Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, eds

A TUTORIAL ON SIMULATION MODELING IN SIX DIMENSIONS

Paul A. Fishwick

University of Florida
Computer and Information Science & Engineering Department
Bldg. CSE, Room 301
Gainesville, Florida, USA

ABSTRACT

Simulation involves modeling and analysis of real-world systems. This tutorial will provide a broad overview of the modeling practice within simulation by introducing the reader to modeling choices found using six dimensions: *abstraction*, *complexity*, *culture*, *engineering*, *environment*, and *process*. Modeling can be a daunting task even for the seasoned modeling and simulation professional, and so my goal is to introduce modeling in two ways: 1) to use one specific type of model (Petri Net) as an anchor for cross-dimensional discussion, and 2) to provide a follow up discussion, with additional non Petri Net examples, to clarify the extent of each dimension. For example, in the abstraction dimension, one must think about scale, refinement, and hierarchy when modeling regardless of the type of modeling language. The reader will come away with a broad framework within which to understand the possibilities of models and of modeling within the practice of simulation.

1 INTRODUCTION

The title of this paper is a little odd because it may sound as if I am referring to "six dimensional objects or space" as one might have in physical theories. However, the word *dimension* is used as a means to categorize important aspects of the modeling practice (Fishwick 1995). Models, for instance, can be abstract or they can be detailed. This observation is true regardless of the model type. Abstraction is a universal quality of objects and their behaviors. By thinking about abstraction, one is able to become a better modeler. There is also a danger of being too general. If I provide definitions for six dimensions, you may be at wits end to come away with a good sense of modeling. So, I will provide definitions for each of the dimensions, but also provide discussion grounded on one model type (a Petri net) for each dimension. Then, I'll employ various other examples for each dimension for a more thorough explanation. By moving from a standard Petri net model type to a free-form discussion, the dimensions can be better understood. Many of the references are from prior Winter Simulation Conference proceedings in the tutorial sections so that the reader can easily link to these freely available online proceedings papers.

Before proceeding with the task of illustrating models, let's briefly define modeling. *Modeling is the task of building a model, with a model consisting of a product containing features that emphasize certain attributes of a phenomenon.* Modeling is so natural to humans that we do it routinely. A map is a model of territory—by taking a map on a journey, we can find our way around. The map serves to highlight, or surface, certain aspects of the territory. A toy train is a scaled down model of a train. Such scale models are often specified by their numerical scale: 1:10 or 1:100 for instance, with a 1:10 model being $1/10^{th}$ the size of the real train. Figure 1 shows a scale model of a favorite destination of mine, Bourton on the Water, which is a small village in Gloucestershire, England. You visit Bourton, park your car, and then locate the model village, which is the model of Bourton (Figure 1, left figure). Interestingly, since this model is

of the village, it must also include a model inside of it—a recursive property of having the model inside of that which is being modeled. The right side of Figure 1 shows this recursion.

Figure 1: A camera shot of the street view inside of the model of Bourton-on-the-water (left), and evidence of a smaller model village inside of the large model village, which is inside of the town. This recursion can be practically constructed only so many levels deep, and to a detailed degree

You come to understand the essence of modeling through such examples. This is not what we would call a simulation model; however, simulation modeling in all of its complexity and variety is fundamentally no different than map-reading and walking through the model village. Maps provide relative distances without having to walk or drive, and the model village does the same but also provides a bit of immersion and sense of presence albeit in a Gulliver sort of way.

In simulation, the focus of the Winter Simulation Conference, rather than highlight relative spatial positions and orientations, the goal is to surface physical *aspects of behavior* for the thing being modeled. Behavior, and more generally dynamics, is a little more difficult to grasp in terms of thinking about models. For a model to capture aspects of behavior, it has to surface how things change in an object. For example, if I wanted to model a light switch, I could do so with two adjacent ceramic tiles. I would say that when your feet are on one tile, the switch is "on" and then you move to the adjacent tile, the switch if "off." The tiles serve as a dynamic model of the light switch. Simulation models often do not look like that which they model since the ceramic tiles do not look like the light switch—the tiles capture a dynamical relationship, which we refer to as a *state change*.

We are going to illustrate simulation models using six dimensions: *abstraction*, *complexity*, *culture*, *engineering*, *environment*, and *process*. Why only six dimensions, or do we even need six of them? There is no magic to the choice of the number six. My purpose was to choose a small, but sufficient, number of dimensions so that you can begin to explore the wonderful world of modeling. The models that I am going to show you, and discuss, may not all look as entertaining as the model village in Bourton, but they are just as fun to learn about. Some of the simulation models actually have the potential to look not unlike scale models, but we'll come to that shortly.

2 PETRI NETS

Carl Adam Petri formally defined networks named after him as Petri Nets in 1962 starting with his PhD thesis (Petri 1962) and more recently covered in tutorial form (Gehlot and Nigro 2010). The Petri net is being chosen to illustrate the six dimensions of modeling. There are many other model types that could have been chosen, but Petri nets are in wide use and will serve as a good exemplar. The Petri net falls under the category of bipartite directed graph, which is to say that the Petri net is a network (i.e., graph in mathematical parlance) containing two types of nodes (a place and a transition). Places and transitions are connected together to form a network. Apart from the mathematical formalism, Petri was instrumental in

creating diagrams of his modeling method and similar diagrams are in use today. Typically, circles are used to depict places, and rectangles or solid lines for transitions. Figure 2 shows three nets, each at two different stages in time.

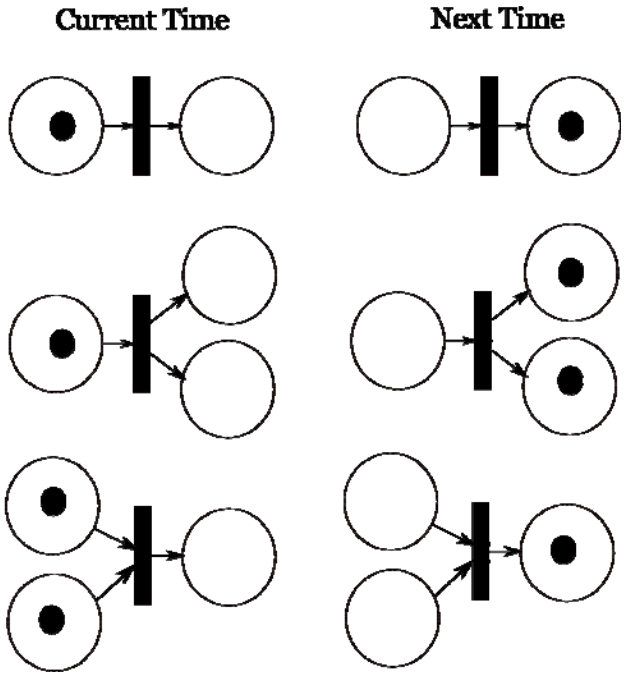

Figure 2: Three Petri nets with their diagrams illustrating the current time and the subsequent time.

Let's begin with the top left subfigure within Figure 2. There are two circles separated by a solid black rectangle oriented vertically. This subfigure is a Petri net composed of two places and one transition. The place on the left has a small black-filled circle called a token. Tokens move through Petri nets according to a rule:

> *For each instant of time, each transition is checked for firing (i.e., processing). A transition can fire only if there is at least one token in each input place for that transition. When a transition fires, one token is removed from each input place, and added to each output place.*

This is a simple rule, and yet governs the entire simulation of a Petri net. Given this rule we can simulate the top-left net by removing the token on from the left place and putting it in the right place at the next time step. Think of the token as moving through the transition from the left place to the right place. It is a discrete sort of motion since there is no in-between condition where the token is hovering over the transition or sitting on one of the arrows. The remaining two examples in Figure 2 show other possibilities that conform to the above rule. In the second row, we see that the token is multiplied in the output places, and in the third row, we see that both tokens are required for the transition to fire, not unlike a synapse in neural circuitry.

At this point, let's cover some key abstract concepts associated with any simulation model, such as Petri nets. Simulation models involve passage of time. This passage is logical rather than physical since the next time Petri net proceed from the current time using any delta time that we care to employ. We could make the tokens move around slowly in real time, or very quickly. The simulation time is, therefore, different from real time unless we wish to synchronize them. *Time* is a major concept for simulation

modeling since we have to configure our simulations to advance time in ways that match our goals for the physical system being modeled. Two other concepts are *state* and *event*. Each Petri net in Figure 2 has a state, defined as the collective number of tokens in each place. When considering the last (third) row, the Petri net starts in state (1,1,0) and moves at the next simulated time to state (0,0,1). A zero (0) means that there are no tokens, and a one (1) means that there is one token. An event is a change in state. Think of an event as a state at one point in time. When a net transition fires, that is an event. Often, an event is denoted with a name and a time. So, (FIRE-1,3) is an event that states that the event is the firing of transition number one at simulated time 3.

3 DIMENSION 1: ABSTRACTION

Abstraction is central to simulation and modeling, and for just about everything else in language, science, and the arts. When thinking about abstraction, I suggest that you consider abstraction as defined by *a one to many mapping* of concepts or entities. The concept of "house" maps to all of the actual houses in existence, and yet I can use that word as a category to represent all house instances. This represents a mapping between the concept word "house" and all houses. Figure 3 shows a Petri net that represents an abstract representation of a manufacturing cell (e.g., workstation) where a pick-and-place robot arm feeds a part into a drill.

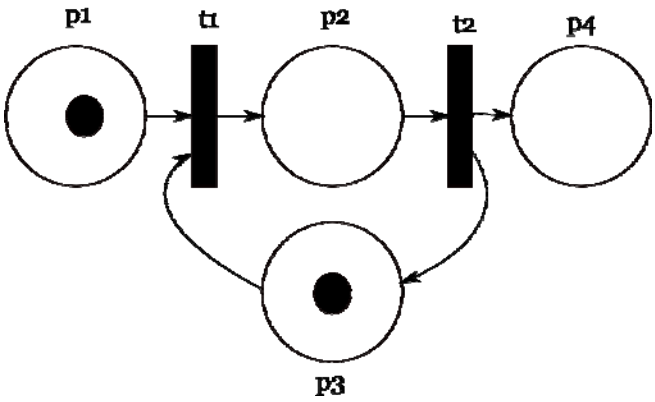

Figure 3: A Petri net for a robot arm/drill synchronized activity

You may be noticing that this doesn't look like a robot arm or a drill, and that is an aspect of abstraction. The Petri net could just as easily be modeling packet transfers into an internet router and it would look the same, so the Petri net serves to abstract away the geometry and identities of the real-world objects so that we can focus on the process. The mapping for the manufacturing example begins with an unprocessed part designated by the token in p1. Transition t1 is the robot arm and t2 is the drill. When the drill is finished, and when the part is ready for drilling, the drilling can proceed according to our Petri net simulation rule.

Abstraction is not directly related to the look of the model. You might think that Figure 3 is abstract because there are bare circles and simple icons. The important thing, in abstraction, is that there is a one to many relationship between model component and real-world entities. Given this logic, a console game that employed metal balls bumping into pressure plates, representing transitions, would also be abstract since the metal balls could refer to manufactured parts, internet packets, or people navigating a theme park. Humans are well adapted to imagining, through role play and analogy that model components, whether metal balls or circles, can represent real world objects.

Scale is often a means for understanding abstraction since it reflects a one to many mapping. For example, once when giving a lecture on a compartmental modeling used for cardiovascular performance (Ezzell et al. 2011), I was asked why we used compartmental models (Walter and Contreras 1998) vs.

CFD (computational fluid dynamics). This was a good question, and one that we must ask ourselves as we model. What level of scale do we need to model systems? What degree of detail is required? For the question on compartmental models vs. CFD, one has to look at the goals of the simulation and whether questions of scale have the potential to affect simulation output that might sway the subsequent analysis. For the human heart, if the main objectives are to study the effects of hypovolemic shock (e.g., from blood loss), for example, then modeling blood volume and pressure over time with differential equations and compartmental models is sufficient. However, if we had decided that we wanted to study the effects of a new artificial heart valve, that would have required making space an independent variable. For many simulations, time is the only independent variable; however, in some cases, space must also be carefully considered. There are no generally available heuristic decision procedures that specify what model types to use, unfortunately, relegating modeling to an art. There are areas in ecology where matters of abstraction play a role in individually-based models. In physics, matters of scale may surface where we use solid body mechanics, plastic deformation, or molecular dynamics. Agent-based approaches to modeling are an example of choosing to model through multiple entities (Chan et al. 2010) rather than something more abstract. The model type depends on the level of abstraction required to answer research questions about the system. Figure 4 shows one way of thinking of abstraction for Figure 3.

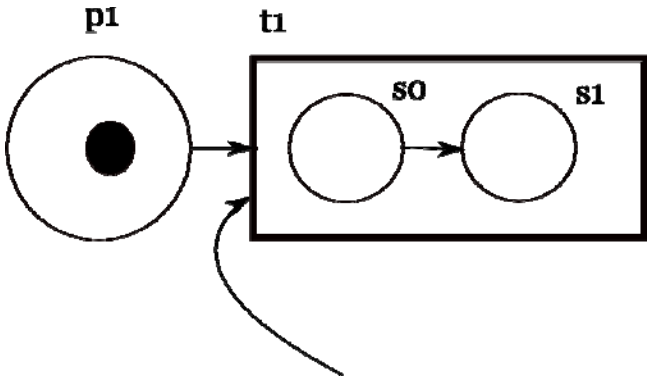

Figure 4: A part of the Petri net in Figure 3 where transition t1 is refined as a state machine

In our hypothetical manufacturing example, t1 was the operation of the robot arm. If the arm performs two operations, pick and place, then we might refine t1 using a finite state machine defined with two states (s0 and s1) with s0 being the pick operation, and s1 being the place operation. Figure 4 illustrates abstraction as a consideration of scale. As we "zoom in" to t1, we find more details, as we would when employing a microscope. The transition t1 in Figure 3 is an abstraction of several possible submodels such as the one in Figure 4.

4 DIMENSION 2: COMPLEXITY

John Muir (1911) is quoted as saying "When we try to pick out anything by itself, we find it hitched to everything else in the Universe." Muir's fascination with nature and this comment holds equally well for the complexity inherent within models—they are connected to many other things, artificial and natural. Complexity can sometimes be thought of as a large amount of something, but more specifically, something is complex if it consists of heterogeneous components. Some definitions for complexity are related directly to information theory where a phenomenon is complex if it takes a longer series of bits to encode it. This is an entropy-based definition where maximal entropy (e.g., chaos, randomness) requires more information. A wall of homogeneous bricks, even if the wall is massive, is simple in construct, and not very complex.

Figure 4, aside from illustrating hierarchy and scale in demonstrating the qualities of abstraction, is also more complex than a simple Petri net because it is heterogeneous; the model is composed of two-

sub-models: a Petri net containing a finite state machine inside of transition t1. It is further complex by the increasing size of code necessary to simulate it since this code must simulate the Petri net and also internal state transitions.

In an emerging era of "big data," where data are increasing in size and distribution, complexity for simulation requires that we think about ways to connect simulation model components together across the internet, but also to connect components more effectively in the human-computer interface. For many decades, the model in Figure 3 would be considered ideal in representing a system. A user might launch a Petri net simulator, import this model, and execute the model to obtain results. This sort of simplicity is no longer acceptable. Users want to connect models to reality, for instance. If Figure 3 captures the dynamics of a robot arm and drill, where are is the actual arm and drill? There should be a connection between the model and the manufacturing station. This synchronization especially within the sense of automatic control is known by the term cyberphysical systems, and is a major thrust where digital models help to control large-scale physical systems. This type of complexity, though, goes beyond cyberphysical to a more general hypermodeling (Fishwick 2012a). When we view the Petri net transition t1 in Figure 3, we want to see an internet video feed of the robot arm, or we might want to immersive ourselves in a virtual manufacturing workstation complete with avatars, robots, drill machines, and other machines. Everything, from reality, to modeling and virtual environments has to mesh and be integrated. This is most certainly a complex task, and represents a highly heterogeneous and interconnected set of virtual and physical assets. The shift to thinking about modeling within larger, more integrated, contexts can be seen in the exhibit hall of our simulation conferences where there is a push for more realistic 3D environments. Users are no longer content with simulation models as diagrams; they want something richer, more complete, and ultimately involving more integrated complexity.

5 DIMENSION 3: CULTURE

The exponential rise of social networking tools such as Facebook and LinkedIn points to the need for us to find ways of improving our social communication. Culture is centered on aspects of group formation, history, and process. Culture is also one of most important aspects of the modeling process since modeling is rarely accomplished in a vacuum; modeling is about people communicating with each other, emphasizing procedural and behavioral aspects of phenomena. A core aspect of culture for the simulation field is *language*. What are the textual and visual languages that we can use to communicate our understanding of system behavior, with corresponding state and event changes? As for language, the most universal one used for simulation is that of mathematics. For Petri nets, the formalism for the net in Figure 3 begins with a tuple: <P,T,I,O>. The tuple is an ordered set and contains sets and relations. P refers to the set of places, T the set of transitions, and I and O are both functions (i.e., a special type of relation) that map the set of input places to output places. Figure 3 can be codified as follows:

- P = {p1,p2,p3,p4}
- T = {t1,t2}
- I defined as the map I(t1)={p1,p2}, I(t2)={p2}
- O defined as the map O(t1) = {p2}, O(t2) = {p3,p4}
- μ is the marking that captures the state of the Petri net, μ = (1,0,1,0) initially

Mathematical definitions are products of the language of mathematics and have advantages and disadvantages as for any type of model. The strength of mathematically defined system formalisms is that the language is widely deployed, understood, and captures the abstract set-theoretic foundations for dynamic systems. The universality and high level of abstraction can also be a disadvantage since the model structure is perceptually distant from the phenomenon under study making it a language with a smaller set of adherents. With dimension 2 in mind, ideally, we will in the future have heterogeneous models that

have both mathematical and visual attributes on demand so that each can be viewed when needed, depending on need and who is working with the model.

Using our Petri net model, Petri nets have *cultures* and *sub-cultures* associated with them. There are conferences and periodicals, for example, where the adherents use Petri nets much as a group of people might all speak to each other in a natural language such as Chinese. A group at company X may orient their business workflow around a specific type of modeling, an approach, or even a commercial simulation package. This group behavior creates a culture around the modeling type or package. Cultures, like any other aspect of group behavior, come and go and so have different lifespans. Cultures formed around computer games are particularly prone to evanescent cultures—groups form and coalesce around a game engine or game package, and simulation activity that appears within the game serves as a seed for manifesting cultural norms.

The reasons why culture is important to modeling stretch across our human needs and group behaviors. There are often not logical, or universal, reasons for a specific type of modeling language used for simulation since the reasons for model choice are based on cultural norms and group preferences. Sometimes, cultural practices suggest to one group that a specific modeling type is superior to others, but often this competition is sociological. As in natural and formal languages, there are no universals—people are motivated frequently for sociological reasons, resulting in a wide variety of modeling languages, types, and presentations. There is no obvious trend that this diversity will lessen in modeling.

Model languages exhibit two different characteristics, both of which are useful in defining dynamics: *rule* and *flow*. A differential equation is a good example of a model based on a rule. Equations are usually formulated in natural systems to conserve variables such as energy and force. The following equation is typical of what you may find in a textbook on system dynamics where one variable defines the value of interest: $x'' - 2x' + 3x + 5 = 0$. This equation, which can also be rewritten as $x'' = 2x' - 3x - 5$, contains shorthand for derivatives (e.g., x' means dx/dt). Even though this model specifies a rule, a behaviorally equivalent model can be built in Figure 5 on the concept of flow.

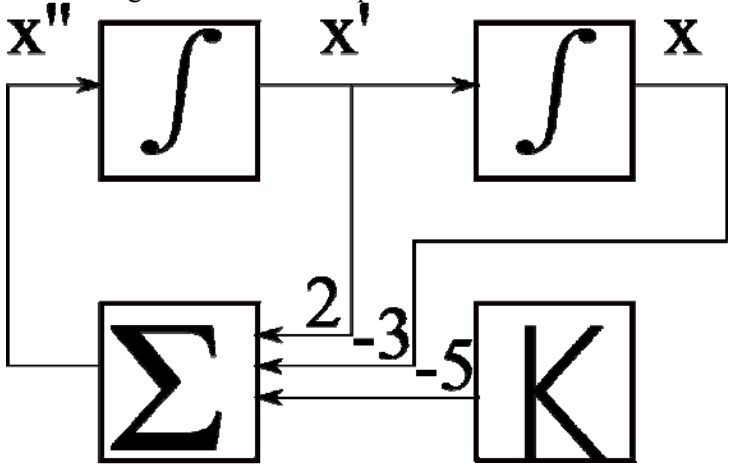

Figure 5: Functional block data flow model representing $x'' - 2x' + 3x + 5$

There are two types of flow that are common in simulation: *data flow* and *control flow*. Figure 5 is a type of data flow network (i.e., graph) because data flow around the network as the data are processed through the four functions (two integrators, one adder using the sigma sign, and one constant using the symbol "K"). Initial conditions at time zero for the system are set on two wires (x' and x). Control flow is where there is a discrete signal sent from one block to another indicating that the receiving block should begin its function. Software flowcharts are good examples of control flow. At a certain level of abstraction, all flow-based models are really the same; terms such as control and data flow are used primarily to indicate differences between discrete vs. continuous signals. The equation is sometimes referred to as de-

clarative knowledge whereas Figure 5 is procedural, or imperative, knowledge (Fishwick 1995,Abelson 1996). This differentiation is due to the form of each model; the equation says little about how it is to be solved whereas for the data flow graph, the solution method is made more transparent in model syntax.

Each year since 2000, I have taught a class called Aesthetic Computing (Fishwick 2012b) where students explore alternative, often analog, approaches to representing formal languages. The main idea behind aesthetic computing is that if one is to learn a formal language, this learning can be facilitated through cultural leverage. For example, if there are a group of people who enjoy playing Minecraft (2012), then if we need to introduce these people to a formalism such as Petri nets, we can do so through leveraging Petri nets in Minecraft. This is a different approach to education than starting with the learning objective, and then proceeding with a universally applied interactive approach to achieve that objective. Bring the topic to the person, rather than the person to the topic.

In the class during Spring 2012, several projects were built along the lines of using game engines to encode formal language constructs. One of the projects (Tadayon, Wilson, and Vo 2012) involved a simple Petri net (e.g., the third Petri net in Figure 2) using Minecraft features including dispensers, eggs, lava, and pistons. Figure 6 shows a side view of the Petri net. The game of Minecraft allows players to build arbitrary objects out of mined cubes, with the redstone cube being the one necessary to model message propagation needed for simulation modeling.

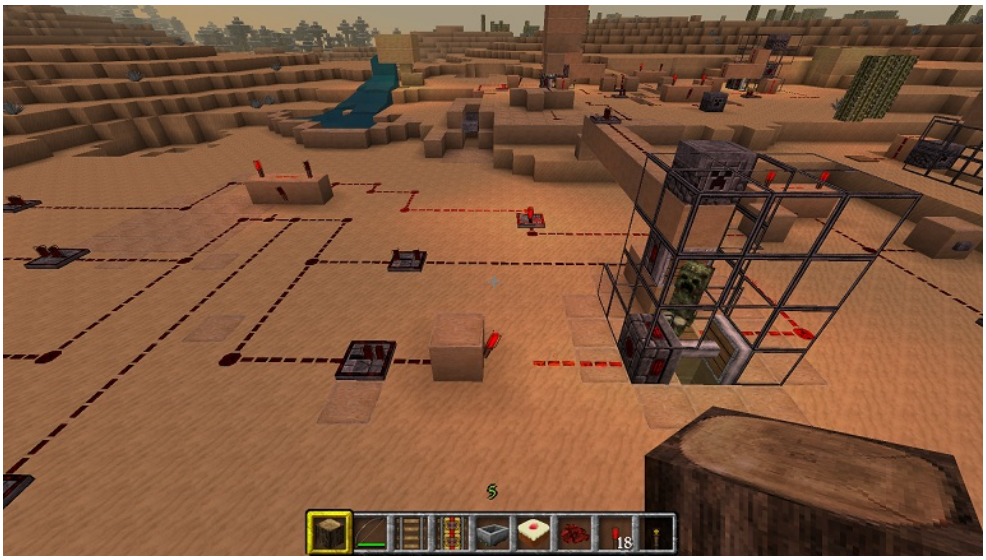

Figure 6: Side view of Petri net in Minecraft (from Tadayon, Wilson, and Vo 2012)

6 DIMENSION 4: ENGINEERING

Modeling is a complex *process*, resulting in one or more models over an extended period of time. The process used to create models is one that has a lifecycle from conceptual design to implementation (Overstreet et al. 1994, Law 2006). It is educational, and fairly accurate, to observe analogies between how simulation models evolve and are analyzed to the field of *software engineering* which is concerned with how software evolves. Software engineering is a discipline where practitioners and researchers practice and study how software is created from the very initial conceptual stage (Robinson 2011) to analysis, maintenance, verification and validation. Let's consider some of the phases used in software engineering while translating our need for modeling into distinct phases:

- *Requirements*: obtaining basic information about why the model is needed, and what the model is intended to achieve, as well as who will be using the model.

- *Design*: design of the model with specific languages and tools for authoring the modeling. Is there a conceptual phase prior to preliminary and detailed model design phases?
- *Implementation*: creating software that will allow some to author models and others to simulate them. What programming and model languages are needed?
- *Verification and Validation*: create testing frameworks to ensure that the model is correctly implemented so that it matches all requirements, and that the model accurately models the phenomena (Sargent 2009). Sometimes, these two terms are known by the questions "Is the model right?" (verification) and "Is it the right model?" (validation).
- *Maintenance and Quality Assurance*: create institutional processes to maintain all models and ensure their evaluation toward the goals identified during the requirements analysis.

Returning to our common topic of Petri nets, how are these engineered? This is a topic that requires more study since as for most model types, engineering and lifecycle issues are too often ignored or not stressed. Modelers who build Petri nets usually construct them directly without stages, unlike in the modeling technique known as System Dynamics (Roberts et al. 1994) where that approach is an engineering method, and does not solely define model structure and analysis.

7 DIMENSION 5: ENVIRONMENT

All of the dimensions we have discussed so far occur in particular environments. The term "environment" is used in products such as the integrated development environment (IDE). An IDE is the encompassing framework defining how models and software are conceptualized, designed, built, and tested. When thinking about environment, we can borrow from the virtual reality community where they have the term *virtuality continuum*. The virtuality continuum spans the real, physical, environment on the left and a virtual environment (Fishwick 2009) on the right. The word "virtual" is meant to describe a synthetic environment. In the middle, between the two extremes of physical and virtual, we have mixed reality where elements of the physical and virtual interact. Environment can also refer to the context in which modeling is done or used. Models can be present within a mobile environment or in a fixed environment. The types of models that we use, or the ways in which we interact with them, differs depending on environment. In this sense, environment is equivalent to context. Figure 6 illustrates the continuum (VC 2012).

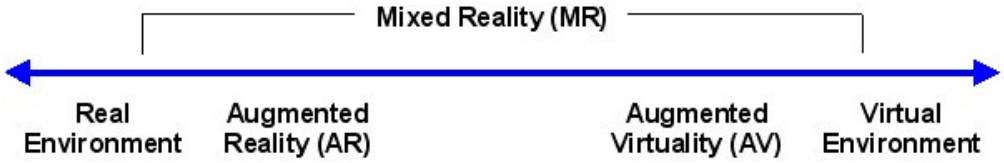

Figure 6: Virtuality continuum spanning real and virtual (from VC 2012)

8 DIMENSION 6: PROCESS

When practitioners of simulation talk of simulation and models, they often define models through the method of execution of the software that results from compiling the model. This is a bit like referring to a car as slow or fast when asked to describe the car, rather than expounding the design or structure of the car. Winter Simulation Conference participants frequently focus on things like discrete event or combined models, and occasionally, continuous models although historically, the conference has emphasized discrete event system behavior. The following captures the main process distinctions:

- *Discrete event*: results in advancing time in irregular increments. Example: the simulation of a process involving the arrival and departure of tokens or people in a network of servers where queuing occurs.
- *Discrete time*: results in advancing time with regularly spaced, fixed time steps. Example: the simulation of a population of animals.
- *Continuous*: results in advancing time in regular, arbitrarily small, time steps. Example: physical systems based on ordinary or partial differential equations.
- *Distributed*: results in advancing time across different computing systems, often using some form of conservative or optimistic time advance strategy.
- *Parallel*: results in advanced time in parallel within one computer system cluster or multi-core/processing element system.

Petri nets are typically simulated using discrete time where all transitions are of equal time duration, and discrete events otherwise. The five categories are not the only process categories—there are Monte Carlo models, for example, where discrete event simulation is usually used, and yet, Monte Carlo refers to an algorithm rather than syntax, and so captures an algorithmic approach to simulating a system. The term input modeling (Kelton 2009, Biller and Gunes 2010) refers to the use of statistical distributions whose random sampling facilitates simulation. Input model definitions are frequently referred to using the term "process." When you hear about models using such terms, people are referring not to the syntax or design of the model, but rather to how the model is compiled and executed. In some instances, if one speaks of a "discrete event model," this says something about how the model is configured since the concept of discrete event must be surfaced somewhere in the model design. However, it is fairly common for one model type to be capable of being simulated using a variety of execution approaches, therefore, be aware of how people use the term model since in some cases they may be using it in the design sense and at other times in a process sense.

ACKNOWLEDGMENTS

I acknowledge all associates, students, friends, and teachers in presenting this material. The Winter Simulation Conference, with all of its attendees, in particular has served as a rich repository of knowledge about modeling and simulation. I am indebted to the conference, its history, and all of these attendees. I would also like to thank the WSC 2012 Program Chair (Adelinde Uhrmacher) and Introductory Tutorial Track Chair (Helena Szczerbicka) for their encouragement to me to create this tutorial.

REFERENCES

Abelson, H. 1996. The Structure and Interpretation of Computer Programs. Second Edition. MIT Press.

Biller, B. and C. Gunes. Introduction to Simulation Input Modeling. In *Proceedings of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan. 49-58. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. Accessed June 8, 2012. http://www.informs-sim.org/wsc10papers/006.pdf

Chan, W. K. V., Y. J. Son, and C. M. Macal. 2010. Agent-Based Simulation Tutorial: Simulation of Emergent Behavior and Differences between Agent-Based Simulation and Discrete-Event Simulation. Edited by. B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan. 135-150. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. Accessed June 8, 2012. http://www.informs-sim.org/wsc10papers/014.pdf

Ezzell, Z., P. Fishwick, B. Lok, S. Lampotang, and A. Pitkin. 2011. An Ontology-Enabled User Interface for Simulation Model Construction and Visualization. Journal of Simulation, Palgrave MacMillan. 5(3): 147-156.

Fishwick, P. A., 1995. Simulation Model Design and Execution: Building Digital Worlds, Prentice Hall. Englewood Cliffs, NJ.

Fishwick, P. A. 2009. An Introduction to Opensimulator and Virtual Environment Agent-Based M&S Applications. In *Proceedings of the 2009 Winter Simulation Conference*, Edited by M. D. Rossetti, R. R. Hill, B. Jhansson, A. Dunkin, and R. G. Ingalls. 177-183. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. Accessed June 8, 2012. http://www.informs-sim.org/wsc09papers/015.pdf

Fishwick, P. A. 2012a. Hypermodeling: An Integrated Approach to Dynamic System Modeling, Journal of Simulation, Palgrave MacMillan, 6:2-8.

Fishwick, P. A. 2012b. Aesthetic Computing. In Soegaard, Mads and Dam, Rikke, Edited by. Encyclopedia of Human-Computer Interaction, Aarhus, Denmark: The Interaction-Design.org Foundation. Accessed June 8, 2012. http://www.interaction-design.org/encyclopedia/aesthetic_computing.html

Gehlot, V. and C. Nigro. 2010. Colored Petri Nets. 104-118. In *Proceedings of the 2010 Winter Simulation Conference*, Edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan. 49-58. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. Accessed June 8, 2012. http://www.informs-sim.org/wsc10papers/012.pdf

Kelton, W. D. 2009. Representing and Generating Uncertainty Effectively. In *Proceedings of the 2009 Winter Simulation Conference*, Edited by M. D. Rossetti, R. R. Hill, B. Jhansson, A. Dunkin, and R. G. Ingalls. 40-44. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. http://www.informs-sim.org/wsc09papers/005.pdf

Law, A. M. 2006. How to Build Valid and Credible Simulation Models. In *Proceedings of the 2006 Winter Simulation Conference*, Edited by L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M Nicol and R. M Fujimoto. 58-66. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. Accessed June 8, 2012. http://www.informs-sim.org/wsc06papers/006.pdf

Minecraft. 2012. Accessed June 8, 2012. http://www.minecraft.net/

Muir, J. 1911. My First Summer in the Sierra. Houghton Mifflin, Boston, MA.

Overstreet, C. M., E. H. Page, and R. E. Nance. 1994. Model Diagnosis Using the Condition Specification. In *Proceedings of the 1994 Winter Simulation Conference*, Edited by J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila. 566-573. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. Accessed June 8, 2012. http://informs-sim.org/wsc94papers/1994_0084.pdf

Petri, C. A. 1962. Kommunicakation mit Automaten. Schriften des Rheinisch-Westfälischen für Instrumentalle Mathematik an der Universität Bonn Nr. 2.

Roberts, N., D. F. Andersen, R. M. Deal, and W. A. Shaffer. 1994. Introduction to Computer Simulation: A System Dynamics Modeling Approach. Productivity Press.

Robinson, S. 2011. Choosing the Right Model: Conceptual Modeling for Simulation, In *Proceedings of the 2011 Winter Simulation Conference*, Edited by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu. 1428-1440. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. http://www.informs-sim.org/wsc11papers/128.pdf

Sargent, R.G. 2009. Verification and Validation of Simulation Models. In *Proceedings of the 2009 Winter Simulation Conference*, edited by M. D. Rossetti, R. R. Hill, B. Jhansson, A. Dunkin, and R. G. Ingalls. 162-176. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. Accessed Jun e 8, 2012. http://www.informs-sim.org/wsc09papers/014.pdf

Tadayon, R., R. Wilson, and P. Vo. 2012. Petri Nets in Minecraft. Accessed June 8, 2012. http://www.youtube.com/watch?v=sUR3xqPWU1I

VC. 2012. Virtuality Continuum. In Wikipedia. Accessed June 8, 2012. http://en.wikipedia.org/wiki/Reality%E2%80%93virtuality_continuum.

Walter, G. G. and M. Contreras. 1998. Compartmental Modeling With Networks. Birkhäuser Boston, Boston, MA.

AUTHOR BIOGRAPHY

PAUL FISHWICK is Professor of Computer and Information Science and Engineering and Director of Digital Arts and Sciences at the University of Florida. He received the PhD in Computer and Information Science from the University of Pennsylvania, and performs teaching and research in modeling and simulation with an emphasis on human-model interaction. Fishwick has over 200 technical refereed publications, is a Fellow of SCS, served as General Chair of the Winter Simulation Conference (WSC) in 2000, and was a WSC Titan Speaker in 2009. He currently serves as Chair of ACM SIGSIM. Recently edited books include the CRC Handbook on Dynamic System Modeling (Taylor & Francis) and Aesthetic Computing (MIT Press) His email address is fishwick@cise.ufl.edu and web page is http://www.cise.ufl.edu/~fishwick