Metasimulation

Andy Ceranowicz
Alion Science and Technology
Harvard, MA
aceranowicz@alionscience.com

ABSTRACT

Simulation is used in an increasingly large segment of our scientific, economic, entertainment, and government activities. Its expanding influence makes it important to understand its strengths and limitations. However, each field specializes simulation for its problem domain making it difficult to agree on a common definition. In this paper, I survey the fundamental mechanisms underlying simulation and attempt to come closer to such a definition. In this quest, I have borrowed heavily from metaphysics, especially the concept of possible worlds from modal logic. I define a simulator as a device that uses deduction and sampling to incrementally create possible worlds. I define simulation analysis as the use of analogical reasoning to map actual or hypothetical target worlds to a simulator and the use of induction on the possible worlds produced by the simulator to make claims about the target world. The analogs that simulators are built on can be physical, human, symbolic, analog (computer), digital or hybrid. The construction of simulation analogs requires approximating the target world with a finite model bounded by means of inputs, state, and objects. Integration of small changes and sampling are the magic bullets that allow simulation to tackle problems that are impossible to solve by analytic means. Small changes decouple complex systems and sampling replaces the complexity of the general with the simplicity of the concrete. Parallel simulator design and its realization as a posteriori composition of simulators are reviewed. I find that advancing simulation time is not a necessary or sufficient criterion for identifying a simulator and that simulation is used widely by the human mind and probably by animals. It may even be the foundation of consciousness. Finally I examine a posteriori simulator composition and interoperability concluding that once it becomes technically viable, a priori composition will be a better approach.

ABOUT THE AUTHOR

Andy Ceranowicz is the Navy Continuous Training Environment (NCTE) Architect at the Navy Warfare Development Command (NWDC) and a Chief Scientist at Alion Science and Technology. At NWDC Andy is responsible for standards development to maintain interoperability in a worldwide training environment that links live ships, virtual trainers, and constructive Joint Semi-Automated Forces (JSAF). At U.S. Joint Forces Command J9 he was responsible for large scale federation development, including the development of the Urban Resolve federation, which used supercomputer assets to simulate urban populations with hundreds of thousands of individual entities. Andy also led the integration of the Millennium Challenge 2002 federation which linked together entity level simulations from all the Services and served as the basis for the Joint Live Virtual and Constructive simulation used by Joint Staff J7. Prior to that, he led the development of a line of SAF systems including SIMNET SAF, ModSAF, and JSAF. Andy received a Ph.D. from The Ohio State University, specializing in communications and control systems.

Metasimulation

Andy Ceranowicz
Alion Science and Technology
Harvard, MA
aceranowicz@alionscience.com

METASIMULATION

I use the term metasimulation in analogy to metaphysics. Metaphysics (Van Inwagen, 2013) comes to us from the writings of Aristotle. It is said to have gotten its name from the canonical order his treatises were placed in after his death. His writings on metaphysics were placed after, "ta meta ta", those on physics. The subject of metaphysics is the real nature of things beneath their changing physical appearances. It is hard to define precisely, but it is deeply concerned with questions of existence and identity. After spending a lifetime developing simulations, I am going step back and try to get a deeper view of what underlies simulation. Rather than attempt to do so from the viewpoint of computer science, mathematics, or systems theory, I am going to do so from a philosophical perspective, asking questions of a more metaphysical character. What is simulation really? What makes it work? Why do we do it?

Although the title of my degree contains the word philosophy, my training was in its natural aspect; the 'physics' not the 'metaphysics'. So I do not attempt to present or solve any philosophical questions. Indeed, it is not clear that there are any new philosophical questions arising from simulation (Frigg and Reiss, 2009). As we gain experience in an area, we build up an intuitive understanding of it. Here, I seek to write out my intuitive understanding of simulation and its properties into a logical framework and in the process discover new connections and common principles. The focus is on the description of what hopefully will turn out to be a productive perspective on simulation.

WHAT IS SIMULATION?

This is a difficult question not because simulation is unknown, but because there are so many different forms of it. Like proverbial blind men trying to describe an elephant by feeling different parts of it, we are all familiar with our little piece of simulation. It is only when we look around at all the other pieces that confusion arises. What we can't see is the Platonic form or ideal that can unequivocally identify simulation. It may sometimes seem as if the various activities performed under the name of simulation are connected only in name; but if they were truly disconnected, then they could easily be separated and classified. Instead, what we see are not isolated clusters of concepts, but clusters that are richly connected to each other.

This correlates with the absence of a universally acknowledged discipline devoted solely to simulation. Only recently have some universities started to grant degrees in modeling and simulation. Each discipline has its own tradition and techniques which its practitioners tend to teach to their students in a parochial manner. The application specific aspects often dominate the general aspects and due to limited time, practitioners tend to focus on the application specific aspects. However, simply slicing simulation into business simulation, chemistry simulation, medical simulation … and so on does not provide a very satisfying answer. In this section, I provide an introduction to simulation.

Teleology

Perhaps simulations are united by their purpose? From a teleological view, simulation is a technique that allows us to answer questions like "What would the world look like if this sequence of actions is applied to it?". Scientists seek to understand the behavior of the world through its response to various interventions. In our personal lives we need to interact with the world through our actions and suffer their consequences, so the question can be rephrased as "What are the consequences of my actions going to be and should I take those actions or not?". Any tool that can help us answer these questions is going to be very useful. Take, for example, the rapid adoption of GPS navigators.

They tell us what actions we should take to get to our destination in the least time. Although these devices include a significant optimization component, in a very real sense they also simulate the journey we are about to take. Simulation is very closely related to planning which turns the simulation question on its head and asks "What actions should I apply to the world to make it satisfy this goal?". As with the GPS navigator, planning often includes optimization. For simulation researchers, it can be very useful to become familiar with the planning literature.

However, the normative form of the teleological goal is not universal. Statistical simulation can simply be a way of calculating integrals. Simulation is now widely used in entertainment and while entertainment can be educational and inspirational, much of it is simply escapist rather than normative. Changing 'the world' to 'a world' helps to better align the teleological question, but still what video game players want is the experience of beating the game. Simulation is also used in entertainment to automate the creation of animation frames that display realistic movement (Erleben et al, 2005). While this does show the effects of actions, the purpose is to produce realistic animations. So the first question posed, "What would a world look like if this sequence of actions is applied to it?", is a better criterion for identifying a simulation than the second two, which are normative and better characterize planning.

Historical Origins

Can we identify simulations by their intellectual heritage? Three of the foundational traditions of simulation are system dynamics, statistical, and discrete event simulation. Systems dynamics simulation is based on Newton's formulation of dynamic systems as differential equations and their solution via numerical methods. Numerical integration of differential equations over time predicts the future state of a system described by those equations. The state is determined by the system's internal dynamics, initial state, and external inputs. These are common characteristics of simulation. Dynamic systems simulation not only predates the digital computer, but was the direct motivation for the construction of the first practical digital computer, the ENIAC (Electronic Numerical Integrator And Calculator) (Weik, 1961).

Because of the tediousness of human calculation and the approximate nature of simulation results, analytical or closed form solutions were preferred for over 200 years after the publication of Newton's *Philosophiæ Naturalis Principia Mathematica* in 1687. However, as the complexity of technology escaped the capabilities of analytical solutions because nonlinear and time varying phenomena had to be included, human simulation became the only option. A primary example is the generation of artillery tables by simulating the flyout of ballistic artillery shells. The wars of the 20th century made these sorts of calculations a critical need and the differential analyzer, a mechanical analog computer, was developed to meet it by Bush in 1931 (Owens, 1986). However, the unreliability of these computers led to the Army Ballistics Lab funding of the construction of the ENIAC in 1943. The first version was completed in 1945. Using a mechanical calculator, a human computer could simulate a 60 second flyout in 20 hours, a mechanical analog computer could complete the simulation in 15 minutes, and the ENIAC could do so in 30 seconds. It was faster than a speeding bullet.

The statistical form of simulation is built on the concept of sampling from random distributions. The first sampling problem is attributed to Buffon in 1777 (Goldsman, Nance, Wilson, 2010). By randomly dropping needles on a surface with parallel lines and counting how many cross a line you can estimate π. The common use of sampling techniques for solving mathematical models did not emerge until the availability of the ENIAC. While working on the design of the thermonuclear bomb, Ulam realized that computers could be used to generate pseudorandom samples and that those samples could be used to solve diffusion problems which were too complicated for solution by means of analytical or dynamic systems simulation techniques (Metropolis & Ulam, 1949). The Manhattan project used Monte Carlo techniques heavily for the design of nuclear weapons (Metropolis et al, 1953). This work evolved into the field of statistical simulation where samples from simpler statistical models, i.e., probability distributions, are used to 'simulate' samples from distributions which are difficult to obtain analytically (Kruschke, 2011).

Discrete event simulation is a form of simulation which emerged with the field of Operations Research (OR). One of the first OR simulations was conducted by the National Coal Board in the UK after a 1953 conveyor-belt fire that killed 80 men (Rand, 2011). The study was conducted with paper and pencil, looking up random numbers from tables. Computers became available shortly thereafter and together with Monte Carlo sampling techniques they were applied to performing these types of studies leading to Discrete Event Simulation (DES). They focused on problems

where an object's entire relevant set of states and activities could be reduced to a small set of disjoint discrete values. Transition events cause objects to instantaneously change state, such as a customer going from a waiting-state to a being-served state. The time spent in each state is modeled by a probability distribution. During a simulation run, Monte Carlo sampling and resource constraints are used to determine the time between transitions. Discrete event models use a simplification strategy similar to lumped parameter models used in electrical and mechanical systems analysis, except that instead of lumping spatially distributed qualities such as resistance and mass, DES lumps temporal activities and states. When looked at in this way the parallel between DES and qualitative reasoning (Kuipers, 1994) seems obvious. Because the first computers had to be programmed in primitive assembly languages, many different simulation languages emerged. Tocher developed the first general purpose discrete event system programming language, the General Simulation Program, as a tool for modeling industrial plants, predating FORTRAN (Goldsman, Nance, Wilson, 2010). Later the SIMULA (Nygaard & Dahl, 1981) simulation language produced the first example of object oriented programming eventually leading to the subsequent revolution in programming.

Although, these strains of simulation emerged as major techniques in a short period of time around the arrival of the ENIAC, they all had precursors prior to the digital computer. All three approaches now use sampling from random distributions and many of today's simulators have features from all three approaches. Using a system theoretic framework, Ziegler has shown that discrete event simulation can be used to simulate both dynamic systems and sampled dynamic systems (Zeigler, Praehofer, & Kim, 2000). The arrival of computers took these simulation and analysis techniques and not only accelerated their use and development, but also linked them together via programming, statistics, and mathematics to create methodologies that have distinct communities but share many ideas and techniques.

If it Looks Like a Duck

Another way to decide whether something is a simulation is to look at its components. Is there a canonical structure for a simulator? To be a little less ambiguous, I am going to try to use the term 'simulator' to indicate the software and hardware that is used to produce simulated data. 'Simulation' will be reserved for the process of executing or running the simulator. 'Simulation analysis' will be used for the process of constructing simulators and performing analysis with them. A 'run' is one execution of a simulator. So what I am now discussing are the components of a simulator. A common depiction of a simulator is shown in Figure 1. At its heart are the state variables.

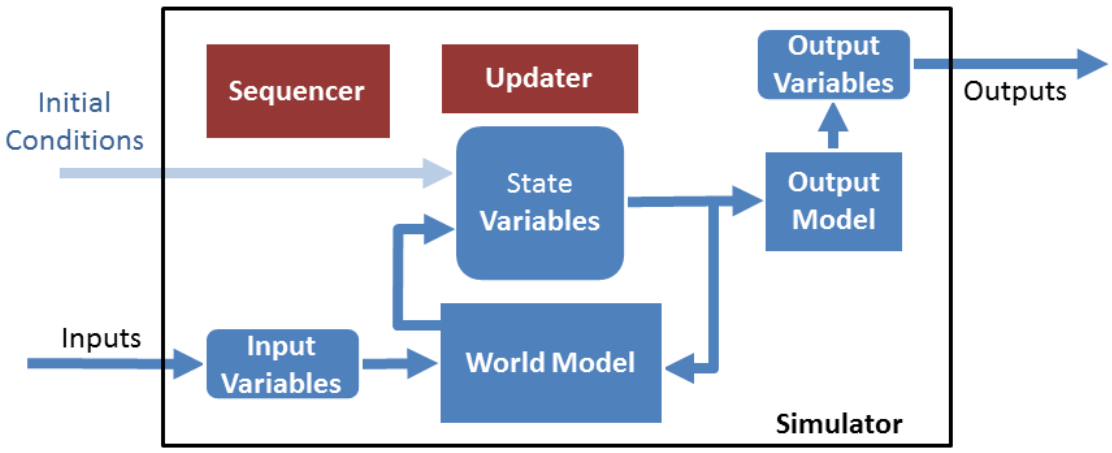


Figure 1: Decomposition of a Simulator

This is the set of variables that is used to express the configuration of the system or "world" being simulated. When assigned values, the state variables represent one particular state or configuration of the system. At the start of a simulation, the state variables are set to the initial conditions. The future values of the state variables change based on their current values and on external inputs. The sets of state, input, and output variables are not fixed but can add or drop members over the course of a run. The world model is a set of functions or rules that use the values of the input variables and the current state variables to calculate what the next state variables and values are going to be.

As is shown in Figure 1, the initial conditions and inputs come into the simulator from outside. The output model uses the values of the state variables to calculate a set of output variables which are the simulated data produced by the simulator. They are accessed from outside the simulator for control, logging, analysis, and display. Two additional pieces of the simulator are shown in red: the sequencer and the updater. These components are not usually shown in diagrams of simulator decompositions. This may be why the term model is sometimes used in place of simulator. For example, "Joint Semi-Automated Forces (JSAF) is a Navy model" would be restated in this paper as "JSAF is a Navy simulator" or "JSAF contains a Navy model". The sequencer and updater are therefore critical pieces of a simulator. The updater performs the calculations to generate new state and output variables. The sequencer guides the updater through the order and timing of the updates.

There are many variations on this decomposition. There may not be any inputs or the inputs may be preset at the start of the run for presentation at future times in the run. This enables batch runs which are executed without any human intervention. It is common for many input variables, especially stochastic variables, to be hidden away in the world model. The state variables may be the outputs in which case the output model may be dropped. The world model may be combined with the state variables. Inputs and outputs may be divided into persistent variables and transient events. Events can be described as instantaneous impulses which cause discontinuous state changes. An explosion is an example of an event. What qualifies as an event depends on the time scale of the simulator. In summary, state variables, with an ability to set initial conditions, a world model, updater, sequencer, and output appear to be necessary to have a simulator.

SIMULATION IS INFERENCE

At its most fundamental, simulation is an inference method. Inference is the process of making implicit knowledge explicit or concrete. If, as the classic example goes, I know that all men are mortal and that Socrates is a man, then I know implicitly that Socrates is mortal. I can apply classical Aristotelian syllogisms (Smith, 2014), or deduction, to make the knowledge that Socrates is mortal explicit. So it is with simulation. If I know the current state of the world, I can apply simulation to see what the world might evolve to in the future. In my knowledge of the present and the past, there is some implicit knowledge of what the future can hold. Simulation analysis is the process of making that knowledge explicit through the use of deduction, sampling, induction, and analogy.

Inference in Simulation

Deduction takes the rules of logic and applies them to statements with known truth values, called premises, to derive the truth values for additional statements. Deduction is the gold standard of inference because as long as all the premises are valid, it will only produce valid conclusions. It is truth preserving. However, this is also its weakness. Its outputs are only guaranteed to be true if its inputs are true. Mathematical and logical equations are deductive inferences of the form, if and only if. That means whatever is on one side of the equation must match what is on the other side. Simulations use mathematical and logical equations to express relationships that must hold between state and input variables at the current time and the state variables at the next. Since the values of the input variables and the initial conditions of the state variables are provided externally, we can use them to compute the state variables of the next time step. Then we can use the newly calculated state variables to calculate the values at the next time step and so on. Deductive inference is also used in world models when classification has to be done to determine the right equations to use for the calculation of the next state. This is needed where the system includes nonlinear effects with a threshold separating one operating regime from another. Examples are decision points and phase transition points. These types of behavior in a simulation can produce chaotic effects in deterministic systems where a small difference in the current state may cause large differences in future states. A slight difference in a threat estimator can make the difference between an aircraft getting shot down or not. A slight difference in position can make the difference between getting killed or not.

Not everything in the world has deterministic properties and behaviors. Some object classes have properties whose values are characterized by probability statements. Probability logic (Denny, Kooi, & Sack, 2013) is used to make inferences from probabilistic statements. Statements describing probabilistic constraints can be used to derive the probability of a particular result. Because simulators produce specific timelines, it is necessary to sample that probability distribution and to instantiate a particular value for it. This is why Ulam's realization that the ENIAC could be used to generate samples from a distribution was such a critical breakthrough in the history of simulation.

Once a sample is instantiated in a state variable, it is used in calculation just like any other variable. This provides a great simplification for future calculations. It is no longer necessary to keep track of its statistical properties as it is now just a concrete value. Sampling replaces the complexity of the general with the simplicity of the concrete.

Induction[1] starts with specific statements derived from observations of the world, such as "Socrates died", to produce general statements about the world, such as "all men are mortal". However, unlike deduction, induction cannot guarantee that its conclusions are valid even when all of its observations are valid and free from contradiction. In the best case, it can only guarantee that the conclusions are consistent with past evidence. This is because induction operates on an open world, someday there might exist a man that never dies. Simulators produce specific samples of system timelines. To draw conclusions from these samples, it is necessary to apply induction. In simulation analysis, induction is accomplished by two mechanisms: statistics and learning. Statistics provides mathematical tools to extract significant conclusions from simulation samples. People participating in simulations can apply their natural ability to learn from experience to their simulated experiences. Simulation training relies on the latter. Induction can also be used within the simulator when agents are programmed to adapt and learn during a simulation run.

Analogical reasoning (Bartha, 2013) uses known similarities between two systems, a target system and an analog, as a license to apply deduction and probabilistic sampling to the analog and then assert the inductive conclusions to the target system. Analogical reasoning adds additional validation requirements. You have to validate that the target and the analog are similar enough for statements about one system to hold for the other. As with induction, analogical reasoning cannot guarantee that its conclusions have the same truth or probability value in the target system as they did in the analog. Simulation analysis uses inductive and analogical reasoning and therefore requires validation. A simulator applies deduction and probabilistic sampling on the analog and simulation analysis uses induction and analogical reasoning to map the results of simulation runs back to the target system. This analogical reasoning is performed by people.

Figure 2 shows the simulation process. The blue circles represent features in the target system and the red circles represent features in the analog. The black arrows show the mapping between analogous features in the two systems. At the start of the simulation, the state of the analog is initialized to the state of the target system. Then the simulator is run until a target time or condition is reached. At that point, you could transform the state of the analog back into the state of the target. The inference is that, given the same initial state and inputs, the state of the target system could have proceeded over an equivalent path realizing this end state. In most cases, many runs will be performed and only summary statistics for the behavior of the analog will be mapped to the target.

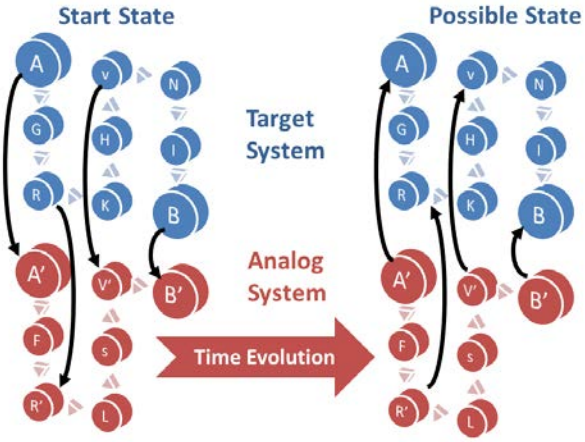


Figure 2: Simulation as Analogy

Analogs

Several types of analogs can be used for simulation. These include physical, human, analog computer, symbolic, digital computer, and hybrid. All of them have different properties and limitations.

[1] Mathematical induction is actually a deductive reasoning technique because it imposes the closed world requirement.

Physical analogs are physical objects used for simulation. Mostly these are scale models. A scaled down model of an aircraft can be placed in a wind tunnel to study the forces on the model and the turbulence it induces into the air stream. A scale model rocket can be used to investigate how well a real rocket will fly. (Gorman, 2011) describes the use of smaller caliber rounds as substitutes for actual rounds in training. These are all physical analogs and must be incorporated into a proper framework of instrumentation and controls to become simulators. Instrumentation is necessary in all but the simplest cases to get usable output from the simulator and the controls are necessary to prevent undesired characteristics of the analog from invalidating the results of the simulation. Under the right conditions, living systems can be used as analogs. There is sometimes a fine line between an experiment and a physical simulation. In a simulation something must be representing something else.

A special case of physical analog is the use of people for simulation. People have been performing simulations long before computers, analog or digital, were envisioned. In order to be a simulation, at least one component of the analog has to be different than it would be in the actual world. Military forces have engaged in training events and field exercises for their entire existence. Here humans are analogs for themselves operating in a different physical location or different context. Actors started simulating human experiences before the Greek dramas which are our first recorded plays. A fire drill is a simulated emergency without a fire. The process can be recursive, with actors pretending to be actors in a play about a play and so on. The phrase "pretending to be" is indicative of human simulation. Self simulation is sometimes referred to as emulation.

Analog computers are calculators that use continuous quantities for computation. Analog computers have been built from hydraulic, mechanical, and electronic systems. In hydraulic analog computers, fluid rate of flow in pipes and levels of fluid stored in tanks are the features used as analogs. In mechanical computers, such as the differential analyzer, turn rate and angular displacement are used. In electronic analog computers the features are current and charge in capacitors measured as voltages. Although they are physical devices, analog computers are designed be programmed like a digital computer to represent systems with completely different realizations. The correspondence is logical instead of direct. This distinguishes them from physical analogs. The mechanical differential analyzer was often used to simulate electrical systems. Programming in these computers requires physical reconfiguration. Like physical analogs the values chosen for analog simulation must be scaled properly. Values which are too large cause tanks to overflow, shafts to slip, and amplifiers to saturate. Values which are too low cause state values to be unobservable.

In symbolic analogs, abstract symbols are used to represent the model. They include natural language, logical, and mathematical models. Initially, only humans could draw inferences from symbolic analogs. Then digital computers were developed to automate symbolic inference. Since digital computers were designed to replicate the conscious mind's processing of symbolic analogs, they are in a sense working models of at least some aspects of human consciousness (Dennett, 1991). Digital computer models are a subset of symbolic analogs. The key difference between symbolic and physical analogs is that abstract symbols are used for representation instead of physical quantities like displacement or charge. Because the human mind is limited in the number of things it can track in working memory, the use of external media such as paper and pencil is necessary for any substantial human symbolic simulation. The advent of spreadsheets has blurred the line between digital and human simulation. People can combine both manual and automated digital calculation to update the state variables.

Digital computer memory is now our preferred medium for implementing simulation analogs. Although they use electronic circuits to represent state in the simulator, the circuits represent values as symbols, i.e., patterns, rather than as voltages directly. Digital computers are ideally suited for performing logical operations, they can be programmed more easily, and their results are easier to access than in analog computers. Digital computers still have scaling limitations such as round off, overflow, and underflow but the ranges of numbers they can represent exceed the ranges of analog devices by many orders of magnitude. Digital computers need not be electronic; Babbage's Analytical Engine (Computer History Museum, 2008) was a mechanical digital computer.

In hybrid simulation, the target system is mapped onto a heterogeneous system composed of elements belonging to more than one class of analog. In the 1960's large mainframe hybrid digital and analog electronic computers were commercially available and in use at large aerospace and engineering companies (Bekey & Karpulus, 1968). Shortly thereafter, the electronics industry started delivering digital computers that doubled in complexity and power about every two years in step with Moore's law. Moore also predicted (Moore, 1965) that linear (continuous) circuitry

would not be able to keep up with the scaling of digital electronics. He was correct and analog computers have disappeared from the consciousness of today's programmers. We are so conditioned to think of simulators as digital artifacts, that you might be surprised to hear that I have spent most of my career building and executing hybrid simulators. Distributed Interactive Simulation (DIS) (Shiflett, 2013) is designed to create Human-In-The-Loop (HITL) simulations. Even though we don't think of people operating virtual simulators as parts of analogs, they often are. The combined system of the human operator and the virtual simulator is the actual hybrid analog. Another form of hybrid simulation is hardware in the loop, where actual field hardware and software is interfaced to a simulator and used as a component of the simulator. The field hardware is part of the analog.

While simulators are currently digital and human, other analogs may eventually make their way into the mix. Moore's law depended on Dennard scaling (Dennard et al, 1999) which stated that transistor dimensions, clock speed, and power consumption would all scale together producing integrated circuits that were denser and faster but otherwise compatible with the current device support infrastructure. That changed in the middle of the last decade when power scaling was lost (Colwell, 2013). So now you can still build smaller, faster devices but you have to add a lot more cooling to do so. Adding multiple processor cores and onboard graphics has sustained Moore's law temporarily, but parallel processing will not be as generous in performance gains as Dennard scaling (Esmaeilzadehy, 2011). With the door for a replacement open, researchers are scrambling to find the next building block of information technology including neuromorphic analogs (Imam et al, 2012). Whatever they discover, neither digital nor HITL is going away so the future will remain hybrid and possibly become more so.

Possible Worlds

When you make a statement about something for which instances exist in the real world, you can determine its truth value from observations or reports from the actual world. But how do you determine the truth value of statements about counterfactuals. Things that don't currently exist and have never existed. For example, consider the statement "Romney could have won the 2012 election." Is there a principled way of determining whether this statement is true? The philosopher's answer is modal logic (Garson, 2014). Modal logic deals with the deduction of truth values for statements that are qualified by the expressions 'it is necessary that' and 'it is possible that'. To interpret the validity of these sorts of statements, modal logic relies on the possible worlds semantics. A possible world is a set of statements describing the world and a set of truth values for those statements. We can generate other possible worlds from an initial possible world by changing the truth values for some of the statements in a way that does not produce any contradictions (Johnson, 2013).That is, all impossible statements must remain false, all necessary statements must remain true, and all constrained statements must continue to satisfy their constraints. If our possible worlds contain the statements 'Obama won the 2012 election' and 'Romney won the 2012 election', they can't both be true in the same world. The initial possible world is just any set of truth valuations that satisfies the constraints, usually one very close to the present actual world. Modal logics allow philosophers to translate modal statements into ordinary statements in possible worlds, apply a standard logic there and then apply modal logic's techniques for combining the results. The details of how this is done are complicated, but what is important for our purposes is the description of how the possible worlds are created. It sounds like simulation is being performed. There are also probability logics for possible worlds. Simulation is the evaluation of possible worlds inferences via the sampling of possible worlds.

We can equate the statements describing the world to simulator state variables and the constraints on statements, including which statements are always true and always false, to the world model. We can also equate the initial possible world truth values with the initial conditions of a simulation run. If the world model includes temporal constraints and we update the model repeatedly while advancing time, we have a very familiar simulation. I find this to be a very intuitive and compelling way of defining a simulator. A simulator is a mechanism for producing new possible worlds, typically from previous possible worlds. Repeated runs of a simulator from the same initial conditions produce a tree as shown in Figure 3. The tree is directed forward in time and a traversal of the tree is a timeline. Two different timelines can intersect. Additional runs from a different initial possible world will produce another tree which may intersect the first tree. If the age of the objects in the system can be neglected, then the links lose their direction and that tree will become a graph.

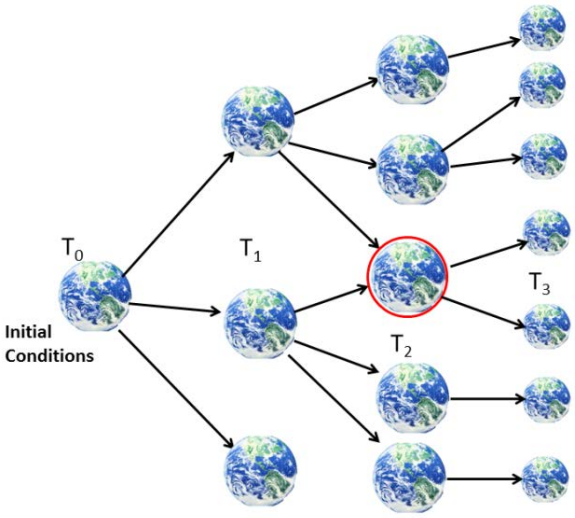


Figure 3: Possible Worlds Tree

One possible response, to my interpretation of simulation in terms of the logical concept of possible worlds, might be, "Oh no, you are taking simulation back to where artificial intelligence was in 1969, when John McCarthy's attempt to equate artificial intelligence to theorem proving ran into the frame problem" (Kamermans & Schmits, 2004) (Shanahan, 2009). The frame problem is the following question. Given you have described the world as a set of statements in logic and your model allows the application of certain actions to it. How do you know that your actions do not affect any statements beyond those specified in your action definition? If you have an action that changes the position of a block how do you know that it doesn't change the color of the block? Does your world model need rules of the form: action a_i does not affect statement s_j, for every statement that a_i does not affect? This question set off thirty years of controversy between philosophers and computer scientists. There are two replies to this response. First, I am not advocating replacing simulators with theorem provers. I am simply looking at possible worlds as a conceptual tool for understanding how simulation works. Second, fortunately the frame problem as a technical issue has been solved.

Some approaches for defining simulators focus on matching the trajectories of equivalent outputs in the target system and its analog. Those definitions are certainly valid and describe simulation development in the context of repeatable laboratory experiments. However, when dealing with chaotic or counterfactual systems, there are no trajectories to compare to. Rather, simulation developers and users examine the possible worlds created by the simulator for violations of logical, physical, and causal consistency. We are interested why a decision to fire a particular weapon was made or what caused that particular change in public opinion. Here, possible worlds semantics seem more natural. Possible worlds semantics have already been applied to computer simulation. Bisimulation (Garson, 2014) is a research area in computer science that uses possible worlds semantics. It is concerned with dissimilar state transition systems that produce equivalent outputs and therefore cannot be discriminated on the basis of their outputs.

Simulation in science is ultimately not used for prediction, but for confirmation. Simulators are built to replicate the proposed causal structure in the target. If an experimental system and a simulator produce the same outputs under the same initial conditions and inputs, the inference is that the causal chains inside the analog are identical to those of the target. The possible existence of a bisimulation means that simulation confirmation is inductive.

SIMULATORS ARE BOUNDED

If building a simulator is the mapping of features between a target system and an analog for the purpose of reasoning on the analog in lieu of the real thing, it seems obvious that what can be represented by the analog is bounded. There is only so much time and so many operations that can be expended on the mapping task. For logical and digital

models, there is only enough hardware for a limited number of state variables or statements to represent the state of the target. There is a quote that I attribute to Isaac Azimov but can't find the reference for: "The only computer that can simulate the universe is the universe". That is, if you are going to explicitly represent every quark in a system, you need something with more variables than there are quarks. Since we don't have anything like that, we need to bound what we seek to simulate.

There are additional reasons for bounding what we simulate. We need our simulators to be closed systems. For real world experiments, we stipulate closed systems so that variations in outcomes are not due to uncontrolled inputs. The same holds for experiments with simulated systems. We also need to understand our simulators sufficiently well to interpret their results and to be able to debug them when they don't work. The building of simulators is often a process of trial and error. So it appears that the world is working with us here. We need simulators that are bounded and we can only create simulators that are bounded.

System Boundaries

The first step in constructing the simulator is to decide which aspects of the world need to be included in the simulator and what parts can be ignored or held constant. This is problem specific, but generally more than the system of interest needs to be included because we need those parts of the environment that interact with it. Rather than isolating our experimental system as a scientist preparing a physical experiment would, with simulators we usually have the option of just not including the state we are not interested in or replacing it with constants. This is not entirely true for physical and human analogs. For humans some care needs to be taken not to provide them with more information than they should know. Once people are exposed to information it influences their future actions and you can't erase their memories. With physical analogs we have to ensure that the physical properties which we are not interested in do not interfere with the effects that we are trying to observe. System boundaries not only include spatial limits of what is going to be simulated but also limit the simulator's ontology; that is the list of allowable objects and events that can exist in the simulation. Furthermore, the conditions that can exist over a simulation run also need to be limited. This is because the world model for the simulator is only valid over a limited range of conditions. Ziegler refers to this as the simulation frame (Ziegler, 2000). Once the boundary has been set, the possible worlds that the simulator can produce are constrained to what has been included or what enters via an input.

Inputs

Inputs exist to allow the simulator access to external state or events. They are holes in the system boundary and must at a minimum be monitored if not controlled throughout the simulation run. They are independent variables whose values can be set by a simulator controller. Simulator inputs exist for a number of reasons. They can be used to intervene in the simulation to cause particular situations, or possible worlds, to come into existence allowing their subsequent evolutions to be observed and analyzed. They can be used to interface a simulator to an external system. In some cases, preplanned or recorded inputs have to be injected into the simulation with a repeatable order and timing. External states that have been assumed to be constant are actually constant inputs, but are rarely expressed explicitly. Instead, they are assumed in the world model. This makes it difficult to distinguish which external state variables have been assumed constant from those that have no effect on the model. However, rolling such assumptions into the world model simplifies the model and makes it more efficient for execution.

Humans

While the above ideas are fairly clear when thinking about purely digital simulators, i.e., constructive, they become somewhat cloudy when considering hybrid simulators composed of digital computers and humans. Are the humans inside or outside the simulator boundary? From the point of view of a simulation programmer, the simulator is the program and the humans are separated from the simulator by a human computer interface. They are outside. However, things may look different to the person responsible for the analysis of the simulation. Some of the roles humans play in simulations include observer, analyst, controller, experimental subject, trainee, or role player. Clearly, if the human has no effect on the possible worlds produced by the simulator, he is outside of the simulation boundary. The observer and analyst should be outside the boundary.

Influencing the possible worlds produced is not sufficient to place you inside the simulation boundary. The determining test is whether the actions of the human participant are a natural response to the current possible world for the character that he is playing. That makes the transition from the current possible world to the next a valid sample of the possibility of that transition occurring in the actual world. Experimental subjects and trainees are expected to behave as they would in the real world. Opposing force and other role players are supposed to behave as the people they are playing, i.e., simulating, would behave in the actual world. Properly behaving subjects, trainees, and role players are all inside the simulator boundary. Validating that these human participants remained inside the simulator requires monitoring of behavior during the simulation and performance analysis after.

Controllers act on the simulation changing the progression of possible worlds but their actions need not be natural or predictive and, therefore, may need to be factored out of the results. Controllers may act to shape the simulator state by driving it toward a possible world of interest to the study. The simulation may be entirely valid and even predictive, but if it doesn't contain any situations of interest, then it is useless. The simulation analyst will probably have to ignore that part of the simulation in his results. Another function of controllers is to correct anomalies in the execution of the simulator. If the bomb software failed to detonate a bomb on the appropriate trigger and the controller detonates it by hand, he is preserving the validity of the simulation so the response to the detonation would be retained in the analysis. The controller is outside the boundary, but is able to affect the simulation through inputs and change the possible worlds generated by the simulator.

SIMULATORS ARE MARKOVIAN

By saying simulation is Markovian, I am claiming that we can express the effects on future possible worlds of all the inputs applied to the system in all prior possible worlds in a finite set of statements or state variables. In other words, the computation of possible future worlds shall only access the statements and inputs in the current possible world. This is illustrated in Figure 4 which shows state as a wall between past inputs and events and the future. This is

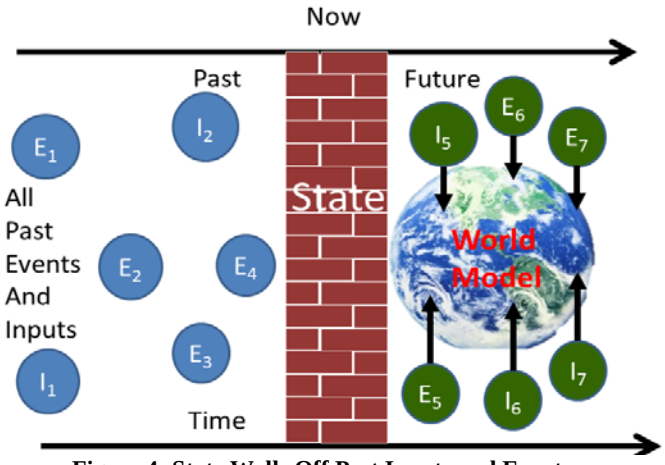


Figure 4: State Walls Off Past Inputs and Events

useful because it limits the amount of memory needed by the simulator and it prevents having to consider an ever expanding body of past states and inputs for each new update. It is also useful because it transforms the task of initializing the simulator from specifying the significant past of each object in the simulation run to specifying a single possible world, usually a more tractable task. Like the system boundary separates the target system from the remainder of the world, state separates the target system from the past.

State and Objects

What is state and how can it perform this function? Essentially state is system memory. However, it is not a literal recording of past events, inputs, and possible worlds. Instead, it is stored in the configuration or states of the objects in the model. The values of these states serve as restrictions on their values in future possible worlds. They include physical states, such as position, velocity, and damage, as well as abstract states like orders, goals, and when and where I told my wife I was going to meet her today. Using the values of these states and the inference rules in the

world model, the updater can calculate how much to change each state variable in the next virtual world. For example, using my position and velocity together with knowledge of the road network and local traffic allows the updater to make a pretty good guess as to where I am going to be in the next possible world. It is only rarely that I suddenly change my mind and turn around and drive to a local store; perhaps because I forgot that I was supposed to pickup a chicken for dinner. The fact that I forgot the chicken would have been established as state in a previous possible world and would have propagated through the states of all the worlds between then and now. A model of my brain would include some state for the 'realization that I was missing the chicken' to randomly attempt to pop into my consciousness and finally when I was almost home it would make it through. These kinds of events inject unpredictability into the simulation.

The road network is not considered state information in most simulators, but is instead part of the parametric data. However, I have just used it to compute the configuration of the next possible world. What distinguishes state from parameters in a simulator? The primary criterion is change. If we allow the roads to be changed during the course of a simulation run, then we are likely to call them state, but if we don't provide a way for them to change, then we are likely to call them parameters. The line between state and parametric data is flexible and is based on what we decide to hold constant.

State in simulators is not just a set of unrelated statements. It is organized around entities called objects. Objects are implicit in the way we think about the actual world and they are carried over into the way we represent the possible worlds in our simulators. A simple statement is composed of a subject, predicate, and grammatical object. The subject is the possible worlds object to which the statement refers. The predicate is the particular property the statement is specifying and the grammatical object is the value of that property. Grammatical objects can have literal values like 3.14159 or "blue" or they can be references to other possible worlds objects. Thus the state of a possible world is a graph with objects and literals as nodes and predicates as links between objects and between objects and literals. For example, a literal statement may have the subject 'car54', the predicate 'has color' and the value "blue". A relational statement might be 'blue248', 'is shooting at', 'red396'. Explicit formulation of models as networks might open up some alternative ways of implementing simulators. Statements in mathematical models are referred to as state variables. In object orient programming, they are expressed as a software object, member, and value. Logical statements are more expressive than mathematical state variables and software objects because they take on truth values and can be declaratively combined into complex compound sentences. In programming, such logical constraints are expressed in program statements. Research into the computational complexity of reasoning has proved that computational complexity and expressiveness are strongly correlated, so while you might want to describe something in a very expressive language, you may not want to compute in that language (Nardi & Brachman, 2007). In particular, queries on some languages are not guaranteed to terminate.

The Necessity of Objects

Is there an alternative way to structure simulator state than our implicit concepts of objects? Clearly, the possible worlds model requires some sort of objects, otherwise, what would our statements refer to? But are you, me, cars, tanks, and planes necessary? There is a form of simulation that does not use our familiar displaceable objects. It is called artificial life or cellular automata (Wolfram, 2002). It is highly abstracted and instead of changing the positions of objects in time and space, it changes the properties of fixed pieces of space itself. It is very similar to the voxel approach to computer graphics (Accomazzi, 2013). Conventional computer graphics uses our common concept of objects to break up the visual scene. Objects are represented by 3D models made up of many different triangles to approximate the external boundary of the object. Colors and textures are assigned to these triangles and a graphics engine calculates the projection of these triangles onto our computer screens to produce an image of the object.

Voxels use a fine three dimensional grid to subdivide space. Each cell, or voxel, in this grid takes on properties depending on the substance that occupies it. If skin occupies one voxel, that voxel gets the properties of skin. If bone occupies another voxel, then that voxel gets the properties of bone. If bone and muscle overlap in the same voxel the substance occupying more of the voxel wins. This approach has many advantages for medical imaging where the images represent solids with heterogeneous interiors such as CAT scans. Reproducing this with triangular polygons becomes unwieldy since each substance boundary must get its own set of triangles. This is a representation without classical objects; there is no cat, just cat substances occupying space. In cellular automata simulation, the properties of cells change based on their properties and those of their surrounding cells. Conservation of mass can be

maintained by ensuring that the number of cells of each substance remains constant. Length and width are not so easily handled. As an object changes angle with respect to the world coordinate system, the projections of its length change. If an object is aligned with the x axis and its projection on the x axis is 10 voxels and the object is rotated 45 degrees, then should its projection still be 10 voxels as conservation of mass would indicate or should it be 7 voxels long to preserve the length of the object in its own coordinate frame? Either the length or the density of the object has to change. Changing density requires coordination between a particular cat's voxels. Maintaining shape is also problematic.

Voxel simulators would require that all the common objects we perceive arise from the interactions and relationships of the properties of individual voxels. The challenge in this "nonobjective" simulation is how to make all the properties of common objects emerge from the interaction of spatially distributed properties without explicitly modeling those relationships. I don't know how to do that without the use of some concept of the form "these voxels form a cat and should work together to act like a cat" which is our old cat object reincarnated. We know that all the objects that we perceive in the real world are composed of molecules, which are composed of atoms, which are composed of electrons, protons, and neutrons, which are composed of even more fundamental particles. We know that our objects and their properties are emergent from the interactions of these particles. The solidness of a table top is an emergent property. However, we can place things on top of it very reliably. From the simulation point of view, objects are abstractions we can hang emergent properties on without the need to explicitly represent the subcomponents whose interactions produce them. We routinely simulate military units, commercial companies, and governments as objects without explicitly simulating the people and materiel that compose them. There are interesting parallels here with philosophical questions of identity (Olson, 2010). How do we remain ourselves when our cells are all gradually replaced? As will be discussed in the next section, objects are mechanisms of abstraction.

Our current cellular automata simulations produce interesting patterns and can resemble processes such as the growth of bacteria in a dish, but they can't replicate the experiences we have of interacting with objects in a 3D world. However, even current cellular automata simulations are capable of generating long term stable shapes that move across the simulation space in ways that make them appear to have the properties of objects even though there is no concept of an object in the underlying properties or update algorithms. Stephen Wolfram, the creator of Mathematica (Wolfram, 2003) has postulated that this is the true nature of reality. He is not alone in suggesting that the universe is digital.

State, Perception, and the World Wide Web

The Markovian property of simulation has almost magical implications for simulation. As long as we can set the initial state of a simulator correctly, we do not have to worry about specifying the past. This is great for systems where we can identify what the state variables are and there are a reasonable number of them. For example, in a simulation of a pool table, we can assign a velocity, position, and spin to all of the balls on the table and our initial conditions are satisfied. We don't have to worry about who hit the cue ball, what sort of cue stick was used, how hard it was hit, what collisions has it experienced and so on. However, in the case where we have memory elements whose states cannot be explicitly identified, such as in humans, or cannot be easily computed, such as algorithms that learn from past inputs like neural nets, the explicit assignment of initial conditions can be impractical and it may be necessary to start the simulation at a point before the desired start time so that learning systems and even simple recording systems are properly initialized. In discrete event simulation, there is also a burn in period during which the initial conditions are forgotten.

We can provide human participants in a simulation with a context briefing that is typically called "The road to war" in order to initialize them. However, providing simulated humans with appropriate memories becomes more difficult as our models of humans get more sophisticated. As organizations utilize more information technology it becomes harder and harder to initialize simulators of these organizations. We can't imagine the world today without the World Wide Web and the vast collections of data it contains. As I write this paper, I am constantly running web searches. Similarly warfighters use the web technology and large databases to figure out how to respond to enemy challenges and natural disasters. All that data is now a required part of the initial state, especially for staff exercises where the work of the staff requires the use of web technology. We do not have ways of creating equivalent databases for hypothetical worlds, so we must provide access to real databases or copies of them and base our initial conditions on the actual world. Real databases must now be treated as inputs from the past that we can't control but

must monitor and review to ensure that the simulation boundary remains intact. A consequence of this is that realistic staff training scenarios are much easier to construct in the context of real world opponents than fake opponents.

SIMULATION IS ABSTRACTED

We have walled off our analog from the rest of the world by not representing parts of the actual world and by requiring all external influences come through carefully monitored if not controlled inputs. We have also closed off the back door to the past, albeit we may have to allow monitored access to the past. Now it is time to look at the floor. Very much like state insulates the simulator from the past and inputs allow virtual walls to be built separating the simulator from the external world, abstraction via the choice of our base objects and their predicates provides the floor of the simulator's representation. Since these objects are state, state does double duty in walling the analog from the target's environment. It limits how deep the simulation goes. This is an extremely important and confusing area of simulation where intuitions can lead us astray. Abstraction is not just a requirement of our finite simulation resources; it is also a requirement of our finite knowledge.

The Necessity and Unpopularity of Abstraction

It seems like a common perception is that abstraction is bad. The more detail the simulator includes, the happier the sponsor tends to be. People tend to trust a detailed explanation more than a terser one. They intuitively expect that it is going to be easier to find contradictions in a false explanation if more details are provided. Another factor is reuse. Removing an element of a simulator reduces the number of problems that the simulator can address. The feeling is that the more applications the simulator can address, the more it can be used and the lower the life cycle cost will be. This factor obviously influences the boundaries of the simulator but can also influence level of detail. If the simulator is going to drive virtual interfaces, adequate detail is necessary to support the required human interactions and produce the required suspension of disbelief. This is an especially relevant topic as we attempt to replace live training with computer simulated training. Then there is the problem of the critic. At any time, a critic can pop up and ask if you considered the effects of X. It is reassuring to have all your bases covered. Sensitivity studies are a recommended way to determine what needs to be simulated. However, to do the sensitivity study, you need to model the detail, so you still have to do the work.

However, the incorporation of more detail is troublesome in many respects. The cost of simulation increases at a faster than linear rate as detail is added. The update of one statement depends on the values of many other statements. Adding one additional statement, not only adds the cost of updating that statement, but it adds to the cost of updating other statements. The amount of underlying data required to support the simulation also grows rapidly with detail. Adding a level of detail requires additional data and code for each object class extended. It also requires additional data for the interactions between objects. It can be cheaper to build multiple specialized simulations than to build one general simulation.

Understanding of the simulation is at least inversely proportional to its complexity which generally is positively correlated with detail. The addition of more detail also increases the number of possible worlds that can be generated by the simulator making each one less probable. Agent based (Miller & Page, 2007) and cellular automata simulations are subfields of modeling and simulation where simplicity is a fundamental part of the approach. Agent based simulation is not semi-automated forces or a constructive simulation modeling people. This flavor of simulation attempts to understand complex phenomena observed in the actual world by recreating them with the simplest digital model possible. Complex phenomena are defined as emergent behaviors that arise from the interactions of elements that are not programmed to produce those behaviors. Agent based simulation takes a simulation which produces the emergent behavior under study and strips away detail to get the minimum model that still produces that emergent behavior.

Measurement of Abstraction

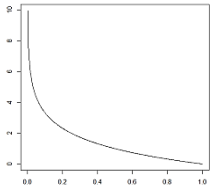
So far we have discussed abstraction in, well, very abstract terms. Can we measure the level of abstraction and compare different simulations along a dimension of abstraction? There are three well known levels of abstraction used in the combat simulation world: aggregate level, entity level, and engineering level simulators. But they are of

no use comparing simulators within a level and the classes themselves don't have strict boundaries. Is a simulator that presents entities but moves them around as a template an aggregate or an entity level simulator? How many components of a tank does a simulator need to represent before it becomes an engineering level simulator? The categories are informal and provide no guidance.

However, we clearly have some intuitions about what abstraction means, so I am going to run with my intuitions and see where they take me. I am also going to consider resolution which is inversely correlated with abstraction. However, there is one intuition that I am going to resist: the claim that resolution is only a function of classes; not the number of instantiated objects. This sort of argument can be used to claim that individual flight simulators have much more resolution than a battalion of low cost tank simulators. However, individual simulators are not able to simulate integrated activities and therefore lack the 'resolution' to represent the dynamic relationships between instances. A simple thought experiment might help show the problem. Let's say the aircraft simulation has 10,000 statements to describe its state variables and rules. Assume the tank simulators have 500 variable statements and 500 rule statements. If we consider a battalion of 56 tanks and ignore all other entities in the battalion, each tank adds its 500 variable statements to the single set of rules to get 28,500 statements for the battalion.

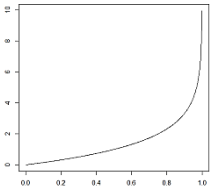
My intuition about abstraction is that it is characterized by how many fewer details there are in the model than in the real world. Similarly, resolution is related to how many details are retained in the model. Assume that there is something called a detail and it is finite and can be counted. Let n_T be the number of details in the target and n_A be the number of details in the analog. Therefore, the number of details dropped is $n_T - n_A$. To define measures for abstraction and resolution, I need to look at what I want my measure to look like at the extremes of the ranges for n_A, which are 0 and n_T. For A, our measure of abstraction, I would like to have infinite abstraction when the model has 0 details and have 0 abstraction when the model has n_T details. A possible measure for abstraction is given in Equation 1 with the plot of A versus n_A/n_T shown next to it.

$$A = \log_2\left(\frac{n_T}{n_A}\right) \quad (1)$$



For resolution, R, I would like to have infinite resolution when the model has n_T details and zero resolution when the model has no details. A possible measure for R is given in Equation 2 with a plot of R vs n_A/n_T shown next to it. Complete abstraction is no resolution and perfect resolution is no abstraction. The actual choice of functions is a matter of mathematical convenience.

$$R = \log_2\left(\frac{n_T}{n_T - n_A}\right) \quad (2)$$



There is still the problem of those pesky details. What could they be and how do we count them? Since a model is an information object, I look to information theory and perform some hand waving. An important consideration in information theory is how many possible symbols there are. For this problem, that turns into the question of how many different configurations can the model take on? This is the number of possible worlds in the model, which is the product of the numbers of values each state can assume. Then there is the number of configurations in the real world system. That is harder, but there has been a lot of work relating information theory to thermodynamics so perhaps there is some hope for absolute measures of abstraction and resolution. However, I can cancel out the number of configurations of the real world target by taking the ratio of the resolutions or abstractions of two models. This agrees with Moon's (Moon, 2013) observation that it only makes sense to evaluate the relative resolution of

two simulators with respect to the same actual world targets in the context of a particular scenario. Otherwise it would be impossible to cancel out the number of real world configurations.

Now that we have a sketch of a way to get the relative resolution of two models, what can we do with it? Unfortunately, there is still a lot of work that remains to be done. My measure does not consider the likelihood of different configurations in the model. It does not consider if all the states are independent. If they are not, some possible worlds will never occur. It does not assess whether the states and rules represent differences in the actual world. It does not consider the rules at all. We would also like to be able to generalize the result away from a single scenario. This is difficult if we require relationships between objects to count in resolution. However, we saw in the previous section that object properties are just ways of explicitly representing emergent relationships in their constituent objects.

Furthermore the overall resolution of the model doesn't say anything about its resolution in a particular subcomponent. With the exception of models of uniform collections of particles, even a single simulation designed and built by a single programmer has multiple levels of resolution. Consider the two tank models in Figure 5. On the left is a decomposition for a tank driving simulator and on the right is a gunnery simulation. Assuming that the displayed objects are of roughly equal complexity, it appears that they have roughly the same resolution. But the resolutions of their subcomponents vary dramatically. These two trees could be merged. In that case, objects could be instantiated at different resolutions. Now the resolution and abstraction of the simulator would not be the same. The abstraction would be determined by the highest point in the tree that an object could be instantiated at and the resolution from the lowest. (Davis and Huber, 1992) have suggested that families of models built at different resolutions would be advantageous for theater and operational level combat analysis.

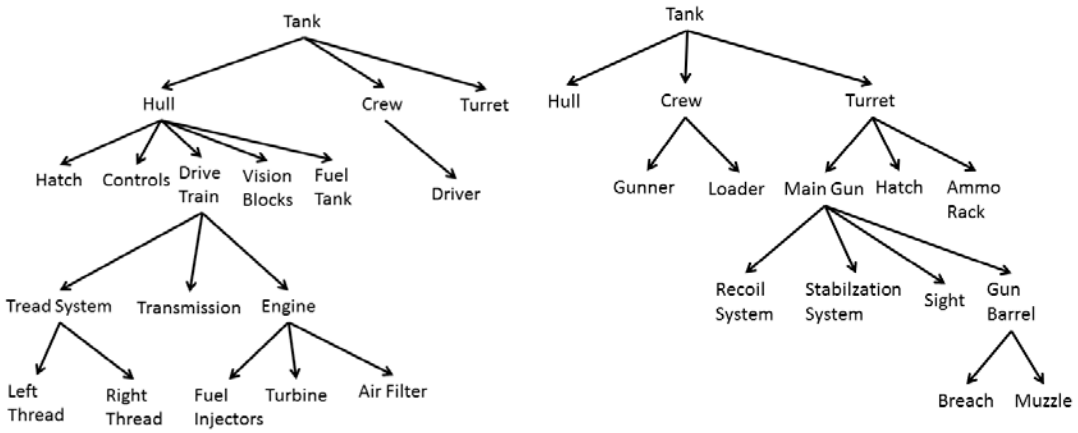


Figure 5: Models include Multiple Resolution

One benefit of using physical and human analogs is that since they are actual world systems, the dynamic and mental properties of the real world are included automatically. A scale model has properties of mass, drag, sound reflection, and electromagnetic reflection. Getting those parameters to correspond properly to the target may be difficult, but the physical properties are there. This stands in sharp contrast to symbolic models which need to be constructed from nothing and have no properties beyond those that are added to them or emerge from them. This often comes as a surprise to customers who draw up their requirements and assume that because they specified one property they are automatically going to get all the properties that normally accompany it in the actual world. I like to refer to it as the 'Oh, you wanted a car with wheels' problem. Nothing in digital simulation comes for free. Emergent simulator properties are especially difficult to match to those in the actual world.

Take the case where a customer specifies a requirement for bad weather to cause icing on an elevated bridge. The customer comes back and watches a spectacular sleet storm which leaves the bridge glistening with ice. The customer is extremely pleased, congratulates the team and watches as a truck comes down the road and turns onto the ice covered bridge and flies off the side of the bridge. The developers beam. "What was that?" asks the

customer. The developers look at him blankly. "The driver has to slow down under icy conditions", he continues. "Oh", says the weather developer, "you'll need to fund the agent team to add logic to the driver to recognize ice and slow down." The problem is fixed and the customer then brings his boss to see the demo. His boss says "If we had that kind of storm, we would always put tire chains on" and the process repeats. The point is that simulations are built for a purpose; there are never enough resources and the requirements always change. As a result, only the highest priority requirements get implemented and the resolution inside the simulator varies wildly. This is not just due to a poor customer developer interface; it happens when the customer is the developer. Emergence can't solve the problem, except under the most abstract of interpretations, because the specific chain of emergence, from the change of state of water under changes of temperature, up to the application of chains to tires to prevent slipping, would be incredibly hard to replicate.

SIMULATION AS INTEGRATION

An essential characteristic of simulation is that it is incremental. Simulation does not go directly from the possible world specified by the initial conditions to some target possible world, say a world ten days from now or a world where the war is over. Generating a new possible world in a single step is the characteristic of an analytical solution. Analytic solutions and simulation solutions are mutually exclusive. Instead, simulation breaks the generation of the target possible world into the problem of generating a sequence of possible worlds that incrementally get closer to the target possible world. The incremental changes are added up, i.e., integrated, to produce each successive possible world. There are many reasons for doing so.

- We may not have any particular target world in mind; we just want to observe how the results of some actions evolve, so we set the initial conditions, apply any inputs of interest and look at the output to see what possible worlds emerge.
- We may not know how to get to the target possible world, so by applying incremental changes each designed to bring us closer to the target world, allows us to determine on each step whether we are getting closer or not.
- The process of generating a new possible world becomes easier when the difference between the current possible world and the next is small.
- We may not have the resources to go from the initial possible world to the target in one step. If we only have a serial process, we can only update the world a single statement at a time.

The third reason is very important. It is why simulation can be used to solve nonlinear problems when analytical solutions fail. In the interval of a sufficiently small time step, each object can be updated and each event evaluated while holding the rest of the world constant. I suspect that it makes simulation universal because it mirrors the way our universe works; an accumulation of infinitesimal simultaneous changes.

The last reason makes the integration of many small steps a necessity for symbolic analogs. Simulation with symbolic analogs is inherently serial because even if there are multiple computation streams available, they are naturally decoupled. Our physical reality[2] is massively parallel. Every atom is moving at the same time. It is also very tightly coupled with every particle affected by the forces of other particles. Thus the world is also massively interactive. So we have the fundamental problem in symbolic simulation of mapping a parallel world onto a serial stream of computation. The only ways that you can do this are by timesharing or parallel processing. Fortunately, because forces between atoms make them coalesce into molecules and molecules coalesce into solid objects where atoms maintain fixed relationships with each other, we don't have to timeshare the movement of atoms when simulating on a human scale. We only have to time share the movement of objects, of which there are far fewer.

As soon as you start to time share, the correspondence between real world and simulation time is shattered. We now have two different clocks, simulated and real time. We take one object at a time and move it over an interval of simulated time. There are many different variations in the order, timing, and computation of updates which I will ignore in this paper. Let's assume the simplest strategy of equal fixed time updates with the objects selected round

[2] I am not making a metaphysical claim here; rather I am trying to describe the difficulties of mapping what appears to be parallel universe onto serial processing streams.

robin. We now have an iteration pattern here that is occurring not in simulation time but in real time in object order. This is a common pattern in simulation.

How long should the update interval in simulation time be? Let's try to set it to the final simulated time requested, so that we only have to move each object once. This would essentially be running a completely independent simulation for each object. If it worked, it would be the most efficient approach with only one context switch required per object. The problem is that the world is not only massively parallel, it is also massively interactive. In order to figure out how to get the first object to the end time, we also have to figure out whether it has any interactions with any of the other objects in the simulation. In effect, we still have to do the entire simulation just to move the first object. So this approach loses.

Next, we decide to try the opposite approach and we pick the smallest time resolution on our real time clock. At the end of each cycle, we will check all the objects for collisions. This sounds good. During each update we don't need to worry about collisions and the computations are straightforward. Unfortunately, while we wait for the simulation to finish, we can see that the value of simulation time displayed on the screen is barely moving. The time tick is too small and the computer is spending all of its time checking for collisions.

Finally after a careful evaluation of the separation between the objects, their sizes, and their speeds, we set a new update time which will ensure that we can detect collisions but is no smaller. Now the simulation runs to completion. In the general case, many different factors will need to be considered in the choice of a time step, such as forces at a distance and the response times required to keep feedback loops stable. Some simulations use variable time steps based on the current state. The use of a small but appropriately sized time step for updates is a magic bullet for simulation. An appropriately sized update time decouples the simulation. It allows integration of complex functions to be performed using constant or linear approximations within the interval of the time step. While we update one object, the rest of the world is considered to be constant. However, it is large enough to allow the simulation to complete in a reasonable time.

SIMULATION AND PARALLELISM

There are sharp distinctions between physical, human, and symbolic analogs. As concrete elements of the real world, physical analogs are inherently parallel, stepping a simulation run with a physical model would be extremely hard if not impossible and in all likelihood will make the run invalid. Instead, physical analogs require instrumentation and sampling. The use of a very high speed camera, for example, gives us a sequence of images, i.e., possible worlds, while a bullet moves through an analog of a target. Even in cases where we get continuous output from the sensors used for instrumentation, the sensors always have some limit in their temporal resolution. So we are still getting snap shots. Ideally the speeds of the dynamics of the analog are much less than those of the instrumentation. So with physical analogs, we capture possible worlds through sampling.

With human analogs, say a battalion rehearsing a mission, everything happens continuously and in parallel as in the actual world. However, it is quite possible to stop the rehearsal and discuss observations and restart at a previous point in the simulation. Instrumentation here may include films, but it is primarily recorded in the minds of observers and the participants. Time jumps are possible, skipping over segments of the mission that involve transportation or waiting. This works because the human mind has to capability to simulate the gaps and reinitialize itself at the point the rehearsal restarts. The inherent parallelism of the world is retained in the analog. However, the descriptions of the event will be serial sequences that consist of first serially describing the state of the initial conditions, i.e., possible world, followed by sequence of descriptions of individual events corresponding to changes to that possible world. So the descriptions that people can provide are sequences of possible worlds; each one differing from the next by an event. Combining their possible worlds into a single narrative is an act of induction on possible worlds.

Analog computers provide true parallelism, but even at their height, they were only capable of computing on the order of a hundred variables in parallel. With symbolic analogs, parallelism has to be built, it does not come automatically. A mental simulation is strictly serial. I can't solve two equations at the same time. Digital computers have a single stream per processor and a typical laptop has two or four processors. Time sharing is used to get around these limitations and approximate many more processes. Supercomputers have been built with thousands of

processors. However, even supercomputers typically employ time sharing for simulation. These supercomputers have many different configurations and their structures have been varied and hybridized in every conceivable way. However, for this analysis, I will only consider three types of parallel computer architectures, vector processors, multiple processors with a shared memory, and distributed systems built out of single processor computers with individual memories connected via a network. Furthermore, I will consider three parallel programming approaches that I feel arise naturally in those configurations. They are all Turing complete, which means that any simulation that will run on one can be ported to the other. They are distinguished by the problems encountered in programming them. Any simulation on a parallel processor will use some combination of these three approaches.

Vector Processing Approach

A vector processor is also known as a Single Instruction Multiple Data processor (SIMD). They are used when there is a lot of data that needs to be processed by exactly the same series of instructions. A section of code used to independently update the position of a set of homogenous non-interacting objects is an example. Instead of updating one object at a time, you could update multiple objects at the same time. The sequence of instructions used must be exactly the same and the instructions required must be available as vector operations on the particular vector processor being used. Other sections of the code with individual branching requirements must be run as a normal single instruction stream - single data stream style of computation. From a programming perspective, the primary difference from a single instruction stream simulation program is that the programmer has to identify those areas where vector processing can be applied. Vector processing parallelism is used in weather, car crash simulation, and graphics processing.

Shared Memory Parallel Processing Approach

This architecture is characterized by fast equal access to all the simulator states. The processing style that arises from this organization is called fine grained parallel processing. It is characterized by the distribution of tasks from a central event list. Each processor goes to the event list to get its next task when it becomes idle. The advantage of this approach is that it is very easy to keep everything coordinated in time. Let's go back to the problem of updating the position of a set of objects while detecting collisions. Because each object's state is equally accessible to all processors, there is no reason to limit the update of any particular object to a particular processor. If there are n objects being simulated, you put n events on the event queue; each event is the update of one object. The processors each pull an event off the queue and update the corresponding object. If the objects are heterogeneous and one particular object takes an inordinately long time to update, the other processors will end up updating more of the remaining objects. The process is inherently load balancing.

Next we have n (n-1)/2 events to check for pairwise collisions. When a processor picks up one of these tasks, it has to gain access to two objects to run the algorithm for this event and it has to know that both objects have been updated. Now two processors may want to access the same object. Assume that when a processor updates an object, it locks it and the other processors cannot access it. Otherwise, a processor could get inconsistent state if was reading while another process was writing or the state of the object could become inconsistent if two processes were writing at the same time. So now a processor with a collision detection task has to check that its target objects have the right time and that they are not locked. If one of them is locked or has not been updated, one of two strategies is available to the programmer. He could have the processor put the task back on the queue a bit further down or have it wait for the resources to be freed up. Placing the task back on the queue can be tricky. Ideally it would be placed in the position where it would cause the fewest additional resource conflicts. Barrier tasks can be used to hold up the execution of subsequent tasks until all the previous tasks have completed (Fugimoto, 2000).

The fundamental problem in parallel programing is to decompose the problem into tasks that minimize resource conflicts. Processors have to wait for each other where the algorithm becomes serialized. If two processors have locked the resources that each other needs to execute their events, then the program deadlocks. Debugging in this approach becomes harder for each processor added to the system because the programmer has to check what was happening in each processor when the deadlock occurred. A pairwise deadlock rapidly turns into a deadlock of all processors so you have to figure which was the first deadlock. Resource conflicts also increase with the number of processors. The processors have to be collocated to get high speed access to the memory.

Today large scale shared memory parallel processor super computers have largely been replaced by Scalable Parallel Processor (SPP) clusters of commodity servers organized by cluster operating systems, such as Beowulf. These clusters are really distributed processor systems. However, the cluster operating system makes them behave more like a shared memory parallel processor by providing a central control processor that assigns tasks to each slave processor in the cluster. The cluster does not have the high speed communication of shared memory, but may use special high speed network connections.

Distributed Processing Approach

Distributed parallel computation was developed with the assumption that each processor has its own memory and that communications between processors are fairly low bandwidth. It uses the course grain parallel computing approach where coordination between processors is far less frequent than in the fine grained approach. In distributed simulation, the shared state variables are duplicated in each processor's memory. Each processor is responsible for the update of a portion of the total state, termed the local state, while the remainder is termed the remote state. When a processor updates its local state, it uses the network to send out messages informing the other processors how its local state has changed. This causes the other processors to update their remote state. If a processor simulates an event that could cause a change in state for a remote object, it sends out a message containing an interaction. This interaction is then interpreted by the owner of the remote object to determine how to change that object's state.

This approach was introduced to simulation in the 1980's by the DARPA SIMNET program (Thorpe, 2010) and then turned into a standard called Distributed Interactive Simulation (DIS) (Neyland, 1997) by the Simulation Interoperability Standards Organization (SISO). What makes this distributed simulation approach more appealing than fine grained parallelism is that it converts the parallel programming problem into multiple independent single processor programming problems. The local state variables of the remote simulators are converted to inputs in the local simulator. So the resulting simulator ends up looking very similar to a serial processor simulation program with the addition of some network routines for exchanging object updates and interactions. The same mechanism that is used in serial simulation to isolate the system being simulated from the rest of the world is used in distributed simulation to subdivide the simulation problem. There are no resource conflicts. This considerably reduces the complexity of the simulation programming task.

The major problem that has to be addressed in this implementation is the synchronization of time between these processors. Unlike the parallel processing approach, there is no central event queue and automatic load balancing in the distributed processing paradigm. There is no central concept of time. Each object is always updated by the same processor. So processors that are heavily loaded update their objects more slowly than those that are lightly loaded. If two processors with different loads are advancing time as fast as they get done with their tasks, they will get out of synch. So some sort of mechanism is required to synchronize time to preserve simultaneity. The more precise you require this synchronization to be and the more processors that are participating in the simulation the harder it is to get good performance from the system. A particularly challenging case is the implementation of fully repeatable simulation when any object can interact with any other object at any time. In some cases, the distributed simulation's time advance can become serialized with each processor taking a turn updating while the rest wait idle. Load balancing can help but the transfer of objects from one processor to another can be very slow when the state is large. Too many object transfers can be worse than none.

Because SIMNET was a human in the loop, first person simulation, the approach was to use real time for synchronization. Each simulator node has its own local clock and those clocks track wall clock time, so the updates in each processor are tied to wall clock time. Because some computer clocks may drift, the Network Time Protocol was used to keep clocks in sync. Between distributed sites, time can be synchronized by using GPS receivers. So in DIS, time is synchronized by an entirely external mechanism manifesting itself as the local clock in each processor. One can call this a passive time synchronization approach because the simulator is not involved in time synchronization. This passive time synchronization approach can support faster and slower than real time operation but all processors need to use the same scaling factor. Simulation messages can be used to tell all the simulators to change the scaling factor and offset, so the time mapping can be dynamic. However, these capabilities are rarely used; people and C4I systems don't work well with scaled time and time jumps. A primary problem is the lack of a training mode in most C4I systems, which would allow non-wallclock time advances. This can be a major limitation in computer exercises. For constructive simulation, the tendency is to use active time synchronization algorithms

based on simulation messages. The High Level Architecture (Kuhl, Weatherly, & Dahmann, 2000) brought these time synchronization mechanisms (Fugimoto, 2000) to distributed simulation in the late 1990's.

Interest Management

The second major problem in the distributed simulation paradigm is state duplication. It is straightforward to see that distributed simulation as described in the previous section cannot scale. Assume a simple distributed simulator composed of identical federates where the cost of maintaining one remote object is 0.1% of the power of the processor. Further assume that the cost of maintain one local object is 1%. Running standalone the processor can simulate 100 objects. The maximum number of objects that can be simulated with N processors (assuming that processing power is the limiting factor) is shown in Table 1. Clearly you cannot scale if every object you add to the simulation has to be maintained by every processor. The same is true for memory and communications bandwidth. However, you only need to keep track of an object if you are going to use its state to update the state of your local objects or if you need to interact with it. Interest management is the process of limiting the delivery of irrelevant remote state information to each processor.

Table 1: Objects that can be Simulated with N processors

N	Local Objects	Remote Objects	Total
1	**100**	**0**	**100**
2	**90**	**90**	**180**
11	**50**	**500**	**550**
31	**25**	**750**	**775**
91	**10**	**900**	**910**
491	**2**	**980**	**982**
991	**1**	**990**	**991**

The HLA introduced two mechanisms for interest management: subscription by class, Data Management (DM), and subscription by value, Data Distribution Management (DDM). Data management is useful for eliminating objects that you will never need. DM provides a publication/subscription interface (Eugster et al, 2003) that will only deliver the object state variable updates and interactions that a processor subscribes to. DDM adds a region to the subscription that defines an additional filter which uses dynamic routing parameters to determine which updates to deliver.

The initial implementation of interest management in distributed simulation was multicast (Calvin et al, 1995). This approach was first employed on a large scale in the DARPA Synthetic Theater of War Program (Budge et al, 1997). However, even enterprise hardware has limited multicast capability. Simulating very large numbers of objects required switching the implementation of interest management from the network level to the application level; performing interest management routing on dedicated processors. In 1998, the DARPA SAF Express program (Burnett & Gottschalk, 1998) used ModSAF, a DIS distributed simulator, running on High Performance Computing Modernization Program (HPCMP) assets to simulate 100,000 vehicles using application-level interest management. U.S. Joint Forces Command (USJFCOM) J9 extended this work to the Joint Semi-Automated Forces HLA distributed simulator federation (Lucas & Davis, 2003), (Helfinstine, Torpey, & Wagenbreth, 2003). While interest management based on sensor range worked well for human and conventional sensors, the rapid development and fielding of high resolution wide area sensors also required the distribution of sensor modeling workloads (McGarry & Torpey, 1999). Between 2003 and 2006, USJFCOM and the Institute for Defense Analysis Joint Advance Warfighting Program ran a series of exercises, named Urban Resolve (Ceranowicz & Torpey, 2005) (Williams and Smith, 2007), experimenting with the use of advanced sensors in urban combat. By applying even more restrictive interest management to vehicles driving on roads, the Urban Resolve simulator federation running on HPCMP assets was able to simulate around 300K entities in production exercises and eventually reach 1,000,000 entities in testing.

In simulations where the only simulation objects of interest are those in the vicinity of a particular object, such as the player in a first person shooter game, it is possible to apply an extreme form of interest management; where trees in a forest with no one to hear them fall, never fall. That is, the only objects and events simulated are those that can be

or might possibly be perceived by the player. This can be thought of as a simulation with a moving boundary and objects and terrain are injected into the simulation as inputs when the boundary moves. Curiously, as physicists investigate quantum physics, we are learning about phenomena that are pretty hard to explain and bear some resemblance to this kind of extreme interest management (Johnson, 2014). Scientists have come to believe that particles such as photons and even electrons don't always exist as particles. Instead, they exist distributed over space in a probabilistic wave function and don't really turn into particles until we measure their properties. Observation collapses the wave function. This is like a player in a first-person video game bringing monsters into existence from a probability function by his presence. Perhaps we are living in a simulation (Bostrom, 2003) and as we explore deeper into the nature of our universe, we are starting to see behind the curtain. There are significant research efforts currently searching for a way to build quantum computers which can use the natural parallelism of the quantum wave function for computing.

Composition

Because distributed processor simulations are very similar to individual processor simulations with extra inputs, it is possible to independently create different simulation programs and then hook them together. The individual simulators that are used to create a composite simulator are often called federates and the composite simulator is often called a federation. The federates must share at least the following assumptions for this to work. They must share a common object model that can express remote objects. They must share a common set of interactions that allow objects in different federates to interact with each other. They must use a common time advance scheme. The HLA provides a standard to guide the construction of these kinds of simulators and an infrastructure for connecting them. Additional architectures for building distributed simulations include DIS, the Test and Training Enabling Architecture (TENA) (Noseworthy, 2011), and Data Distribution Service (Data Distribution Service Portal). The advantages of this style of simulator construction go beyond obtaining more processing power or parallelism. They extend to the process of building the simulator. The task of building simulators is difficult and expensive. The larger and more comprehensive a simulator is, the larger the expense. Distributed simulation allows the independent construction of heterogeneous simulators that can be a posteriori connected into different federations. This was the primary motivation behind the development of the HLA. It has been a very useful technology in enabling the construction of simulator federations. However, the connection of independently constructed simulators can lead to problems due to the different choices their authors have made in the process of reducing their targets down to analogs. It can be quite expensive to build federations a posteriori and fix major misalignments by reengineering federates. A great deal of energy has gone into the analysis of this problem (Davis & Anderson, 2003), (Tolk & Muruira, 2003), (Diallo, Padilla, & Tolk 2010), (Ziegler, Mittal, & Hu, 2012), (Petty & Weisel, 2003).

SIMULATION AND TIME

Thus far, time has been assumed to be an element of simulation. Most simulations are concerned with generation of sequences of possible worlds in time. The management of time and clocks has always been a significant concern in computer simulation. In this section, I examine the question of whether time is a necessary element of simulation or simply a very common one. The most common use of time is to pick a future time period and conduct simulation runs to create timelines of possible worlds that cover the period of interest. Alternative sets of runs can be carried out to reflect different strategies, variations in initial conditions or inputs, or to explore uncertainties in the world model. If the point of the simulation is to inform our choice of actions, then conducting simulations of the future is the only logical choice. Our actions cannot change the past. The future start time can be picked to correspond with the commencement of a future operation or campaign. If the time of the operation is not known, for example, if we are practicing a response to an attack, then a hypothetical future time can be used.

Simulation and Reenactment

Another application of simulation is reenactment. In the Battle of 73 Easting (Orlansky, Thorpe, 1991) during the first Iraq war, the U.S. 2nd Armored Cavalry Regiment encountered and destroyed significantly larger units of the Iraqi Army including the Tawakalna Division of Republican Guards. Over 300 Iraqi armored vehicles were destroyed and 2000 prisoners of war were captured. The battle was a unique accomplishment and Army Chief of Staff, General Gordon Sullivan directed DARPA to capture it in simulation using SIMNET technology. Unlike a movie, recording the battle as a 3D simulation allows future viewers to watch the event from any perspective

including from inside the U.S. and Iraqi tanks and even to experiment with possible alternative outcomes. However, this was not simply a recording of history captured by cameras and other sensors. Rather it was an active act of reconstruction via simulation.

To understand what happens in such a reconstruction, we need to consider the difference between subjective and objective probabilities. Subjective probabilities are based on our knowledge of the world, while objective probabilities are produced by the actual properties of the world (Johnson, 2013). For example, the objective probability of a fair die landing with the six on top is one sixth. Our subjective probability for that event is identical. But if the die is thrown and lands with three on top but is covered with a cup before we can see the outcome, then our subjective probability remains at one sixth but the event's objective probability is now zero. When we deal with future possible worlds our subjective probabilities should normatively follow the objective probabilities of those events. However, when we are dealing with current or past worlds all the objective probabilities are 0 or 1, since the events have completed. Our subjective probabilities now reflect what we know about the outcome rather than what the outcome actually was. If we have no knowledge about the outcome, our best guess is the objective probability prior to the outcome. So it was with 73 Easting. Knowledge of the event was incomplete. It had to be extracted from the distributed recollections of the battle's participants and the physical effects visible in the battlefield area. Subject matter experts were dispatched to the battlefield where they took photographs and made measurements capturing as much physical evidence as possible. Message logs were reviewed. A 3D replica of the battlefield terrain was constructed from the best available topographic sources. At the SIMNET site in Grafenwoehr, Germany, battle participants drove M1 simulators to integrate their recollections and recreate the battle while their movements were logged. Finally, ModSAF (Ceranowicz, 1994) was used to fit the available data within the dynamic capabilities of the platforms involved and create the final reenactment. The possible worlds that were created were the combination of the best estimates of the objective truth that could be produced from the objective probabilities provided by the physical evidence, subjective probabilities of the recollections of the participants, and the physical constraints expressed in the ModSAF world model. So simulation can be applied to past worlds to fill in or correct subjective statements which we use to approximate the objective statements that define the fixed reality of the past.

Simulation and Counterfactuals

Another use for simulation in the past is the investigation of counterfactuals. The use of simulation here is very much like simulation of the future. The simulation is initialized at a point in history where some important event is postulated to have an alternative outcome. Then the simulation is run forward creating a new history. Many novels explore historical counterfactuals and can be considered a form of mental simulation. A writer may envision how Europe might be different if Germany had won World War II.

Time and Monte Carlo

The most serious challenge to the requirement for time in simulation comes from Monte Carlo, i.e., statistical, simulation. Statistical Monte Carlo simulation dates back to the Metropolis Monte Carlo algorithm (Metropolis et al, 1953). The paper presents a sampling algorithm for determining the lowest energy configuration of several hundred particles. An initial configuration of particles is setup establishing the initial conditions. Then one particle is displaced randomly and a formula is used to determine whether to keep or reject the displacement. If we keep the move, the result is a new possible world slightly different than the original. If the move is rejected, the new possible world is excluded from the sequence of possible worlds that leads to the final one. The process repeats generating a sequence of possible worlds. This sequence is an ordering of possible worlds in real computer time as opposed to simulation time. Let's evaluate this process against our criteria for simulations.

- Purpose: find the pressure in a gas at equilibrium
- Historical: the start of statistical simulation
- Analogical: uses a digital analog
- Possible Worlds: starts an initial possible world and produces another on each accepted step
- System Boundaries: no inputs, scope limited to several hundred objects
- Markovian State: yes
- Abstraction and objects: each molecule is an object
- Integration: repetitively adds up small changes

- Time: uses real computer time, but not simulation time

This algorithm does not seem to meet the criteria for use of simulated time and the purpose. What about other examples of statistical simulation? (Braun & Murdoch, 2007) provide a very simple example of statistical simulation. To simulate taking a twenty question true-false exam, you can draw twenty samples from a uniform distribution with a range from 0 to 1 and test the result. False is returned if the value is greater than p and true otherwise. Even though time is not mentioned, it is used implicitly because the test taker goes through the questions sequentially. So if we allow the use of logical time (Lamport, 1978) for the last criteria, this simple procedure will satisfy all the criteria. In this case, logical time is simply an abstraction of simulation time. There is so little difference between this and the original case, that it is hard not to accept the former if we accept the latter.

Time and Finite Element Analysis

Finite element analysis is sometimes referred to as simulation. It is a numerical technique that partitions an object into small elements so that the forces in the object can be calculated (Ross, 2012). Points on the boundary between elements are chosen for calculation. The points are allowed to move based on the forces applied to them. The only exceptions are the boundary points, which are fixed. Equations for the relationship between the physical forces applied and the movement of the nodes are formulated as linear matrices. Independent external forces are applied to the directly affected nodes. Matrices are combined in pairs eliminating internal nodes until a solution for boundary elements is found. Then the stresses in the internal elements can be found. If the external forces are time varying, the external forces are moved by applying a time step and the whole processes is repeated. For the case of a moving external load, we clearly have time but it would be a better fit if the next iteration was calculated from the last solution instead of starting from the initial unloaded mesh state. But we do have a timeline of possible worlds.

For the static case, the problem is broken up into small pieces in space not time. Node point displacements are solved successively in space instead of time. It uses spatial consistency rules instead of temporal consistency rules. It is very reminiscent of what happens in a temporal simulation within a time step, where we update one object at time while the remaining objects are held constant. But where are the possible worlds? In most temporal simulations we start with a set of given initial conditions as the first possible world and generate the next one from there. In the finite element case, we do start with an initial mesh with each node having a position in this mesh, potentially a possible world. However, it is not a possible world because it doesn't seem to obey the internal consistency laws of the world. If we look carefully however, we see that this is a very special type of problem. The solution that is being obtained is the steady state solution that would be achieved in equilibrium. All the dynamics involved are ignored, so there are no temporal consistency rules. This is justified in cases where the time it takes for the system to achieve equilibrium is much shorter than the time steps required to adequately represent the movements being studied. We saw the same sort of thing in the Metropolis paper just discussed. The initial conditions given by the location of the nodes in the undeformed mesh could 'potentially' be achieved with the right set of external forces. Those forces could then be assumed to be removed at the time the finite element analysis is started. It is common in discrete event simulation analyses to ignore the initial burn in period to filter out any bias produced by the choice of initial conditions. When these problems have a unique solution at equilibrium, the choice of initial conditions is not important.

Next there is the question of why successive time steps get calculated from the initial undeformed mesh rather than the previous world. This also arises from special nature of the problem; it is an equilibrium problem and it is linear. This makes it possible to produce an analytical solution in time. The repeated insertion of time values into an analytic solution is how you get possible worlds timelines from an analytic solution. So we have is an integral solution in space, but an analytical solution in time. This reverses the solution order we are used to: analytic/sampled solutions between time steps but integral solutions in time. The distinction between analytic and simulation solutions was one of our earliest assumptions. This result suggests that there might be a symmetry in solution approaches across time and space. You can solve a possible worlds problem entirely analytically, entirely by integration, or using analytical solutions in space and integration in time, or using analytical solutions in time and integration in space. The implications for time from this result seem to be that while we might have a simulation in the static case, we definitely don't have one in the temporal case, so time cannot be a sufficient criterion for a simulation.

Time Concluded

Next consider an example of a class of problems which can be solved by processes that look like simulation but do not involve advancing simulation time. I will assume time is required for a simulation. The problem is reception seating. There is a list of guests and chairs for everyone distributed evenly at a fixed number of tables. The state is who occupies each seat. We also have a function for how happy each person is based on the people at their table and their relative seating positions. The initial condition is that everyone is standing. The solution algorithm goes through people in sequence and executes the following seating rules. If they are standing, they sit down in the chair that provides the best happiness level. They won't sit down if none of the seats provide a happiness level exceeding a threshold. If they are sitting and their happiness level goes below the threshold they go back to standing. The threshold decreases slightly each time a cycle of the guests is completed. The algorithm for calculating happiness and the rules for sitting and standing constitute the world model. The world model is executed until nobody is left standing and a seating plan has been produced. There is no time advance, so we do not have a simulation. Otherwise all the properties for simulation are satisfied. Now assume that instead of a reception with assigned seating, this is a reception with open seating. Now we have time and a simulation. It should not be this easy to go from a non-simulation to a simulation. Therefore I conclude that time advance is not a necessary condition for simulation. The advance of pseudo-time is sufficient. Other problems that belong to this class include the painting of a picture and the construction of the initial conditions for a combat simulation.

I have stretched my definition of simulation by rejecting the advancement of simulation time as a necessary criterion. The problem here lies in what else I have allowed into the class of simulation. I seem to have left the door wide open for many sorts of iterative processes or techniques as long as they fill in details in a possible world and require sampling. Most authors, however, seem to agree that the displacement of simulation time is required and would not admit the construction of initial conditions to the class of simulation. Hartmann provides an explicit rejection of my use of iteration over computational time to support the requirements for time advance and iteration:

> I shall mention that the term "simulation" is sometimes also used in the context of mathematical-statistical algorithms. A special example is the technique of so called Monte-Carlo-Simulations. It has often been emphasized that a Monte-Carlo-Simulation is in fact not a simulation at all. It is an effective numerical technique to tackle, say, complicated integrals. After all, this technique is often used in simulation procedures. In a way one can try to "save" the label "simulation" here: The problem of evaluating an integral with the Monte-Carlo method is dynamized in so far as one has to select successive random numbers, then calculate the value of the respective function, sum all the values up, do the same again for a new set of random numbers, . . . and finally average over all those preliminary results (Hartmann, 2005)

It seems Hartmann would exclude all statistical simulations which do not represent time dependent processes. However, the difference between a simulation that considers time and one presented in the 1953 Metropolis paper is very small. Hartmann is consistent with (Rasmussen & Barrett, 1995) who view simulation as a way of observing emergent phenomena produced by the interaction of objects. The concept of simulation at a single point in simulation time does not admit such interactions. Perhaps, interactions in pseudo-time are sufficient. The question for the emergent view is how does emergence as a criterion for simulation classify those temporal programs that are currently considered simulations? How many will need to be renamed? (Miller & Page, 2007) actually attempt to avoid the term 'simulation' because of their focus on emergence and minimalism, by calling their simulations 'computational models'. One problem with requiring simulated time in simulation is the question of how much time is enough. Let's say that a single time step is enough as I can see no support for choosing any other number of steps. Now in a discrete system, if I make the step smaller and smaller, at some point I will get no change. The supporter of time might say that's it, you need just enough time for one change. But then, what about a continuous system? There is no time step that is small enough to avoid a change. Any time step I choose is completely artificial and so is my criterion for simulation.

However, I have to agree with Hartmann that integration by itself does not seem like simulation. That is why I used Metropolis's second paper on Monte Carlo for this section instead of the first one with Ulam (Metropolis & Ulam, 1949) which is undisguised integration. Since I am trying to look for commonality between different forms of simulation, I am going to accept the generation of possible worlds in pseudo-time as a way of allowing statistical simulations that don't use simulated time and the painting of a picture to qualify as simulations. I hope that the

concept of creating a possible world can be sufficiently constrained to close the door on arbitrary iterative techniques. However, I will not attempt to do so in this paper.

SIMULATION IS NATURAL

It is easy to forget that simulation is anything but the province of digital computers. Virtually all other forms of physical analogs have been displaced by digital analogs and only exist as curiosities. Human simulation is not usually referred to as simulation. Any serious form of abstract simulation in science and engineering now utilizes computers and a great deal of time is spent is the creation, running, and analysis of simulators and their output. I would like to set that aside in this section and examine how well activities that people conduct every day fit into the model of simulation as the creation of possible worlds. This is just a cursory and speculative survey not based on psychological data.

I have already discussed various instances of human and mental simulators. So it is clear that, in my quest for a regularity underlying the 'big tent' characterization of simulation, I include activities of the conscious mind. The clearest example of simulation as a human activity is the hand execution of a symbolic model. Although the days of human calculators simulating ballistic trajectories are long gone, we know that there was a time where all simulations were executed by people. Today, only if a model is extremely simple would we run through a few iterations of updating the statements in the model by hand.

What are other human activities that qualify as the creation of possible worlds? The one that leaps to mind is writing fictitious stories. Let's compare the properties of stories against the postulated properties of simulation (shown in bold). The **analog** for expressing stories is human language, which is still beyond computers' ability to fully interpret. Stories are a sequence of **possible worIds**. The possible worlds are very sharply **bounded**. One reason why they must be is that the author plans to transmit the stories to an audience through some media. The number of primary characters that are tracked in a story is small although more characters and objects pop in and the back out of the story. They are not tracked after they leave. Stories do not appears to be **Markovian,** beyond the fact that they can be held in finite human and computer memories. The **abstraction level** is usually at the level of direct human sensing. There are exceptions that explore the large and the small. Stories are expressed as linear sequences of statements, each one being a difference between the last possible world and the next, these statements have to be **integrated** together to produce full possible worlds. Stories follow what happens to characters over time.

Stories differ from contemporary simulations in a number of ways. They are expressed as differences from a default world. Only interesting difference states are expressed, the rest remain in a don't care state until it becomes necessary to instantiate them. Introspection does not reveal the use of Markovian states in stories. Furthermore, time in stories can jump around rapidly so that sometimes there is not a clear statement of now or even necessarily an ordering between every event. Where order is not essential to understanding the story, it may be dropped. Another difference between stories and most current simulations is that stories are very flexible in their scope and resolution. They can change their scope at any time, adding and dropping new elements and changing resolution as necessary. Stories need to take advantage of commonly understood expectations about the behavior and organization of the world because so much of the context has to be filled in by the audience. This would not seem to be possible without evoking preexisting context patterns. Perhaps one of the biggest differences between today's simulations and stories is that the story writer knows what the focus of the story is when he creates it. Simulations make no such commitment. The story writer can come up with a story line for the main characters and then work out the events around the story line to support it. General simulations treat each event with equal importance. If we could change that aspect of simulation, it would be very useful for training exercises where the trainees need to be presented with certain complex events to generate the training. General simulations can execute all day without presenting an appropriate training situation. Exercise controllers are required to drive the simulation in the appropriate direction.

Stories are delivered in many different forms: storytelling, books, plays, movies, television, music, games, and now computer games. For most of these formats, additional simulation needs to be done to add more detail to the existing story. Illustrations can be added to books to show what the possible worlds look like. Again the issue of simulation in time versus space is raised. Movies and television change the story from a verbal description to visible scenes that dramatize the interactions. Game designers need to transform the story into a digital analog. These are all examples

of hierarchical simulation, where a skeleton set of possible worlds is augmented by more details and adapted for different presentation media.

Plays, movies, and television transfer stories from the realm of pure language into the realm of human simulation complete with rehearsals. People act out and interpret stories and here we find a different form of natural simulation, acting. In the past, simulation tended to have negative connotations. It was associated with false expressions of behavior designed to deceive people. People can use their outward behavior to simulate emotional displays that are not really in effect and they can tell lies to induce belief in counterfactual possible worlds in others, typically so as to benefit themselves. This is a much more complicated form of simulation than story writing. The person working to induce a counterfactual possible world in others has to not only create a story but one that is not contradicted by the knowledge other people have. To do it well for any period of time, the perpetrator has to not only know the current state and history of the world (assuming the absence of Markov independence), but also know the current state and history of the counterfactual world and ensure that the two worlds appear identical to the target.

Being aware of other's intentions and feelings is called having a theory of mind. Simulation is also the name of a psychological theory that holds that we understand other's intentions and feelings by simulating how different intensions and goals could bring about their actions and emotional displays (Marraffa). In the 80's and 90's, researchers at the University of Parma discovered and studied neurons in monkeys that would fire both when they performed a certain action and when they observed another animal perform that action (Winerman, 2005). The presence of the same mirror neurons was also found in humans and tests show that some mirror neurons activate when we feel pain and when we watch other people feel pain (Morrison et al, 2004). If true, this would indicate that we are capable of simulating the feelings and emotions of other people by watching them and perhaps explain empathy and why we are so captivated with movies and stories. They stimulate our mirror neuron system and cause us to feel the emotions of the characters. This brings up the question of how much of our ability to perform mental simulation is based on general computing elements in our conscious minds versus special purpose hardware in the unconscious. Since these neurons were first found in monkeys, it suggests that monkeys also perform this type of simulation.

The most primitive area, which I have heard about, where simulation may be occurring, is in the control of movement. Tactile sensory input is very noisy and yet we are able to very precisely control our movements. Researchers (Wolpert, 2010) have found evidence that when we send out movement commands to our muscles we also generate estimates of what the sensory feedback caused by that movement is going to be, as shown in Figure 6. The difference between the expected feedback and the actual feedback is used to detect how well our limbs are executing their movement commands and allow us to assert finer control than would otherwise be possible. I suggest that the generation of that estimated sensory feedback is an example of subconscious simulation and one that is very likely to be present in many animals in addition to monkeys.

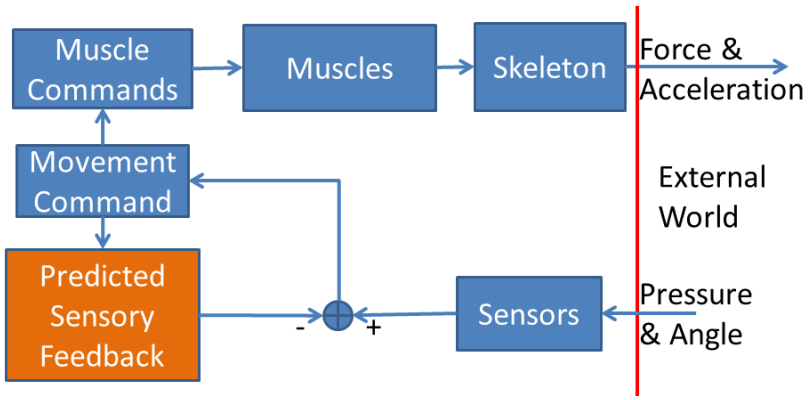


Figure 6: Simulation at the Muscle Control Level

Hawkins postulates that the function of the neocortex is to produce exactly the sensor feedback that has been found to accompany muscle movements (Hawkins, 2004). According to this theory, input from the senses travels up through the neocortex on upward pathways. For every upward pathway, there is a parallel downward pathway which

carries sensory predictions. When they diverge, alarms go off. We don't have to go through a mental inventory checklist to detect changes, it just happens as part of the differencing of sensory expectations and inputs. The neocortex creates possible sensory worlds predicting new sensor inputs. This type of approach may one day be applied to C4I and surveillance systems to help turn data into information.

We shall end our brief tour of natural simulation with mental simulation. Hesslow has proposed that internal simulation is responsible for the inner world we refer to as consciousness (Hesslow, 1994, Hesslow & Jirenhed, 2007). He has even created simulators of robots with internal consciousness producing simulators. He further proposes that we can use our internal representations to cause our perceptual analysis and motor control machinery to be activated just as if we were receiving external stimuli and planning to move but then suppress our actual motor response. Iterating this process produces a mental simulation.

Hawkins' theory of how the neocortex works allows us to imagine how we might run mental simulations as Hesslow proposes. Consider reading a book. As we read, images of words representing abstract verbal concepts are input into our sensory processing apparatus. These symbols activate high level concepts that have been learned by the upper layers of the neocortex. Since the neocortex is a distributed auto-associative memory, the presentation of the word 'dog' not only causes the corresponding symbol recognition part of the neocortex to activate but also causes the activation of the concept of 'dog'. The downward channels of the neocortex start to fire and we get some elaboration of 'dog', although the lack of an upward signal may limit how far down the elaboration can go. Our skill at reading and visualization may determine how much visual activation we get. The use of picture books for young readers allows the corresponding upward and downward visual flows to be available in the reader's mind for association. It seems logical that experienced readers would need less elaboration. Movement symbols cause our motor control concepts to fire adding more predictions of sensory stimuli to get generated. Repeated activation of high level action symbols steps us through the story setting off cascades of elaborated sensory and emotional activations. When we watch a movie, the visual and auditory upward flows are satisfied, leaving the simulation of the emotional responses.

We can turn all this machinery to the task of imagining our own future actions and conduct planning by sequentially activating different higher level concepts in the neocortex without sensory input. Goal seeking brain functions select an action concept to activate causing downward sensory elaborations to show us what the world would look like if we executed that action. The auto associative neocortex activates not only sensory concepts but also new movement or action concepts that are appropriate in that situation. The goal seeking functions repeatedly choose action concepts showing us a sequence of possible worlds into the future. This is very speculative, but it agrees with our experience of being able to imagine future actions in real time seemingly effortlessly without the painful application of individual constraint rules from the world model and without detailed specification of the full initial conditions. It is a very different process from pen and paper simulation. The constraint rules of the world model are built up from all our previous auto-associative processing of sensory data including symbolic data. Conscious processes decide which chain of activations to follow. Thus, we are able to use internal simulation to explore our own futures.

SUMMARY

Having spent a lifetime working in the simulation field, I set out to synthesize my experience into a generic characterization of simulation. I wanted to produce an accessible synthesis that addressed fundamental issues. In the process, I discovered that the net cast by the name simulation covers a wider and more heterogeneous set of species than I initially imagined. I also found philosophical concepts that could be applied to the problem. The result is the unifying concept that simulation can be understood as the process of incrementally constructing possible worlds by sampling within the bounds of spatial, relational, and temporal consistency rules. Simulators sample possible worlds. Simulation analysis is inference over these samples. It is worth noting that many simulators are not used to perform simulation analyses. Some will find this attempt to find common patterns in the many activities performed under the name of simulation as too generic and admitting too much; a difference that makes no difference. However, if this interpretation allows us to see differences between application areas as alternative implementations of common functions, we may be able to adapt those implementations to different domains.

Possible worlds can be represented in analogs which require physical resources to implement. They are by necessity subset worlds, restricted in system scope, allowable external conditions and configurations. The seemingly infinite

detail of the actual world must be reduced to the finite in the modeling process through isolation, establishment of finite memory of the past, and the selection of a basis of descriptive objects. Analogs consist of two parts, statements about the configuration of the possible world, i.e., state variables, and statements about rules that govern acceptable changes and configurations in the state variables, i.e., the world model. A simulation model is an ontology of possible configuration statements plus the world model. Analogs may be symbolic, physical, human, or hybrid. Additional physical resources which can induce updates in analogs are added to create simulators which can produce different possible worlds in the analog.

Simulators have common features. They update analogs by performing inferences and probabilistic sampling that is consistent with the current state, inputs, and the world model. Updates are conducted in steps small enough that the system is approximately decoupled during the update. They produce concrete values devoid of any considerations of uncertainty. Updates are additive and are integrated to produce new possible worlds. Whether simulation is performed at a single instance in time, as in the creation of a painting, or along a timeline, possible worlds are created with similar techniques under an appropriate constraint model. Simulation is natural; it is performed by natural systems, humans and other animals, both at the conscious and unconscious level and consciousness itself may be a form of simulation.

IMPLICATIONS FOR COMPOSITION AND INTEROPERABILITY

The necessity of radically reducing the actual world to a manageable subset and engineering it for efficient updating, before it can be realized as a simulator, has strong implications for a posteriori composition of simulators. A posteriori composition is inspired by the real world composition of forces. Independently built actual platforms are able to operate together without coordination by their manufacturers. Consider the interoperability of the platforms of opposing countries. They are built in a high degree of secrecy and yet despite all efforts to the contrary, they are capable of detecting and destroying each other. This level of interoperability is an artifact of operating in common environments with the same temporal, relational, and spatial consistency laws. In areas, such as information systems, where universal physical laws are absent, interoperability problems are common. Because simulators are radical subsets of the real world, their designs are the result of very large numbers of design decisions for which the chances of independent duplication are vanishingly small. So there is a high probability that rework will be required to achieve full simulator interoperability. Often problems are managed via scenario restrictions instead. Furthermore, some rework is going to be required for most new applications because different applications admit different reductions. Making simulators general, defeats reduction and causes an explosion in complexity and resource requirements. Achievement of full generality can be harder than building real world systems.

Fair play is a particular problem in composition of simulators caused by overlaps in functional representations. In a single simulator it is normal to have a single representation of terrain, a single representation of each movement modality, a single representation of detection per sensor modality, and so on. When we build federations by a posteriori composition, we duplicate all these representations because partitioning by objects is necessary for simulators to be able to operate standalone. Object partitioning also seems to minimize communication requirements. The only way to eliminate fair play problems is to eliminate functional representation overlaps. The representation may still vary in resolution by functional area but at least it is equal for all agents. This can be very difficult to do, for example, when a human trainee is matched against a computer opponent. However, human against human, driving the same synthetic vehicles over the same synthetic terrain and using the same synthetic weapons does provide a fair fight. Standards can help to limit fair fight problems. However, to eliminate all problems, they would have to address every possible reduction decision a simulator developer would make, which is impossible. Each new simulator adds more functionality which is not covered by the existing standards. The use of common data and algorithms will not solve the problem if they are subsequently customized to work in the context of a particular simulator, because then they are no longer common. While these mitigations can help, they are just not guaranteed to solve all problems. Clearly, we have many examples of a posteriori federations; we have been building them for over 20 years. However, they continue to contain problems that are exposed when they execute new scenarios.

There are two approaches to solving the problems of a posteriori composition interoperability. First, there is the construction of common environment models for simulators to be built on top of; essentially a common framework of services that provide all the common rules and physical constraints provided by nature. In addition, we need to

add man-made common constraints such as those impacting the operation of Link 16. Otherwise, you can get the same sort of interoperability problem, where two Link 16 simulators won't work with each other because of different reductions. This means that independently constructed models need to be tightly coupled into a particular framework. However, these frameworks are subject to the same reduction requirements as individual simulators and cannot be general. The second approach is to perform composition a priori instead of a posteriori. The Discrete Event System Specification (DEVS) (Ziegler, 2000) is an approach for separating models from simulators. Yet to fully realize a priori composition, we need tools that can help with the complexity of model merging and the application of appropriate reductions to enable executable realizations of the merged models. Even when such tools become available, it is likely to remain at least a semi-automated activity because some models may need to have information added to them to enable merging. Efficient a priori composition requires solving the problem of semantic mapping between models.

The a posteriori composition approach has been much easier to implement thus far. It capitalizes on the large number of existing simulators in use in DoD. Many differences between simulators are not significant in the context of particular classes of scenarios and do not generate anomalies. However, in the long run, the a priori composition of models and compilation into simulators will be the preferred approach. It will enable the efficient generation of simulations for different applications and architectures without fair fight problems. It is sometimes remarked that DoD should only have one simulator for each platform or weapon. I hope that this discussion convinces you that a single model that can be compiled into multiple simulators would be a better idea.

CONNECTIONS

One of my goals for writing this paper was to find connections between elements of simulation that I was not previously aware of. In this regard, I feel I have been successful, although it remains to be seen if any of the connections I have made will actually lead to useful results. I have found that resources in philosophy can be useful in understanding simulation, including possible worlds semantics, modal logic, and probability logic. The realization that state variables are simply abbreviated statements from logic showed me that the state of a simulator can be laid out as a graph. Furthermore, the possible worlds viewpoint lays the dynamic behavior of a simulator out as a graph. A simulation can be viewed as a Markov chain, albeit usually one with an untenably large set of nodes. Possible worlds also link simulation to information theory. The relationship between simulation and graphs is well known in discrete event simulation (Schruben, 1983) and the simulation theory of (Rasmussen & Barrett, 1995) is based on graphs. I have also come to appreciate how inputs, state variables, and objects work together to limit the complexity of a model and the implications of that reduction on composition.

The concept of simulation as the creation of possible worlds seems to have potential to unify our concepts of simulation. For example, scientists simulate to create possible worlds to test against objective experience and artists simulate to create possible worlds to test against human subjective experience. I wonder how many of the features of stories might be applicable to digital simulations. Could they generate cheaper and more effective training? If simulation helped us to mechanize our natural ability to detect change and recognize situations in the world, it would be a huge step for addressing the crush of data hitting decision makers. However, this analysis is clearly unfinished and the development of a possible worlds semantics underpinning for simulation is still just a suggestion. Even the concept of what it means to create a possible world needs to be placed on a sounder footing.

REFERENCES

Accomazzi, V. (2013). *Voxel Based Graphics on Intel Architectures,* Retrieved August 8, 2014 from https://software.intel.com/en-us/articles/voxel-based-graphics-on-intel-architectures.

Bartha, P. (2013). Analogy and Analogical Reasoning, *The Stanford Encyclopedia of Philosophy (Fall 2013 Edition), Edward N. Zalta (ed.)*, URL = <http://plato.stanford.edu/archives/fall2013/entries/reasoning-analogy/>.

Bekey, G., & Karplus, W. (1968). *Hybrid Computation*, New York, John Wiley and Sons, Inc.

Bostrom, N. (2003). Are You Living In a Computer Simulation, *Philosophical Quarterly, Vol. 53, No. 211*, pp. 243-255.

Braun, W., & Murdock, D. (2007). *A First Course in Statistical Programming with R*, Cambridge, Cambridge University Press.

Budge, L., Strini, R., Dehncke, R., & Hunt, J. (1997). Synthetic Theater of War (STOW) 97 Overview, *Spring 1998 Simulation Interoperability Workshop*, 98S-SIW-086.

Burnett, S., & Gottschalk, T. (1998). A Large-scale Metacomputing Framework for ModSAF Realtime Simulation, *Parallel Computing 24*.

Calvin, J., Cebula, D., Chiang, C., Rak, S., & Van Hook, D. (1995). Data Subscription in Support of Multicast Group Allocation, *13th Workshop on Standards for the Interoperability of Distributed Simulations*, 95-13-093.

Ceranowicz, A. (1994). ModSAF Capabilities, *Proceedings of the Fourth Conference of Computer Generated Forces and Behavioral Representation*, Orlando FL, Institute for Simulation and Training, 3-8.

Ceranowicz, A., & Torpey, M. (2005). Adapting to Urban Warfare, *The Journal of Defense Modeling and Simulation 2 (1)*, 3-15.

Chan, S., Chan, S., & Chan, S. (1972). *Analysis of Linear Networks and Systems*, Reading MA, Addison-Wesley Publishing Company.

Colwell, B. (2013), *EDA at the End of Moore's Law*, Retrieved August 8, 2014 from http://www.cra.org/ccc/files/docs/esda/esda_keynote1.pdf

Computer History Museum (2008). *The Babbage Engine*, Retrieved August 11, 2014 from http://www.computerhistory.org/babbage/engines/

Data Distribution Service Portal, Retrieved on June 7, 2014, http://portals.omg.org/dds/documents,

Davis, P., & Huber, R. (1992). *Variable-Resolution Combat Modeling: Motivations, Issues, and Principles*, Santa Monica CA, RAND.

Davis, P., & Anderson, R. (2003). *Improving the Composability of Department of Defense Models and Simulations*, Santa Monica CA, RAND.

Demey, L., Kooi, B., & Sack, J. (2013) "Logic and Probability", *The Stanford Encyclopedia of Philosophy (Spring 2013 Edition), Edward N. Zalta (ed.)*, URL = <http://plato.stanford.edu/archives/spr2013/entries/logic-probability/>

Dennard, R., Gaensslen, F., Yu, H., Rideout, V., Basous, E., & Leblanc, A. (1999 Reprint). Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions, *Proceedings of the IEEE, 87 (4)*, 668-687.

Dennett, D. (1991). *Consciousness Explained*, Boston, Little Brown and Company.

Diallo, S., Padilla, J., & Tolk, A. (2010). Why is Interoperability Bad; Towards a Paradigm Shift in Simulation Composition, *Fall 2010 Simulation Interoperability Workshop*.

Esmaeilzadehy, H., Blemz, E., Amantx, R., Sankaralingamz, K., & Burger, D. (2011). Dark Silicon and the End of Multicore Scaling, *Proceedings of the 38th International Symposium of Computer Architecture*.

Erleben, K., Sporring, J., Kenrikesn, K., & Dohlmann, H. (2005). *Physics Based Animation*, Newton MA, CharlesRiver Media.

Eugster, P., Felber, P., Guerraoui, R., & Kermarrec, A. (2003). The Many Faces of Publish/Subscribe, ACM *Computing Surveys 35 (2)*, 114-131.

Frigg, R., & Reiss, J. (2009). The Philosophy of Simulation: Hot New Issues or Same Old Stew, *Synthese 169*, 593-613.

Fugimoto, R. (2000). *Parallel and Distributed Simulation Systems*, Toronto, John Wiley and Sons, Inc.

Garson, J. (2014). Modal Logic, *The Stanford Encyclopedia of Philosophy (Summer 2014 Edition)*, Edward N. Zalta (ed.), URL = <http://plato.stanford.edu/archives/sum2014/entries/logic-modal/>

Goldsman, D., Nance, R., & Wilson, J. (2010). A Brief History of Simulation Revisited, *Proceedings of the 2010 Winter Simulation Conference*.

Gorman, P. (2011). Learning to Learn: Reminiscences and Anticipation, *Proceedings, Interservice / Industry Training, Simulation, and Education Conference (I/ITSEC)*, December 2011.

Hartmann, S. (2005). *The World as a Process: Simulations in the Natural and Social Sciences [Preprint]*, Retrieved August 10, 2014 from http://philsci-archive.pitt.edu/cgi/export/2412/Text_Chicago/philsci-archive-2412.txt

Hawkins, J., & Blakeslee, S. (2004). *On Intelligence*, New York, Owl Books.

Helfinstine, B., Torpey, M. & Wagenbreth, G. (2003). Experimental Interest Management Architecture for DCEE, *Proceedings of the 2003 Interservice / Industry Training Simulation and Education Conference*.

Hesslow, G. (1994). Will neuroscience explain consciousness? *Journal of Theoretical Biology*. 171:29-39.

Hesslow, G., & Jirenhed, D. (2007). Must Machines be Zombies? Internal Simulation as a Mechanism for Machine Consciousness, *AAAI Symposium*, Washington DC, 8-11.

Imam, N., Cleland, T., Manohar, R., Merolla, P., Arthur, J., Akopyan, F., and Modha, D. (2012). Implementation of olfactory bulb glomerular-layer computations in a digital neurosynaptic core, *Frontiers in Neuroscience 6 (83).*

Johnson, K. (2014). *Exploring Metaphysics Course Guide Book,* Chantilly VA, The Great Courses.

Kamermans, M., & Schmits, T. (2004). *The History of the Frame Problem*, Retrieved August 21, 2014 from http://staff.science.uva.nl/~bredeweg/pdf/BSc/20032004/KamermansSchmits.pdf

Kruschke, J. (2011). *Doing Bayesian Data Analysis*, Burlington MA, Academic Press.

Kuhl, F., Weatherly, R., & Dahmann, J. (2000). *Creating Computer Simulation Systems,* Upper Saddle River, NJ, Prentice Hall.

Kuipers, B. (1994). *Qualitative Reasoning,* Cambridge MA, The MIT Press.

Lamport, L. (1978). Time, Clocks, and Ordering of Events in a Distributed System, *Communications of the ACM 21(7)*, 558-565.

Lucas, R., & Davis, D. (2003). Joint Experimentation on Scalable Parallel Processors, *Proceedings of the 2003 Interservice / Industry Training Simulation and Education Conference*.

McGarry, S., & Torpey, M. (1999). Back to Basics: Balancing Computation and Bandwidth, *Fall 1999 Simulation Interoperability Workshop.*

Marraffa, M., Theory of Mind, *Internet Encyclopedia of Philosophy,* Retrieved August 13, 2014 from http://www.iep.utm.edu/theomind/print

Metropolis, N., & Ulam, S. (1949). The Monte Carlo Method, *The Journal of the American Statistical Association, 247 (44)*, 335-341.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., & Teller, E. (1953). Equation of State Calculations by Fast Computing machines, *The Journal of Chemical Physics, 21(6),* 1087-1092.

Miller, J., & Page, S. (2007). *Complex Adaptive Systems*, Princeton NJ, Princeton University Press.

Moon, I. (2013). Theoretic Interplay Between Abstraction, Resolution and Fidelity in Model Information, *2013 Winter Simulation Conference*, 1283-1291.

Moore, G. (1965). Cramming More Components onto Integrated Circuits, *Electronics, April,* 114-117.

Morrison, I., Lloyd, D., Pellegrino, G., & Roberts, N. (2004). Vicarious responses to pain in anterior cingulate cortex: Is empathy a multisensory issue?, *Cognitive, Affective, & Behavioral Neuroscience, 2004, 4 (2),* 270-278.

Nygaard, K., & Dahl, O. (1981). The development of the SIMULA Languages, *History of Programming Languages*, New York: Academic Press.

Nardi, D., & Brachman, R. (2007). An Introduction to Descriptive Logics, *The Description Logic Handbook (Second Edition)*, Cambridge, Cambridge University Press.

Neyland, D. (1997). *Virtual Combat: A Guide To Distributed Interactive Simulation*, Mechanicsville PA, Stackpole Books.

Noseworthy, J. (2011). Providing Interoperable Real-Time Data Communication with TENA, *MILCOM*, Retrieved August 17, 2014 from https://www.tena-sda.org/display/intro/Documentation

Olson, E. (2010). Personal Identity, *The Stanford Encyclopedia of Philosophy* (Winter 2010 Edition), Edward N. Zalta (ed.), URL = <http://plato.stanford.edu/archives/win2010/entries/identity-personal/>

Orlansky, J., & Thorpe, J. (1992). *73 Easting: Lessons Learned from Desert Storm via Advanced Distributed Simulation Technology, IDA Document D-1110,* Alexandria VA, Institute For Defense Analyses.

Owens, L. (1986).Vannevar Bush and the Differential Analyzer: The Text and Context of an Early Computer, *Technology and Culture, 27(1),* 63-95.

Petty, M., & Weisel, E. (2003). A Composability Lexicon, *Spring 2003 Simulation Interoperability Workshop*, 03S-SIW-023.

Rand, G. (2011). Berwyn Hugh Patrick Rivett, *Profiles in Operations Research: Pioneers and Innovators (Chaper 26),* New York, Springer.

Rasmussen, S., & Barret, C. (1995). *Elements of a Theory of Simulation*, Retrieved August 10, 2014 from http://www.santafe.edu/research/working-papers/abstract/75d1bca2a300c2c3d9da29789fcea463/.

Ross, C. (2012). *A Video on the Finite Element Method*, Retrieved August 10, 2014 from https://www.youtube.com/watch?v=lrpj3cZrKn4

Shanahan, M. (2009). The Frame Problem, *The Stanford Encyclopedia of Philosophy (Winter 2009 Edition), Edward N. Zalta (ed.),* URL = <http://plato.stanford.edu/archives/win2009/entries/frame-problem/>.

Schruben, L. (1983). Simulation Modeling with Event Graphs, *Communications of the ACM, 26 (11),* 957-963.

Shepard, R., & Cooper, L. (1982). *Mental images and their transformations*. Cambridge, MA: MIT Press.

Shiflett, J. (2013). Observations on the Development and Implementation of Distributed Simulation, *Interservice / Industry Training, Simulation, and Education Conference (I/ITSEC),* December 2013.

Smith, R. (2014). Aristotle's Logic, *The Stanford Encyclopedia of Philosophy (Spring 2014 Edition),* Edward N. Zalta (ed.), URL = <http://plato.stanford.edu/archives/spr2014/entries/aristotle-logic/>

Thorpe, J. (2010). Trends in Modeling, Simulation, & Gaming: Personal Observations About The Past 30 Years & Speculation About The Next 10, *Proceedings, Interservice / Industry Training, Simulation, and Education Conference (I/ITSEC).*

Tolk, A., & Muruira, J. (2003). The Levels of Conceptual Interoperability Model, *2003 Fall Interoperability Workshop*, 03F-SIW-007.

Van Inwagen, P. (2013). Metaphysics, *The Stanford Encyclopedia of Philosophy (Winter 2013 Edition),* Edward N. Zalta (ed.), URL = <http://plato.stanford.edu/archives/win2013/entries/metaphysics/>

Wolpert, P. (2010). *The real reason for brains*, Retrieved 4/10/2014, from http://www.ted.com/talks/daniel_wolpert_the_real_reason_for_brains#t-146033, July 2011,

Wolfram, S. (2002). *A New Kind of Science*, Champaign, Wolfram Media Inc.

Welk, M. (1961). *The ENIAC Story*, Retrieved August 4, 2014, from http://ftp.arl.mil/mike/comphist/eniac-story.html

Williams, R., & Smith, S. (2007). UR2015: Technical Integration Lessons Learned, *Proceedings, Interservice / Industry Training, Simulation, and Education Conference (I/ITSEC).*

Winerman, L. (2005). *The Mind's Mirror*, Retrieved August 14, 2014 from http://www.apa.org/monitor/oct05/mirror.aspx

Zeigler, B., Praehofer, H., & Kim, T. (2000). *Theory of Modeling and Simulation, Second Edition*, Amsterdam.

Zeigler, B., Mittal, S., & Hu, X. (2012). *Toward a Formal Standard for Interoperability in M&S/Systems of Systems Integration*, Retrived August 13 from http://acims.asu.edu/wp-content/uploads/2012/02/Towards-a-Formal-Standard-for-Interoperability-in-MSSystem-of-Systems-Integration.pdf