A Functional/Declarative Dichotomy for Characterizing Simulation Models

P. A. Fishwick
Department of Computer and Information Sciences
University of Florida, Bldg CSE, Room 301
Gainesville, FL. 32611

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 to our own relating specifically to modelling dynamic systems. We present a perspective that serves to characterize simulation models in terms of their procedural versus declarative orientations. In teaching simulation students using this perspective, we have had success in relating the field of modelling within computer simulation to other sub-disciplines within computer science.

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, we find it tempting to confuse the method of modelling such as *process orientation* (i.e., a functional modelling approach) with the method of simulating or executing a model such as *event scheduling* (i.e., an approach of simulating parallelism of an abstract, lumped model on a sequential computer).

Our emphasis is on *modelling methodology* [7, 6] rather than *analysis methodology*. Methodology in analysis has received a much more comprehensive treatment in the general simulation literature [17, 2] as compared with methodology in modelling. The art and science of modelling, per se, is more akin to 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 and SE. 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." For example, the data flow diagram in SE and the block model in simulation 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, modeled— 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 valuable for a great number of models. In simulation, we should also use this functional category of modelling to represent queuing networks, digital logic circuits and systems for control

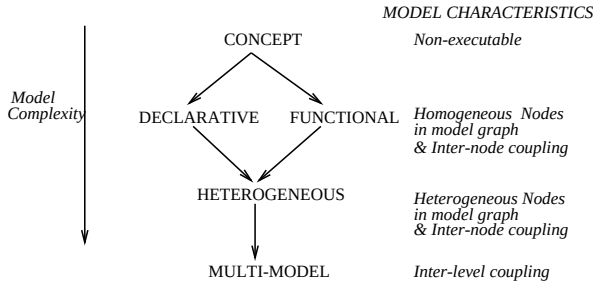


Figure 1: The proposed modelling paradigm.

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 haved 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 traditional 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 [31]. 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. One may combine these forms to form *heterogeneous* models where model graph nodes may be of different types. Finally, the *multimodel* [13, 18, 15] is the most comprehensive type of model that supports multiple models tied together with homomorphic mappings from one model to another. 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 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 [26, 4], 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 the class itself. Our approach is to stress the utility of having *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; however, it has not yet penetrated the general textbook simulation 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 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 new 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 [25] and block models, for instance have functional orientations. The term "dichotomy" is used somewhat loosely, though, since there exist some kinds of modeling methods such as Petri nets [23] that have equal shares of declarative (i.e., place) and functional (i.e., transition) subrepresentations.

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. 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 in-

[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 *simulation* — and especially with *modelling*— the differences are not that pronounced.

complete knowledge about the domain. 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 texts have appeared [28, 14]. Simulation models have characteristically been composed of simple entities and objects; however, the introduction of autonomous agents (including robots and humans) 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 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 sometimes shaped (or controlled) through engineering. 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 an environment. The fields of qualitative reasoning and qualitative physics [27, 3] 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 [29]; 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[27, 5].

2.2 SE & Simulation Models

Software engineers are pioneering new and novel methods for computer assisted methods; tools such as Foresight [1] provide the modeller with the capability of modelling dynamic systems using finite state automata and block modelling all under the umbrella of object organization. It appears as if software engineers are developing a keen interest in simulation; there is an apparent convergence between these two areas [21, 20]. Some of our previous research [11, 10, 12, 15] has suggested the study of *model engineering* as a direct analog to software engineering. The key to the convergence between the SE and simulation fields lies in the area of distributed and real-time 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. Zeigler presents a theory for modelling autonomous agents [32] while implementing a model engineering methodology. 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 interbank transaction project, for instance, is also highly concerned with the efficiency of the entire 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.

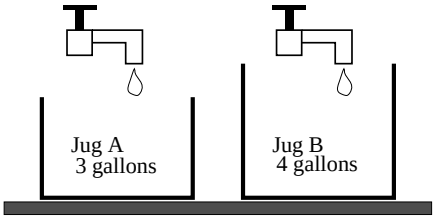


Figure 2: Two jug system.

3 The Two Jug System

Consider the water jug problem as described in AI[16, 24]. In the water jug problem, there are two water jugs (one with a four gallon capacity, and the other with three gallons). Jug A is the three gallon jug, and jug B is the four gallon jug (see figure 2). There are two water spigots and there are 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. In addition to operations, it is common, in this problem, to specify a goal that we want to obtain the sequence of operations that will yield 2 gallons in Jug A. We will study the system as a whole without focusing on the discrete optimization aspect of reaching a particular goal.

4 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 simple metaphor provides for a whole class of models. The FSA and Markov model in simulation are the most basic model types.[2] In SE, the state transition model is termed the "dynamic" or "state" model. 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. Even though this is true for the most part, memoryless functions 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 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. If a state matches a certain structure then we delineate in our model the transition to the next state. We can, in fact, use pattern matching capabilities so that state "pattern structures" are used instead of state "instantiations" with built-in constant values. The production system and logic approach utilize this form of calculation to afford declarative methods the capability of representing complex behaviors with a modicum of mathematical notation.

Methods of production systems [5] and formal logic [8] (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 (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." 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 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. We will define them as follows:

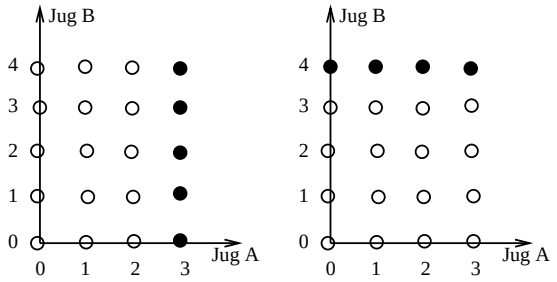
- The operator and description.

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

- 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 $(3, 0)$.



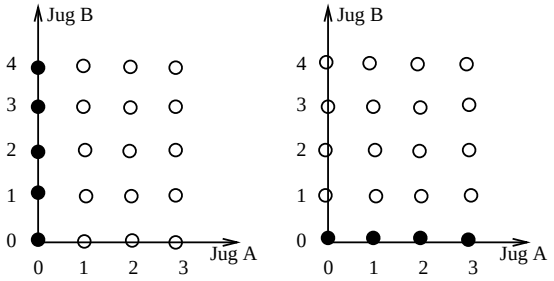
(a) Jug A full (b) Jug B full

Figure 3: Full phase.

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 6 displays this FSA where the following acronyms are defined:

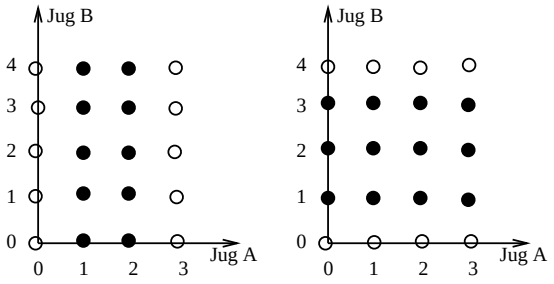
1. Ex: Empty jug x.
2. Fx: Fill jug x.
3. $TAxy$: Transfer all water from jug x to jug y. No overflowing of water is permitted.
4. $TFxy$: Transfer all water from jug x to jug y. until jug y is full.

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 participle created from the object model.* That is, the present participle form of "fill" is "filling." Figures 3 through 5 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



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

Figure 4: Empty phase.



(a) Jug A filling (b) Jug B filling

Figure 5: Filling phase.

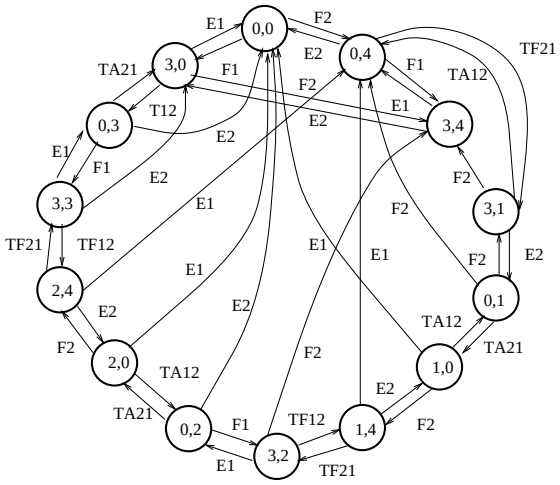


Figure 6: FSA for the jug system.

contain arbitrarily large amounts then the state space for fig. 6 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 [9] are created in response to the questions asked of them [15].

5 Functional Models

We will use a liberal interpretation of the word "functional" to include modelling approaches that stress procedural or "process-oriented" models. 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 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 [1]. 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 [19] bear remarkable resemblance to those of systems and simulation theory [22, 30]. This further demonstrates a convergence in model theory between SE and simulation.

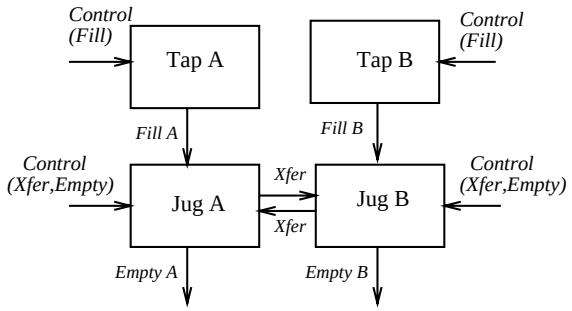


Figure 7: Functional model of two jug system.

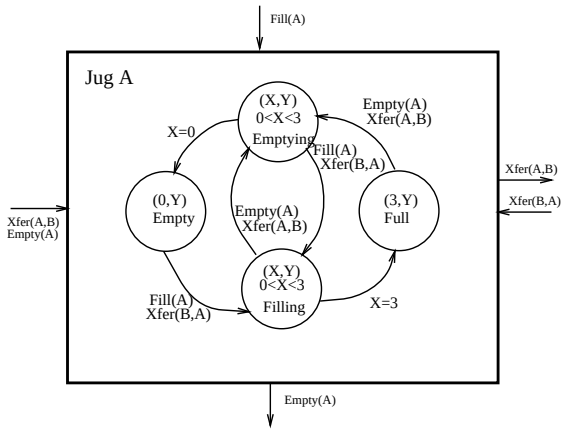


Figure 8: FSA for jug A.

If we consider each tap and each jug to be a function then we can create a functional block model illustrated in figure 7. 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 8. The entire functional model shown in fig. 7 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. 6. 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 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.

6 Conclusions

Our purpose was primarily to demonstrate the need for simulation modelling taxonomy to take a form more compatible with system modeling efforts in SE and AI. We have demonstrated how a declarative/functional dichotomy already exists within each discipline, and we gave definitions of the two views using a simple example of dynamics — the jug problem. 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 much to offer each other; formal studies in precisely how system models differ is a good beginning, and this has been our motivation behind the presented work. We are presently extending this study to delineate two other key classes of model: the *concept model* and the *multimodel*.

References

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

[2] Jerry Banks and John S. Carson. *Discrete Event System Simulation*. Prentice Hall, 1984.

[3] Daniel G. Bobrow. *Qualitative Reasoning about Physical Systems*. MIT Press, 1985.

[4] Grady Booch. *Object Oriented Design*. Benjamin Cummings, 1991.

[5] Ernest Davis. *Representations of Commonsense Knowledge*. Morgan Kaufmann, 1990.

[6] Maurice S. Elzas, Tuncer I. Oren, and Bernard P. Zeigler. *Modelling and Simulation Methodology*

in the Artificial Intelligence Era. North Holland, 1986.

[7] Maurice S. Elzas, Tuncer I. Oren, and Bernard P. Zeigler. *Modelling and Simulation Methodology: Knowledge Systems' Paradigms*. North Holland, 1989.

[8] Herbert Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

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

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

[11] Paul A. Fishwick. Studying how Models Evolve: An Emphasis on Simulation Model Engineering. In *Advances in AI and Simulation*, pages 74 – 79, Tampa, FL, 1989.

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

[13] Paul A. Fishwick. Heterogeneous Decomposition and Coupling for Combined Modeling. In *1991 Winter Simulation Conference*, pages 1199 – 1208, Phoenix, AZ, December 1991.

[14] Paul A. Fishwick and Richard B. Modjeski, editors. *Knowledge Based Simulation: Methodology and Application*. Springer Verlag, 1991.

[15] Paul A. Fishwick and Bernard P. Zeigler. A Multimodel Methodology for Qualitative Model Engineering. *ACM Transactions on Modeling and Computer Simulation*, 1992. Submitted for review.

[16] Robert Kowalski. *Logic for Problem Solving*. Elsevier North Holland, 1979.

[17] Averill M. Law and David W. Kelton. *Simulation Modeling & Analysis*. McGraw Hill, 1991. Second edition.

[18] Victor T. Miller and Paul A. Fishwick. Heterogeneous Hierarchical Models. In *Artificial Intelligence X: Knowledge Based Systems*, Orlando, FL, April 1992. SPIE.

[19] Harlan D. Mills. Stepwise refinement and verification in box-structured systems. *IEEE Computer*, 21(6):23 – 36, June 1988.

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

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

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

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

[24] Elaine Rich and Kevin Knight. *Artificial Intelligence*. McGraw-Hill, 1991.

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

[26] James Rumbaugh, Michael Blaha, William Premerlani, Eddy Frederick, and William Lorenson. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[27] Daniel S. Weld and Johann DeKleer. *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, 1990.

[28] Lawrence E. Widman, Kenneth A. Loparo, and Norman R. Nielsen. *Artificial Intelligence, Simulation and Modeling*. John Wiley, 1989.

[29] Terry Winograd. Frame Representations and the Declarative/Procedural Controversy. In Daniel Bobrow and Allan Collins, editors, *Representation and Understanding*. Academic Press, 1975.

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

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

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