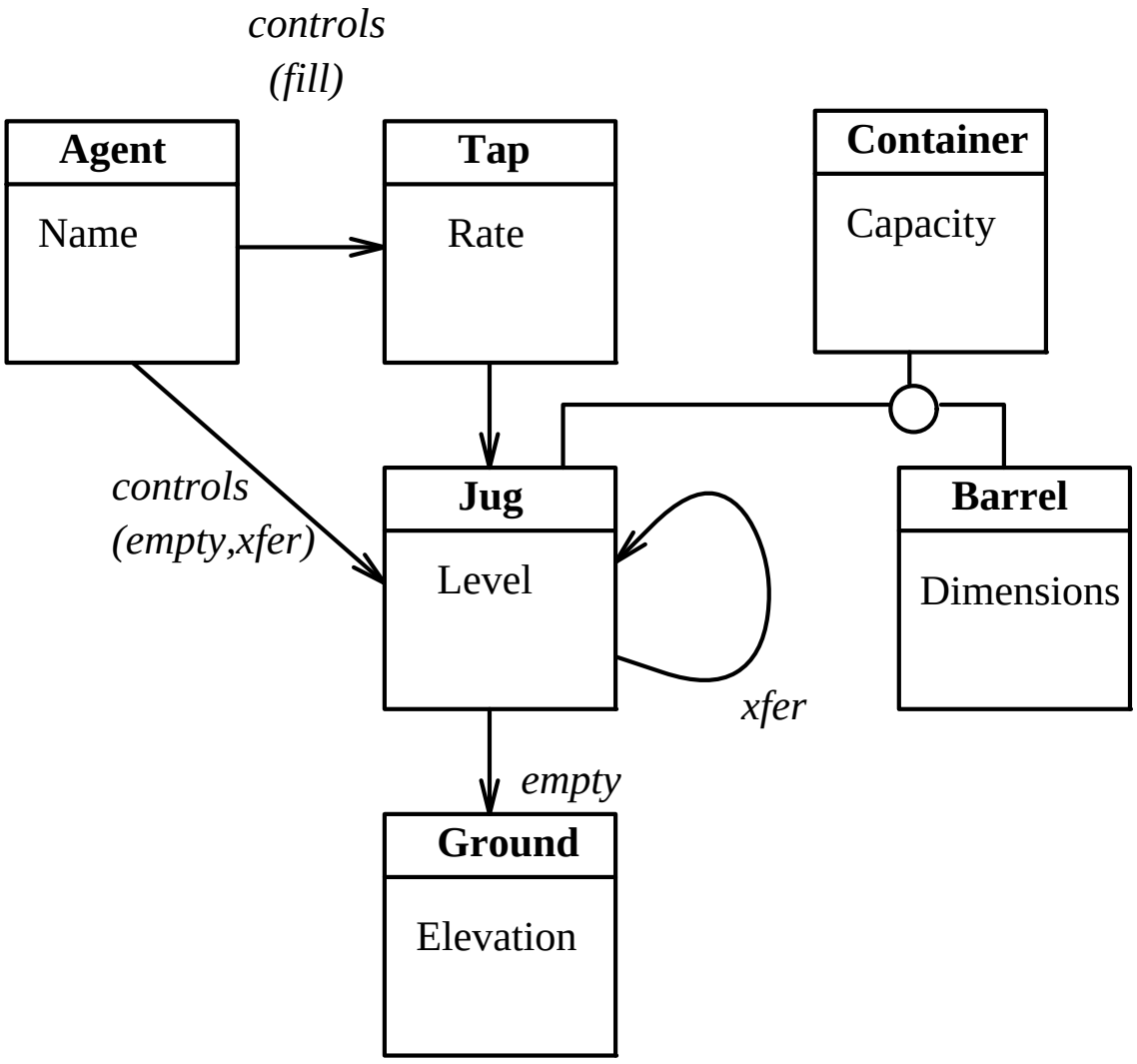


Figure 3: Class model: two jug system.

a tap, and jug and barrel to be sub-classes of *container*. JugA and JugB compose every known jug in this closed "world." We can then create a data dictionary where encyclopedic information about the classes and instances are stored. Our associations often take the form of verbs or operations associated with the system; therefore, *fill*, *empty* and *transfer* are natural selections for associations. The choice of attributes depends on the kinds of questions that we will be asking of the system. If we will —in the section on declarative modelling— define the operations in terms of integral amounts of water, then *level* of water and *rate* of water flow will be key attributes. After some additional iteration we arrive at the concept and instance models depicted in figures 3 and 4.

5 Declarative Models

In declarative modelling, we build models that focus on state representations and state-to-state transitions: given a state and a transition, the model will provide the next state. This

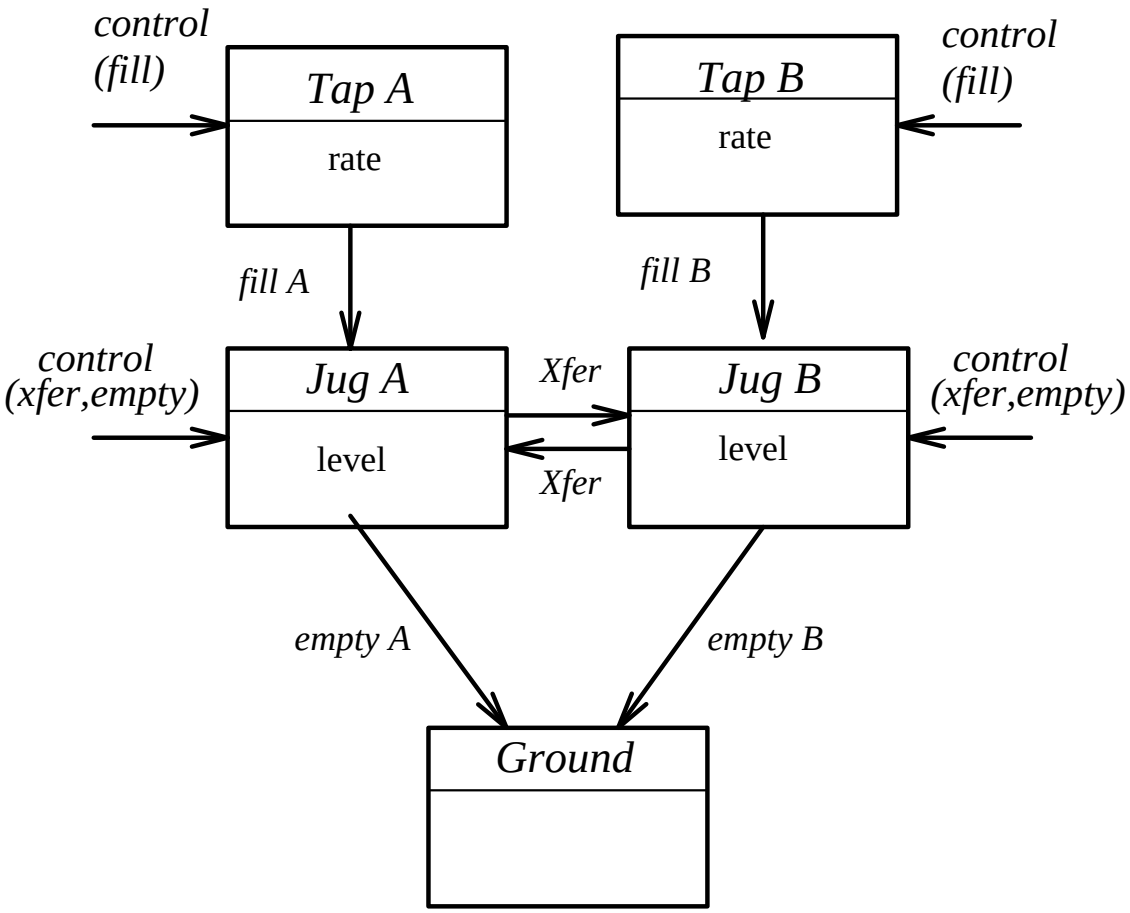


Figure 4: Instance model: two jug system.

simple metaphor provides for a whole class of models. The FSA and Markov models in simulation are the most basic declarative model types.[4] In SE, the state transition model is termed the "dynamic" or "state" model. Harel [44] extends the basic state model to form *state charts* containing embedded hierarchies of FSA levels. The use of the word "dynamic" is somewhat unusual from a simulationist's perspective since data flow diagrams are also a valid form of dynamic model representing input, transfer functions and outputs coupled within a block network diagram. On the other hand, in SE, the reference to "dynamic" reflects that state change is at the heart of dynamics. This is in agreement with systems theory and all three disciplines although memoryless functions can also represent dynamical systems. Moreover, if FSAs are embedded within another formalism (such as a data flow diagram) then that formalism is also considered dynamic. In AI, the declarative approach to system modelling is quite advanced since there are several AI declarative representations that can be useful in modelling. From AI, we can enrich the state-oriented declarative modelling approach to include logic, rules, and production systems. The declarative approach need not be limited to simple state to state transitions. For instance, in AI we often find pattern matching or constraint approaches in the unification process present within logic programming; pattern matching over a state space Q provides a convenient method of defining superstates (i.e., subsets of state space). For instance, if $Q = X_1 \times X_2$ for a two dimensional hockey rink then the goal net location could be specified by the constraint $\{(X,Y)|Y \leq 10\}$. Logic programming and, especially, constraint based programming [55, 47] typify the declarative approach to modelling using state space partitions to create superstates. The production system, constraint and logic approaches utilize unification and pattern matching to afford declarative methods the capability of representing complex behaviors with a modicum of mathematical notation.

Methods of production systems [23] and formal logic [26] (either standard or temporal) may be used as a basis for simulation modelling. We need to define the concept of state, input and time with respect to these models:

- State is defined as the current set of facts or truths in a formal system. For predicate logic this equivalences to a set of predicates. For expert systems, the rule or "knowledge" base is the state of the system.

- For production systems, inputs are known as "operators" which are executed by the controlling agent. A sequence of parameterized calls to the operators serve as the input stream that controls the system. We will associate time durations with input events by saying that whenever an input event occurs (which causes a change in state), the ensuing state will last for some specified period of time.

- Time can be assigned to each production or inference so that the process of forward chaining produces a temporal flow. We could make state variables vary continuously or discretely.

The state of the water jug model will be identified by a set of predicates. Note that predicates and arguments are both in lower case. The state is defined as (X,Y) where X is the amount of water (in gallons) in the 3 gallon jug, and Y is the amount of water in the 4

[4] In systems theory, the the FSA is sometimes termed a "local transition function."

gallon jug. We define states to vary using discrete jumps so that filling an initially empty jug A, for instance, would cause a jump from state $(0,0)$ to $(3,0)$. We will assume an initial state of $(0,0)$ (i.e. both jugs are empty). Time will be measured in minutes, therefore rates are measured in terms of gallons per minute. Filling is somewhat slow and proceeds at a rate of 2 gallons per minute. All other operations take 10 gallons per minute. There are 4 operators defined as follows:

- The operator and description.
- Conditions for the operator to be applied (i.e., fire).
- The time duration ΔT of the operator if it is applied. The duration is associated with the current state.

1. OPERATOR 1: $empty(J)$.
   (a) Empty jug $J \in \{A, B\}$.
   (b) For $J = A$, $(X, Y | X > 0) \rightarrow (0, Y)$ and $\Delta T = X/10$.
   (c) For $J = B$, $(X, Y | Y > 0) \rightarrow (X, 0)$ and $\Delta T = Y/10$.
2. OPERATOR 2: $fill(J)$.
   (a) Fill jug $J \in \{A, B\}$.
   (b) For $J = A$, $(X, Y | X < 3) \rightarrow (3, Y)$ and $\Delta T = (3 - X)/2$.
   (c) For $J = B$, $(X, Y | Y < 4) \rightarrow (X, 4)$ and $\Delta T = (4 - Y)/2$.
3. OPERATOR 3: $xfer\_all(J1, J2)$.
   (a) Transfer all water from jug $J1$ to jug $J2$.
   (b) For $J1 = A$, $(X, Y | X + Y \leq 4 \wedge X > 0 \wedge Y < 4) \rightarrow (0, X + Y)$, and $\Delta T = X/10$.
   (c) For $J1 = B$, $(X, Y | X + Y \leq 3 \wedge Y > 0 \wedge X < 3) \rightarrow (X + Y, 0)$, and $\Delta T = Y/10$.
4. OPERATOR 4: $xfer\_full(J1, J2)$.
   (a) Transfer enough water from jug $J1$ to fill $J2$.
   (b) For $J1 = A$, $(X, Y | X + Y \geq 4 \wedge X > 0 \wedge Y < 4) \rightarrow (X - (4 - Y), 4)$, and $\Delta T = (4 - Y)/10$.
   (c) For $J1 = B$, $(X, Y | X + Y \geq 3 \wedge Y > 0 \wedge X < 3) \rightarrow (3, Y - (3 - X))$, and $\Delta T = (4 - Y)/10$.

The application of various operators will change the current state to new state. For instance, given the initial system state as being $(0,0)$ we see that we can apply only operator $fill$. Specifically, we can do either $fill(A)$ or $fill(B)$. Both operations take a certain amount of time ΔT. The ΔT is associated with the current state. Let's look at the time taken to fill jug A. We note that the production rule associated with this operation is: "For $J = A$, $(X, Y | X < 3) \rightarrow (3, Y)$ and $\Delta T = (3 - X)/2$." To go from state $(0,0)$ to $(3,0)$, for instance,

will take a total time of $T = (3-0)/2$ or $T = 1.5$. This is interpreted as: "if the system is in the state $(0,0)$ and the operator *fill* is input to the system then the system will enter state $(3,0)$ in 1.5 minutes." In other words, the system remains in state $(0,0)$ for 1.5 minutes before immediately transitioning to state $(3,0)$.

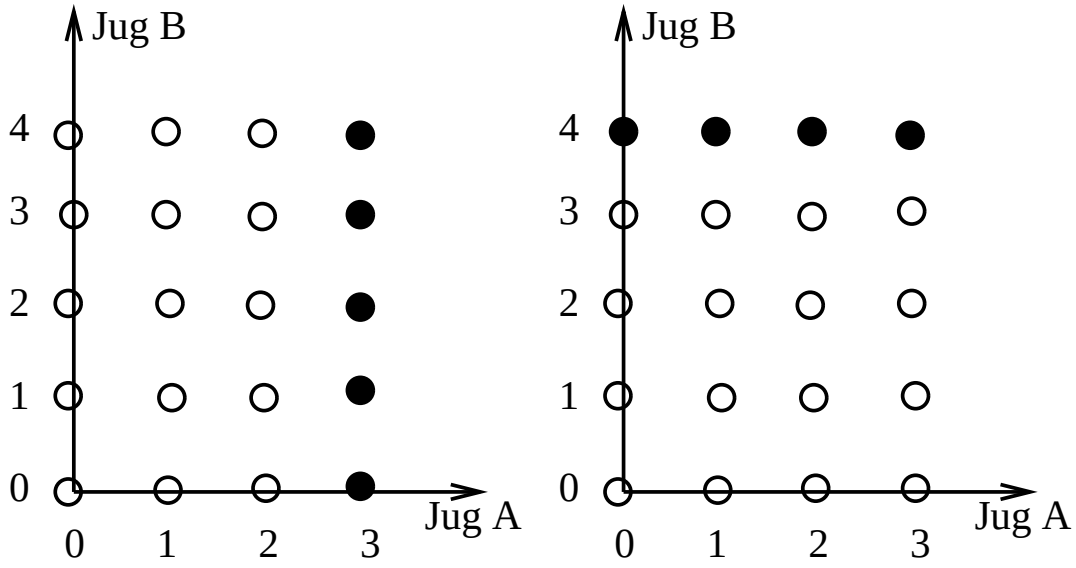
If we consider each operator as representing an external, controlling input from outside the jug system, then we can create a an FSA that represents the production system. Figure 8 displays this FSA where the following acronyms are defined:

1. Ex: Empty jug x.
2. Fx: Fill jug x.
3. $TAxy$: Transfer all water from jug x to jug y. No overflowing of water is permitted.
4. $TFxy$: Transfer all water from jug x to jug y. until jug y is full.

Even though the state space contains 20 states there are only 14 possible states in the FSA. The two jug system dynamics are quite complex; this attests, primarily, to the power of production rules (with pattern matching) over simple FSAs (without pattern matching). We can simplify the system by partitioning state or event space. There is no automatic method for state space partitioning; however, we can use a heuristic to help us: *form new states using participles created from the object model.* That is, the present participle form of "fill" is "filling." A similar approach creates the state "emptying." We can combine both the emptying state and filling state to form a stated called *In-between* or *Emptying-or-filling.* Figures 5 through 7 suggest a partitioning of state space that provides a lumped, more qualitative model. Since these segments overlap, and do not form a *minimal* partition, we must combine substates —reflecting levels in jugs A and B— to form 9 new states. The greatest "lumping" effect is contained in the state (Jug A filling, Jug B filling) where the term "filling" means neither empty nor full. It is important not to underestimate the lumping effect when considering other possible, but similar, systems. For instance, if the jugs could contain arbitrarily large amounts then the state space for fig. 8 would also be large, but the state space for the lumped model would remain the same (9 states). It is possible to further reduce the number of states by mapping and partitioning state or event space in accordance with natural language terminology; state space could easily be broken into "wet" and "dry" where dry corresponds to both jugs being empty, and wet covering the remainder of state space. Models at varying levels of abstraction [29] are created in response to the questions asked of them [37].

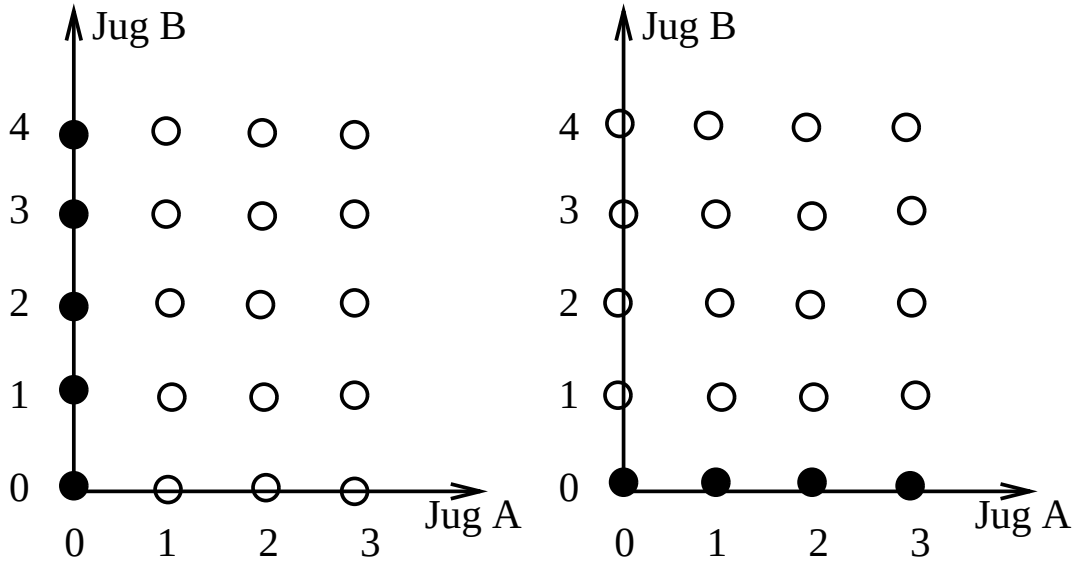
6 Functional Models

We will use a liberal interpretation of the word "functional" to include modelling approaches that stress procedural or "process-oriented" models [34]. A purely functional model is termed "memoryless" since there is no state information; so we define a functional model as one that focuses on "function" rather than state to state transition. Functional models, therefore, will often contain FSAs embedded within the definition of a functional block; the model is considered "functional" if this view dominates any subsidiary declarative representations. A



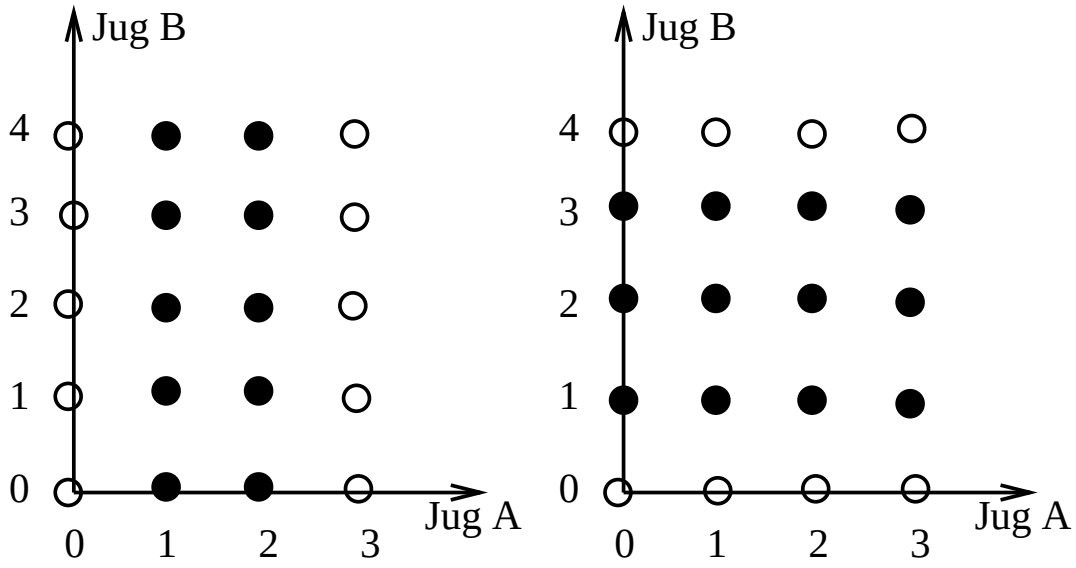
(a) Jug A full. (b) Jug B full.

Figure 5: Full phase.



(a) Jug A empty. (b) Jug B empty.

Figure 6: Empty phase.



(a) Jug A In-between. (b) Jug B In-between.

Figure 7: In-between phase.

function will be represented as a block with inputs and outputs. Inputs and outputs can represent data flows or control flows, and they are defined as such within SE [5]. From our perspective, these flows are all data flows regardless of whether a function treats its input data from a control perspective. The functional SE box structuring techniques of Mills [58] bear remarkable resemblance to those of systems and simulation theory [69, 88]. This further demonstrates a convergence in model theory between SE and simulation.

If we consider each tap and each jug to be a function then we can create a functional block model illustrated in figure 9. This dynamical model could be simulated as it is represented; that is, messages would be input to blocks that would *delay* the corresponding outputs to represent the passage of time. Use of a delay, therefore, represents a simple way to create an abstract model. If we want to increase model detail, the delay can be further decomposed into an FSA. For the tap dynamics, we use a delay; however, for the jug dynamics, we choose an FSA. The dynamics for jug A are shown in figure 10. The entire functional model shown in fig. 9 has four functional blocks with two of the blocks (jugs A and B) having an internal FSA to further refine state to state behavior. It is worthwhile to contrast this model with the declarative model in fig. 8. In the declarative model, each state represents the state of the entire system whereas, for the functional model, states are "local" to the specific function. In the object oriented sense, these states are hidden or encapsulated within the block. The aspect of locality is what has made the functional approach more acceptable to the systems community — a system is broken into functional blocks, each of which represents a physical subcomponent of the system, and each subcomponent has its own dynamics. The declarative approach of a rule-based production system —while it accurately represents the jug system

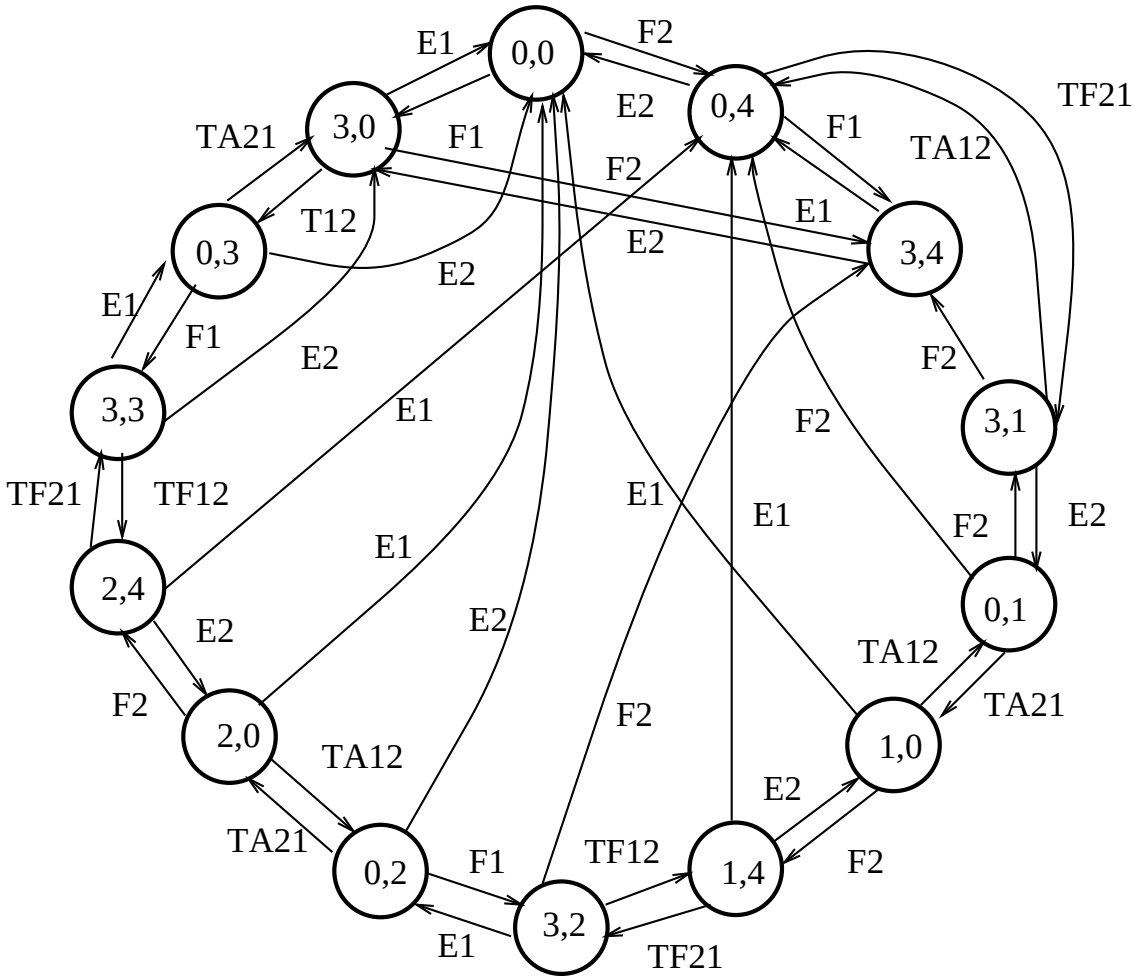


Figure 8: FSA for the jug system.

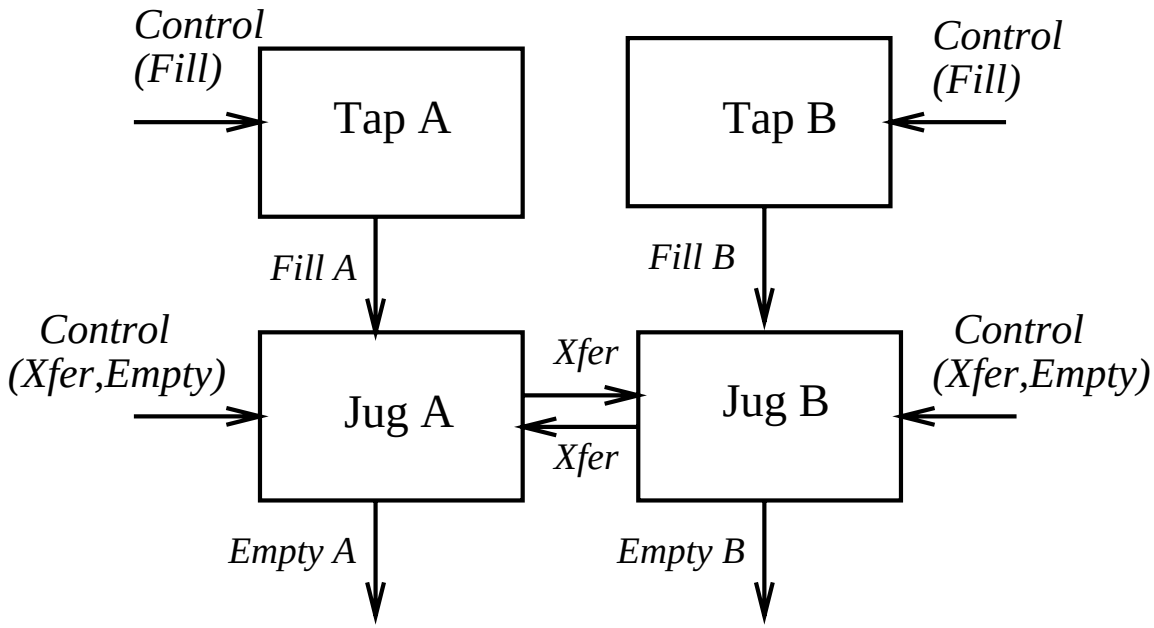


Figure 9: Functional model of two jug system.

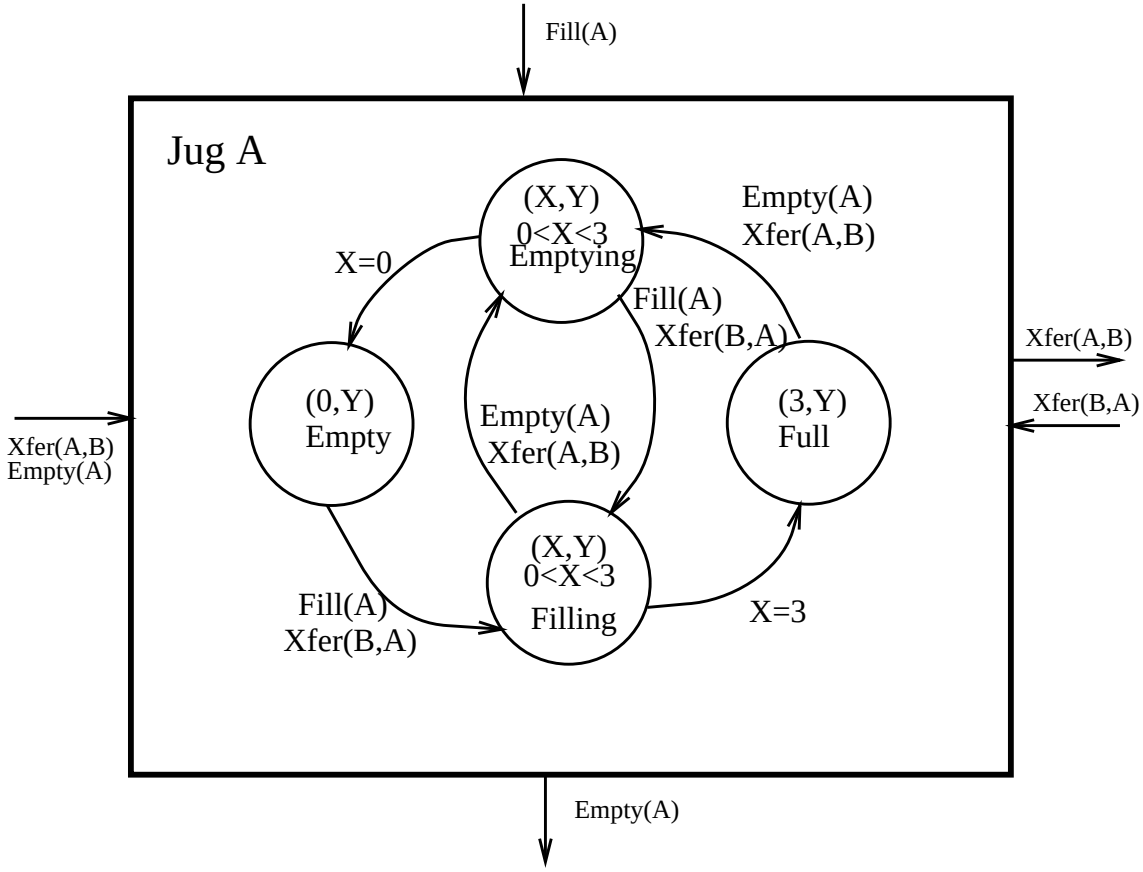


Figure 10: FSA for jug A.

dynamics— partitions state space without an accompanying separation of function. Despite these differences, there is not an inherent advantage of using the functional approach over the declarative approach since the production rule model separates behaviors according to pattern matches in state space; it is simply another way of viewing the system.

7 Multimodels

A multimodel is a model composed of other models. The purpose of a multimodel is to have a unifying system representation that contains many different levels of abstraction and perspective views within the system. Within the simulation field, the term multimodel was introduced by Oren [67] and subsequently refined by Fishwick and Zeigler [37]. Recently, AI researchers such as Addanki [2] and Forbus [39, 40] have also performed research dealing with multiple models.

Each level or segment of a multimodel can be represented by a different *type* of model; By permitting different types [33] of models, we create a more flexible modelling environment where each level is represented by its most appropriate form. In the previous section, we defined a functional model where two of the functions contained embedded models of the FSA variety; this model was a simple example of a multimodel consisting of two heterogeneous models. Let's summarize a progression of increasingly complex model forms:

- An FSA as depicted in fig. 8 is perhaps one of the simplest forms of models. The FSA is depicted graphically as a set of circles and arcs; however, their is also an implicit function box placed around the entire graph. The input to the function box is the same input referred to on the FSA arc labels; the output from the box is the same as the FSA output represented using either the Moore or Mealy [50] conventions.

- A functional block model captures system dynamics from a functional or procedural perspective rather than the declarative form of the FSA. In most functional models [69], the only representation of state is the one in the *integrator* and *delay* blocks. For instance, an integrator functional block using a single step approach keeps track of the last value for a state variable. Graphically, one can envision a functional block with a single circle representing the saved state.

- The DEVS model [90, 92] extends the standard functional notation by permitting, not only a collection of states within a function block, but also an FSA within each functional block. The semantics for the functional block now has two transition functions associated with it: transitions for input events δ_{ext} and transitions for state events δ_{int}. For this reason, the DEVS model is quite powerful as it permits more detailed semantics within the canonical functional block model.

- With respect to FSA semantics, one can view DEVS models as functional block networks where each function contains an optional FSA. We can extend this notion by permitting other models with the blocks — such as Petri nets. Moreover, we can take each state within the FSA and call it a *phase* of a more detailed process associated with the phase. This necessitates a formal partitioning of a more complex state space

as described in [33, 37], and we must also carefully define the level traversal semantics so that the transition among phases is well-defined.

Figure 11 is a multimodel that contains several layers — or abstraction levels. Note that as we proceed down the figure, we use more powerful refinment techniques. The topmost two levels represents a homogeneous refinement [33] which is common in most model languages. In this case, we have taken a single function and hierarchically decomposed it into 4 sub-functions.

The breakdown is "homogeneous" since each level uses the same type of model — functional block. The dynamics of $JugA$ is performed by an FSA. In such a model, there are external events which occur as a result of changing input and internal events which occur as a result of a time advance specification.[5] If we cannot specify a time advance then we can represent the dynamics for this state by associating yet another level of detail with the state. In this case, the state becomes a "phase" and we heterogeneously decompose the phase into a functional block model. In fig. 11 we represent the act of water filling (or emptying) by a first order differential equation $\dot{X} = U - kX$ where U is the control input and X is the continuously changing level of the water.

8 Conclusions

We demonstrated the need for a simulation modelling taxonomy to take a form more compatible with system modelling efforts in SE and AI that are related to modelling system dynamics. Currently, the terminology within the simulation field is somewhat fragmented. A declarative/functional dichotomy already exists within each discipline and, therefore, this was shown to serve as a point of convergence for all three disciplines. By presenting a common focal point, students and researchers in AI, SE and simulation can talk about dynamics using similar terminology; without a common terminological base there might be reinventing of the wheel and misunderstanding among colleagues who are in different disciplines working on the same basic system problems. We gave definitions of two views —declarative and functional— using a simple example of jug dynamics taken from a frequently cited AI problem in discrete search spaces.

Several simulation researchers have pointed out the need for further integration with SE and AI, but most simulation texts fall behind in their discussions of modelling and how it relates, for example, to system models within the object oriented design approach. Simulation has much to offer the fields of SE and AI; however, we have slanted this discussion to talk of a general integration of terminology and general taxonomic breakdown. All three fields have contributions for each other; formal studies in precisely how system models differ is a good beginning, and this has been our motivation behind the presented work. Our immediate goals are to finish a simulation modelling textbook using the integrated view, and to build a software environment that encourages a stepwise creation of models from *conceptual* through *multimodel*.

[5] In DEVS, the time specification is denoted by creating a time advance function $ta()$. A time advance value is created by analytically solving for the underlying equation representing dynamics for that state.

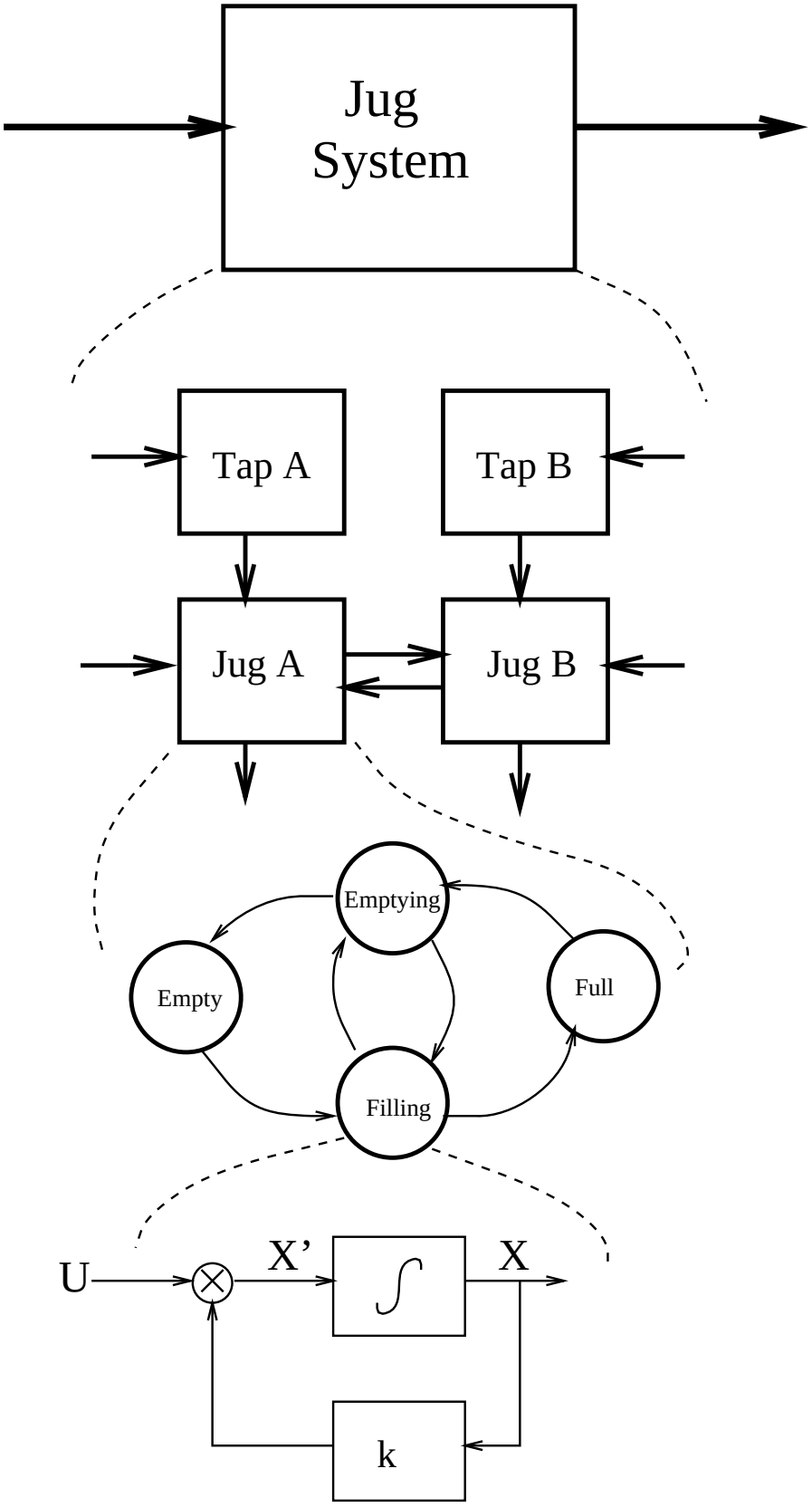


Figure 11: Heterogeneous refinement in the jug multimodel.

Acknowledgments

The motivation for writing this paper came from numerous sources including special workshops and conferences on simulation and private correspondence with many people. For four years, I participated in AI and Simulation workshops and an annual conference entitled "Artificial Intelligence, Simulation and Planning in High Autonomy Systems." These workshops in addition to the Modelling Track at the annual Winter Simulation Conference have collectively served as a springboard for issues that suggested ways in which modelling in AI, SE and simulation can relate and contribute to each another.

References

[1] Abramsky, S., and Hankin, C. *Abstract Interpretation of Declarative Languages.* Ellis Horwood Limited/John Wiley and Sons, 1987.

[2] Addanki, S., Cremonini, R., and Penberthy, J. S. Reasoning about Assumptions in Graphs of Models. In *Eleventh International Joint Conference on Artificial Intelligence* (August 1989), IJCAI, pp. 1432 – 1438.

[3] Akkermans, H. A., and Dijkum, C. v. Worlds Apart? The Modeling Cycle, Paradigms and the World View in Simulation Modeling. In *European Simulation Multi-Conference* (Nuremberg, Germany, June 1990), Society for Computer Simulation, pp. 99 – 106.

[4] Ashby, W. R. *An Introduction to Cybernetics.* John Wiley and Sons, 1963.

[5] Athena Systems. *Foresight User's Manual*, February 1989.

[6] Bailey, R. *Functional Programming with HOPE.* Ellis Horwood Limited/Simon and Schulster, 1990.

[7] Balci, O., and Nance, R. E. Simulation Model Development Environments: A Research Prototype. *Journal of the Operational Research Society 38*, 8 (1987), 753 – 763.

[8] Balci, O., Nance, R. E., Derrick, E. J., Page, E., and Bishop, J. L. Model Generation Issues in a Simulation Support Environment. In *1990 Winter Simulation Conference* (New Orleans, LA, December 1990), pp. 257 – 263.

[9] Banks, J., and Carson, J. S. *Discrete Event System Simulation.* Prentice Hall, 1984.

[10] Beck, H. W., and Fishwick, P. A. Incorporating Natural Language Descriptions into Modeling and Simulation. *Simulation Journal* (March", volume=52, number=3,pages = "102 - 109 1989).

[11] Beck, H. W., and Fishwick, P. A. Natural Language, Cognitive Models and Simulation. In *Qualitative Simulation Modeling and Analysis*, P. A. Fishwick and P. A. Luker, Eds. Springer Verlag, 1990. (in press).

[12] Bertalanffy, L. v. *General System Theory.* George Braziller, New York, 1968.

[13] Bobrow, D. G. *Qualitative Reasoning about Physical Systems.* MIT Press, 1985.

[14] Booch, G. Object-Oriented Development. *IEEE Transactions on Software Engineering 12*, 2 (Feb. 1986), 211 – 221.

[15] Booch, G. On the Concepts of Object-Oriented Design. In *Modern Software Engineering*, P. A. Ng and R. T. Yeh, Eds. Van Nostrand Reinhold, 1990, ch. 6, pp. 165 – 204.

[16] Booch, G. *Object Oriented Design.* Benjamin Cummings, 1991.

[17] Brachman, R., and Levesque, H., Eds. *Readings in Knowledge Representation.* Morgan Kaufman, 1985.

[18] Cellier, F. E. *Combined Continuous System Simulation by Use of Digital Computers: Techniques and Tools.* PhD thesis, Swiss Federal Institute of Technology Zurich, 1979.

[19] Cellier, F. E. *Continuous System Modeling.* Springer Verlag, 1991.

[20] Checkland, P. B. *Systems Thinking, Systems Science.* John Wiley and Sons, 1981.

[21] Davis, A. M. A Comparison of Techniques for the Specification of External System Behavior. *Communications of the ACM 31*, 9 (September 1988), 1098 – 1115.

[22] Davis, E. *Representations of Commonsense Knowledge.* Morgan Kaufmann, 1990.

[23] Davis, R., and King, J. An overview of production systems. *Machine Intelligence 8* (1977).

[24] Elzas, M. S., Oren, T. I., and Zeigler, B. P. *Modelling and Simulation Methodology in the Artificial Intelligence Era.* North Holland, 1986.

[25] Elzas, M. S., Oren, T. I., and Zeigler, B. P. *Modelling and Simulation Methodology: Knowledge Systems' Paradigms.* North Holland, 1989.

[26] Enderton, H. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[27] Findler, N. V., Ed. *Associate Networks: Representation and Use of Knowledge By Computers.* Academic Press, 1979.

[28] Fishwick, P. A. *Hierarchical Reasoning: Simulating Complex Processes over Multiple Levels of Abstraction.* PhD thesis, University of Pennsylvania, 1986.

[29] Fishwick, P. A. The Role of Process Abstraction in Simulation. *IEEE Transactions on Systems, Man and Cybernetics 18*, 1 (January/February 1988), 18 – 39.

[30] Fishwick, P. A. Qualitative Methodology in Simulation Model Engineering. *Simulation Journal 52*, 3 (March 1989), 95 – 101.

[31] Fishwick, P. A. Studying how Models Evolve: An Emphasis on Simulation Model Engineering. In *Advances in AI and Simulation* (Tampa, FL, 1989), pp. 74 – 79.

[32] Fishwick, P. A. Toward an Integrated Approach to Simulation Model Engineering. *International Journal of General Systems 17*, 1 (May 1990), 1 – 19.

[33] Fishwick, P. A. Heterogeneous Decomposition and Coupling for Combined Modeling. In *1991 Winter Simulation Conference* (Phoenix, AZ, December 1991), pp. 1199 – 1208.

[34] Fishwick, P. A. A Functional/Declarative Dichotomy for Characterizing Simulation Models. In *AI, Simulation and Planning in High Autonomy Systems* (Perth, Australia, 1992), IEEE Computer Society Press, pp. 102 – 109.

[35] Fishwick, P. A. *Computer Simulation Modeling: Methodology, Algorithms and Programs.* 1992. (to be published as a textbook in early 1993).

[36] Fishwick, P. A., and Modjeski, R. B., Eds. *Knowledge Based Simulation: Methodology and Application.* Springer Verlag, 1991.

[37] Fishwick, P. A., and Zeigler, B. P. A Multimodel Methodology for Qualitative Model Engineering. *ACM Transactions on Modeling and Computer Simulation 2*, 1 (1992).

[38] Flood, R. L., and Carson, E. R. *Dealing with Complexity: An Introduction to the Theory and Application of Systems Science.* Plenum Press, 1988.

[39] Forbus, K. Qualitative Physics: Past, Present and Future. In *Exploring Artificial Intelligence*, H. Shrobe, Ed. Morgan Kaufmann, 1988, pp. 239 –296.

[40] Forbus, K. D., and Falkenhainer, B. Self-Explanatory Simulations: An Integration of Qualitative and Quantitative Knowledge. In *AAAI* (1990), pp. 380 – 387.

[41] Forrester, J. W. *Urban Dynamics.* MIT Press, Cambridge, MA, 1969.

[42] Forrester, J. W. *World Dynamics.* Wright-Allen Press, 1971.

[43] Futo, I., and Gergely, T. *Artificial Intelligence in Simulation.* Ellis Horwood Limited/John Wiley and Sons, 1990.

[44] Harel, D. On Visual Formalisms. *Communications of the ACM 31*, 5 (May 1988), 514 – 530.

[45] Harel, D. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering 16*, 3 (April 1990), 403 – 414.

[46] Harel, D. Biting the Silver Bullet: Toward a Brighter Future for System Development. *IEEE Computer 25*, 1 (January 1992), 8 – 20.

[47] Heintze, N., Jaffar, J., Michaylov, S., Stuckey, P., and Yap, R. *The CLP($\mathcal{R}$) Programmer's Manual: Version 1.1*, November 1991.

[48] Henderson, P. *Functional Programming: Application and Implementation.* Prentice Hall International, 1980.

[49] Hogger, C. J. *Essentials of Logic Programming.* Oxford University Press, 1990.

[50] Hopcroft, J. E., and Ullman, J. D. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 1979.

[51] Kalman, R. E., Falb, P. L., and Arbib, M. A. *Topics in Mathematical Systems Theory.* McGraw-Hill, New York, 1962.

[52] Klir, G. J. *Architecture of Systems Problem Solving.* Plenum Press, 1985.

[53] Kowalski, R. *Logic for Problem Solving.* Elsevier North Holland, 1979.

[54] Law, A. M., and Kelton, D. W. *Simulation Modeling & Analysis.* McGraw Hill, 1991. Second edition.

[55] Leler, W. *Constraint Programming Languages: Their Specification and Generation.* Addison Wesley, 1988.

[56] Miller, D. P., Rothenberg, J., Franke, D. W., Fishwick, P. A., and Firby, R. J. AI: What Simulationists Really Need to Know. In *1990 Winter Simulation Conference* (New Orleans, LA, December 1990), pp. 204 – 209.

[57] Miller, V. T., and Fishwick, P. A. Heterogeneous Hierarchical Models. In *Artificial Intelligence X: Knowledge Based Systems* (Orlando, FL, April 1992), SPIE.

[58] Mills, H. D. Stepwise Refinement and Verification in Box-Structured Systems. *IEEE Computer 21*, 6 (June 1988), 23 – 36.

[59] Nance, R. E. The Time and State Relationships in Simulation Modeling. *Communications of the ACM 24*, 4 (April 1981), 173 – 179.

[60] Nance, R. E. A Conical Methodology: A Framework for Simulation Model Development. In *Conference on Methodology and Validation* (San Diego, CA., April 1987), Society for Computer Simulation, pp. 38 – 43.

[61] Nance, R. E. Modeling and Programming: An Evolutionary Convergence, April 1988. Unpublished overheads requested from author.

[62] Narain, S., and Rothenberg, J. Qualitative modeling using the causality relation. *Transactions of the Society for Computer Simulation 7*, 3 (1990), 265 – 289.

[63] Nielsen, N. R. Applications of AI Techniques to Simulation. In *Knowledge Based Simulation: Methodology and Application*, P. Fishwick and R. Modjeski, Eds. Springer Verlag, 1991, pp. 1 – 19.

[64] O'Keefe, R. M. The Role of Artificial Intelligence in Discrete Event Simulation. In *Artificial Intelligence, Simulation & Modeling*, L. E. Widman, K. A. Loparo, and N. R. Nielsen, Eds. John Wiley and Sons, 1989, pp. 359 – 379.

[65] Oren, T. I. Model-Based Activities: A Paradigm Shift. In *Simulation and Model-Based Methodologies: An Integrative View*, T. I. Oren, B. P. Zeigler, and E. M. S., Eds. Springer Verlag, 1984, pp. 3 – 40.

[66] Oren, T. I. Simulation: Taxonomy. In *Systems and Control Encyclopedia*, M. G. Singh, Ed. Pergammon Press, 1987, pp. 4411 – 4414.

[67] Oren, T. I. Dynamic Templates and Semantic Rules for Simulation Advisors and Certifiers. In *Knowledge Based Simulation: Methodology and Application*, P. Fishwick and R. Modjeski, Eds. Springer Verlag, 1991, pp. 53 – 76.

[68] Overstreet, C. M., and Nance, R. E. A Specification Language to Assist in Analysis of Discrete Event Simulation Models. *Communications of the ACM 28*, 2 (February 1985), 190 – 201.

[69] Padulo, L., and Arbib, M. A. *Systems Theory: A Unified State Space Approach to Continuous and Discrete Systems.* W. B. Saunders, Philadelphia, PA, 1974.

[70] Peterson, J. L. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981.

[71] Praehofer, H. *System Theoretic Foundations for Combined Discrete-Continuous System Simulation.* PhD thesis, Johannes Kepler University of Linz, 1991.

[72] Pressman, R. S. *Software Engineering: A Practitioner's Approach.* McGraw Hill, 1992.

[73] Rich, E., and Knight, K. *Artificial Intelligence.* McGraw-Hill, 1991.

[74] Richardson, G. P., and Pugh, A. L. *Introduction to System Dynamics Modeling with DYNAMO.* MIT Press, Cambridge, MA, 1981.

[75] Roberts, N., Andersen, D., Deal, R., Garet, M., and Shaffer, W. *Introduction to Computer Simulation: A Systems Dynamics Approach.* Addison-Wesley, 1983.

[76] Rothenberg, J. Object-Oriented Simulation: Where do we go from here? Tech. rep., RAND Corporation, October 1989.

[77] Rothenberg, J. Knowledge-Based Simulation at the RAND Corporation. In *Knowledge Based Simulation: Methodology and Application*, P. Fishwick and R. Modjeski, Eds. Springer Verlag, 1991, pp. 133 – 161.

[78] Rozenblit, J. W., and Zeigler, B. P. Knowledge-Based Simulation and Design Methodology: A Flexible Test Architecture Application. *Transactions of the Society for Computer Simulation 7*, 3 (1990).

[79] Rumbaugh, J., Blaha, M., Premerlani, W., Frederick, E., and Lorenson, W. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[80] Shannon, R. E. *Systems Simulation: The Art and Science.* Prentice Hall, 1975.

[81] Sowa, J. F. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, 1984.

[82] Weld, D. S. The Use of Aggregation in Causal Simulation. *Artificial Intelligence 30*, 1 (October 1986), 1 – 34.

[83] Weld, D. S., and DeKleer, J. *Readings in Qualitative Reasoning about Physical Systems.* Morgan Kaufmann, 1990.

[84] Widman, L. E., Loparo, K. A., and Nielsen, N. R. *Artificial Intelligence, Simulation and Modeling.* John Wiley and Sons, 1989.

[85] Winograd, T. Frame Representations and the Declarative/Procedural Controversy. In *Representation and Understanding*, D. Bobrow and A. Collins, Eds. Academic Press, 1975, pp. 185 – 210.

[86] Woods, W. A. What's in a Link: Foundations for Semantic Networks. In *Representation and Understanding*, D. Bobrow and A. Collins, Eds. Academic Press, 1975, pp. 35 – 82.

[87] Wymore, A. W. *A Mathematical Theory of Systems Engineering: The Elements.* Krieger Publishing Co., 1977.

[88] Zeigler, B. P. *Theory of Modelling and Simulation.* John Wiley and Sons, 1976.

[89] Zeigler, B. P. *Multi-Facetted Modelling and Discrete Event Simulation.* Academic Press, 1984.

[90] Zeigler, B. P. Multifaceted Systems Modeling: Structure and Behavior at a Multiplicity of Levels. In *Individual Development and Social Change: Explanatory Analysis.* Academic Press, 1985, pp. 265 – 293.

[91] Zeigler, B. P. DEVS Representation of Dynamical Systems: Event-Based Intelligent Control. *Proceedings of the IEEE 77*, 1 (January 1989), 72 – 80.

[92] Zeigler, B. P. *Object Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems.* Academic Press, 1990.