A Simulation Engine to Predict Multi-Agent Work in Complex, Dynamic, Heterogeneous Systems

Amy R. Pritchett
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0150
Email: amy.pritchett@ae.gatech.edu

H. Claus Christmann
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0150
Email: hcc@gatech.edu

Matthew S. Bigelow
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0150
Email: mattbig@gatech.edu

***Abstract*—This paper documents a simulation engine developed to accurately and efficiently simulate work by multiple agents in complex dynamic systems. Agents (human or mechanical) are modeled as responding to, and changing, their environment by executing the actions that get and set the value of resources in the environment. Each action comprises the processes that need to be evaluated at the same time by the same agent, which are used to reference (get) resources, consider them according to simple or complicated processes, and then interact back on the environment by setting resources appropriately. This paper specifically addresses timing within the simulation. The simplest approach would update all actions at the smallest unit of conceivable time, an approach that is not only computationally inefficient, but also not an accurate representation of situated behavior. Instead, every action declares its next update time as required to accurately model its internal dynamics and the simulation engine executes them asynchronously. Thus, an action and the resources it 'gets' from the environment are not inherently contemporary; instead, each action also specifies, for each resource value that it gets, the quality of service required in terms of its temporal currency. This reflects dynamics of the real processes being simulated: when, in actual operations, would the environment be sampled, and how accurately must its state be known? Additionally, this also reflects dynamics of environmental resources how often (or how fast) does each inherently change? Using these constructs, the list of actions to be simulated are sorted by the simulation engine according to their next update time. Each action, when its time comes, is given to their agent model to be executed, and then is sorted back into the action list according to its self-reported next update time. Thus, actions are each updated when they need to be. In situations where, for example, action Y needs to get a resource which, because action X has not set it recently, does not meet action Ys required Quality of Service. The simulation engine will invoke action X immediately before action Y, mimicking cases in the real system where one process calls on another to establish the conditions it needs. The presented simulation engine is a complete redevelopment, designed and written from scratch at the Cognitive Engineering Center at the Georgia Institute of Technology.**

I. Introduction

This paper describes simulating work in complex, heterogeneous dynamic systems that include humans, physical systems, computer agents and regulatory requirements. We define '*work*' as '*purposeful activity acting on, and responding to, the environment.*' The purpose of this research is to form computational models suitable for the description and prediction of real, complex work domains in which situated cognition is fundamental to performance. Thus, these models are not directed at any specific theory of situated cognition, but instead are driven by their engineering utility in supporting analysis and design of complex, multi-agent systems.

Multi-agent simulation has been previously used for such analysis [1], [2], [3]. However, these simulations modeled the behavior of components. For example, air traffic simulations created a model of an air traffic controller, of aircraft, and of pilots; each component model described its behavior separately such that components did not directly act on a shared environment using shared constructs. In contrast, this paper describes a simulation engine that enables detailed models of the collective work (and interaction) of multiple agents towards clear work goals. This work has inherent patterns and structures as established by the physical environment and by a procedural environment defined by established work practices, procedures and regulatory requirements. Any number of agents may collectively perform the work; each may be human or automated. In these models, cognition is assumed to be embodied such that agent knowledge is represented as the ability to perform the actions achieving the work goals, without necessarily distinguishing between physical and cognitive activities. At its most atomic, these simulations represent work as actions that evaluate the current situation, and then change environmental variables as appropriate. For example, an air traffic controller scans a radar display (i.e., gets values from the environment), and then calls for changes to aircraft headings to avoid conflicts (i.e. sets values in the environment).

This paper specifically focuses on the insights provided by simulating such models - and on the temporal constructs required to adequately capture the timing and interplay of actions situated in a model of the environment. The Cognitive Engineering Center[1] at Georgia Tech developed the simulation engine described in this paper to address two requirements. The first is computational time - it is easy to construct a simulation with a run-time too long to estimate meaningful statistics about real systems. The second requirement is that the behaviors are not only represented correctly in a static sense, but also that their temporal semantics are defined and

[1]http://www.cognitiveengineering.gatech.edu/

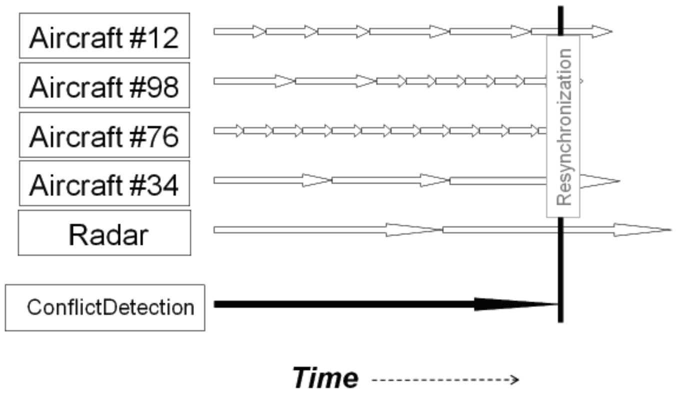

Fig. 1. Schematic of an asynchronous simulation with resynchronization. Each arrow represents the incremental next update time reported by each model.

applied correctly: any model should update only at that time where it would in the real system, as well as ensuring that each model can have a contemporary view of environmental resources as required.

This paper starts by reviewing prior simulations of complex, dynamic systems such as air traffic control, noting how they tended to focus on component models, rather than situated behavior captured by a work model describing fluid interaction between action and environment within a team of agents. The work models and agent models are briefly reviewed to illustrate how they provide the (static) representation of situated cognition inherent in work models and agent models; for fuller descriptions, see [4], [5]. Then, this paper describes a simulation engine that enables asynchronous timing of actions within the work model. Specific considerations are outlined that ensure that actions act upon the environment at the times needed for other actions to have sufficiently-contemporary values. Finally, further timing constructs describing time in models of resources and and agents are described.

II. Background: Simulation of Complex, Dynamic, Heterogeneous Systems

Hybrid-system simulations have been described suitable to examine systems comprising heterogeneous behaviors, i.e., behaviors best described as including both continuous-time and discrete-event dynamics. Previous studies have highlighted the range of behaviors that a hybrid-system simulation must accurately model; for example, see [3] reviewing hybrid-system simulation applied to examine air traffic control.

A simulation engine should capitalize upon those capabilities shared by these disparate model forms: to update themselves when required; to report when their next update is required; and to report interactions with other objects that warrant a joint update. All other dynamics can remain internal to the models. This can be considered a feature, as it prevents fundamental restrictions on the type of model allowed in the simulation, and as such it allows for the simulation to model those aspects of a complex, dynamic, heterogeneous system at a range of resolutions as required by the task at hand. Further, without placing undue restrictions on model forms, the simulation engine also needs to support interactions between models.

In large-scale simulations integrating multiple models of disparate forms, concerns with computational efficiency extend beyond making each model individually efficient. Overall efficiency is achieved when each model updates only when needed to accurately model its interior dynamics and interact correctly with other models. Any unnecessary updates of models may be considered wasted use of the processor. Thus, the timing method that commands updates of each model is a primary determinant of both computational efficiency and accuracy in model interactions.

Several timing methods can be defined [3]. For example, the '*Synchronous Fixed Time-Step*' timing method updates all models at the same time, with the time step externally fixed through the simulation. This method is commonly used in current flight simulation techniques, where the time step may be fixed by conservative analysis of the fastest dynamics in the system, or by the system clock in real-time simulation. This method provides conservative results that can be guaranteed to not miss any interactions between models. However, it also forces all models to update at a rate governed by a conservative, worst-case estimate of the fastest dynamics in the system, which is computationally inefficient. Similarly, the '*Synchronous Variable Time-Step*' timing method has all the models update at the same time but varies the update time from one time step to the next to meet the needs of the simulation. For instance, the update time may be chosen by polling all models for their desired time step, and then selecting the worst-case (smallest) time step. This method still forces some models to update more often than they would require when running alone, but it can relax the time step when conditions allow.

Of particular interest here, the timing method '*Asynchronous with Resynchronization*' allows for models to be updated independently according to their own update times [3]. This is shown schematically in Fig. 1 for a simulation with four aircraft, a radar unit and a conflict detection algorithm; the aircraft and radar update at their own rates until the conflict detection algorithm requires synchronization. This method allows models with fast dynamics to update frequently without requiring other objects to be bound by such small time steps, yet also resynchronizes the relevant models when interactions require values from temporally co-located models.

The computational benefits of asynchronous timing methods have been described previously in [3], and such simulations have been used to analyze for emergent safety concerns in new air traffic control concepts of operation [1], [2], [3]. However, to date these simulations modeled 'components' each as a separate model, an approach with several conceptual and pragmatic limitations. First, the various component models did not have a shared view of their collective work and did not have shared work environment, except to the extent that they passed information to each other during resynchronizations. Second, they could not fluidly coordinate their responses to the environment, such as collectively adopting new strategies in response to emerging affordances. Third, changes in team design or allocation of functions across the team required re-programming the component models, rather than more-directly re-assigning work models to different agents.

III. Simulation Constructs to Model Work

In contrast to simulations of complex, dynamic, heterogeneous systems comprised of component models, this simulation focuses on the construct of '*work*', modeled here as actions responding to, and acting upon, a shared work environment. The basic constructs used to model and simulate work are detailed in [4], and may be summarized as:

*The **Environment** is the aggregation of resources required to describe the dynamics of the work. This may include both physical and social/cultural/policy constructs. An agent does work by sampling and changing the environment. The environment may have some inherent dynamics within itself; these dynamics are guided by the agents' actions.*

*Each **Resource** represents a tangible state of the environment, such as speed, altitude etc. The collective set of resources represents the current state of the environment. A resource may represent a physical aspect of the environment with continuous dynamics, or may be a discrete value representing a categorization of the state of the environment or a policy decision such as specification of a particular function allocation between agents within the team. In the computational model, a resource also has data type, such as* `double`, `int`, *or* `bool` *or, for resources best described by multiple variables, its data type can be a structure or a vector.*

*Each **Action** is temporally and organizationally atomic in that it represents work performed by one agent at the same time. Actions sample the environment by 'getting' resources and change the environment by 'setting' resources. Actions represent the knowledge of 'work' and are represented in the work model, but are not autonomous and may not execute by themselves - instead, they are passed to agent models or, for dynamics within the environment, a default executor of actions.*

*A special class of actions, **Decision Actions**, respond to the environment by selecting strategies based on contextual factors (environmental, team design and within-agent). Their decisions serve to 'manage the work in response to the situationn' by activating/deactivating actions, assigning actions to agents, and identifying which resources an action gets and sets during its functioning.*

A simple example of two actions and two resources is shown in Fig. 2. The action '*updateSineWaveValue*' models temporal dynamics that update a continuously-valued resource '*sineWaveValue*'; the decision action '*checkSineWaveSign*' models a decision to be implemented when an environmental resource affords it (in this case, when the sine wave changes value), and sets the discrete resource '*SineWaveSign*' accordingly. These actions may be executed by different agents.

Even this simple model illustrates how interactions between actions can arise when they reference a shared set

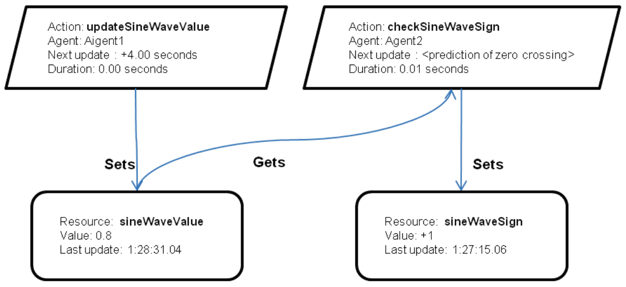

Fig. 2. Two interacting actions and resources.

of environmental resources. Going even further, modeling work in complex, heterogeneous dynamic systems generates a vast number of actions and resources. This complexity is inherent to the work, and requires both modelers and the workers themselves to develop more aggregate abstractions that describe the work according to inherent structures in the work as it relates to the team's goals. Therefore, actions can be aggregated into progressively higher-level '*functions*', building on cognitive engineering models representing work, such as the abstraction hierarchy established by work domain analysis [6], [7].

IV. Simulating a Work Model

Once a work structure is created, the simulation engine translates it into the construct needed for efficient simulation: a list of actions sorted according to their next update time. Fig. 3 illustrates such a translation as a work structure described statically within an abstraction hierarchy is assembled into an action list for dynamic simulation. Those actions which are known to be 'active' at the start are listed at the top of the action list in the order they are referenced when evaluating the work structure. Ultimately, all actions in the work structure are included in the action list, but if not 'active' at the start they are given an unknown update time and, thus, placed at the bottom of the action list.

Thus, once the entire work structure is parsed at the start of a simulation run, the simulation starts advancing its clock. Each action, when its time comes, is given to their agent model to be executed. For example, in the figure the decision action '*Configuration of Control?*' from a higher-level function in the work model sets configuration variables that determine the strategies that will be selected at lower-levels. Then, the decision action '*How to Control Speed?*' assigns several actions to the agent Pilot and schedules the decision action '*Need to Set Autopilot Targets?*'. Finally, once all strategies have been selected by executing decision actions within progressively-lower-level functions, the other actions start to be invoked, such as the '*Update Target Speed*' action setting the resource '*Target Airspeed*' to 200 knots. Once each action has completed its process, it is sorted back into the action list according to its self-reported next update time.

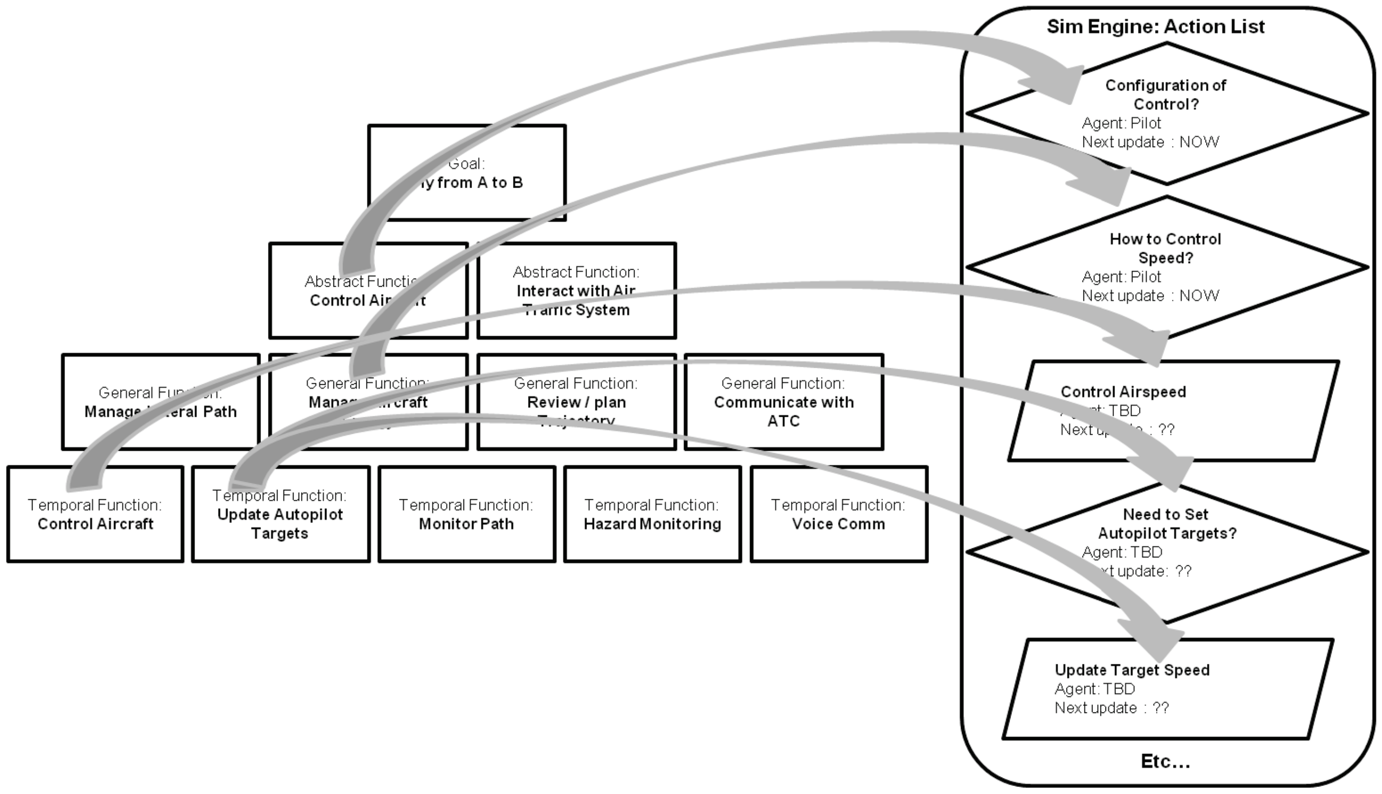

Fig. 3. Translating the work model into the action list used by the simulation engine (figure adapted from [8])

V. Accurately Timing Actions During Simulations of Work

Action models, to work within this simulation engine, need to provide two methods: each action needs to do its 'work' when executed and, of note here, each action needs report to the simulation engine when it next needs to be executed. For models of physical dynamics, the next update time can be determined by numerical methods (such as numerical integration) as a factor of allowable computational error. For models of intermittent dynamics, such as monitoring a varying signal, the next update time is itself a model of monitoring and sampling behaviors, which themselves may be an adaptation to physical dynamics. For example, for a decision action monitoring when an unpredictable signal crosses zero, the next update time may be driven both by the precipitating conditions in the environment and by the prescribed or self-directed scanning intervals of the agent performing the action.

For example, consider the decision action '*checkSineWaveSign*', which 'gets' the resource '*SineWaveValue*' and 'sets' the resource '*SineWaveSign*' to a discrete value: $+1$ if the sine wave is greater than zero and to -1 if the sine wave is less than zero. Fig. 4 illustrates its behavior through the simulation if it can declare a next update time in a smart manner based on a perfect assessment of the environment. In this case, mirroring more-realistic models where the exact time of an event can't be exactly predicted, this decision action makes a worst case estimate of its next update time to be the current value of the sine wave divided by the maximum rate at which it can change. These times are shown in green in Fig. 4; at each of these times the decision is evaluated and the next update time is re-estimated. As the potential time to the event of interest (in this case, a zero crossing) becomes smaller, the decision action is evaluated more often for precision; when the potential time to the event of interest is larger, the decision action is delayed accordingly for computational efficiency.

Calculation of a 'next update time' captures agent execution of predictable or on-going behaviors. Additionally, actions that represent unpredictable events can indicate other actions that they immediately trigger. For example, an '*air traffic controller speed command*' action, which may come at an unpredictable time, may be modeled as triggering the '*set autopilot target speed*' action immediately by the pilot agent. For actions performed by humans, this capability is intended to describe dependencies between actions where (1) the trigger is rare and unpredictable from the point of view of the recipient and (2) the trigger, in the real-world, would generate an interrupt so salient as to disrupt their current activity.

The just-preceding example provided an illustration of how an action may predict its required next update time based on simple knowledge of the dynamics in the environment. However, the example also assumed that the decision action '*checkSineWaveSign*' has perfect, contemporary knowledge of the environmental resource '*sineWaveValue*'. A fuller picture of timing recognizes that this resource is itself set by the

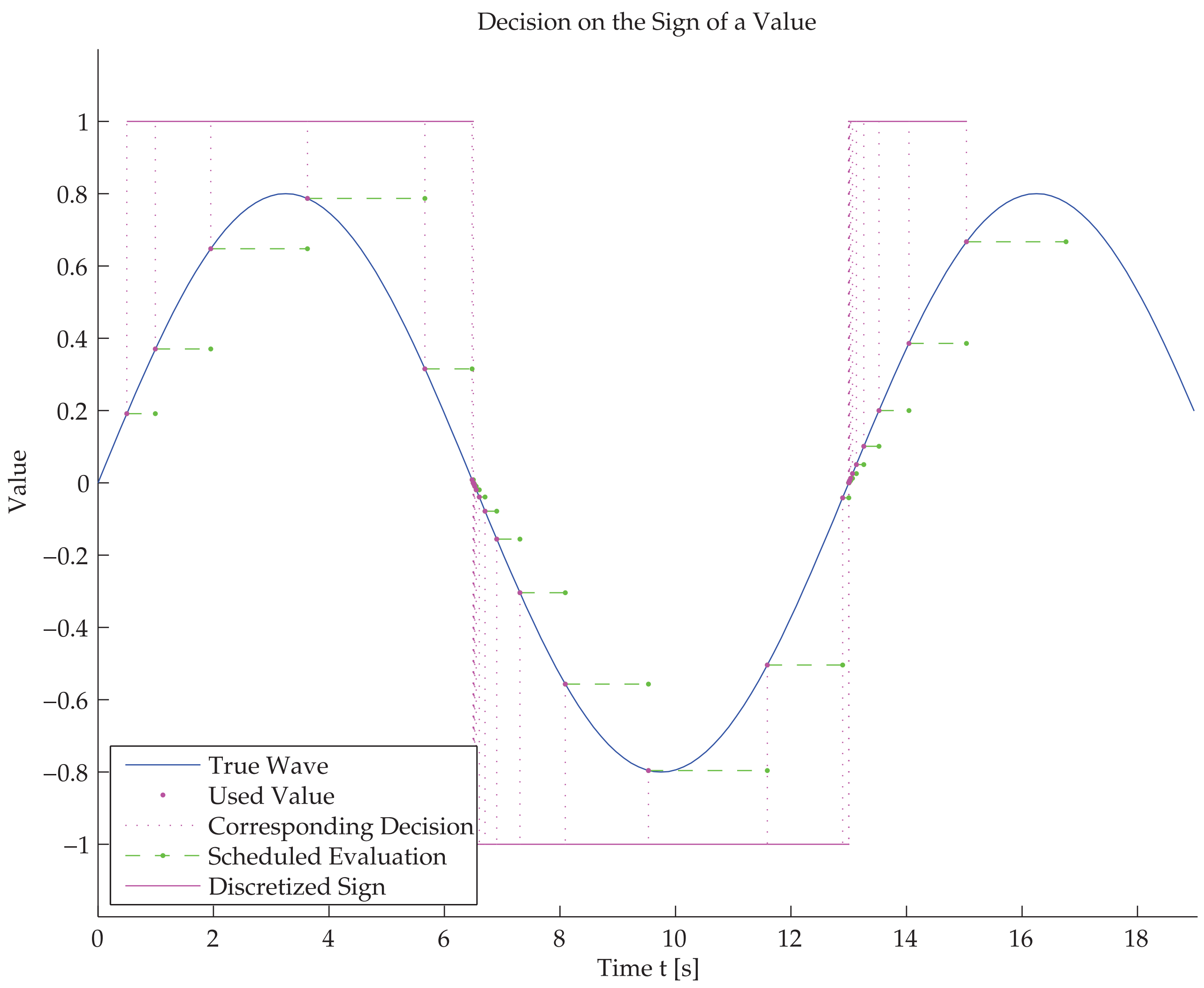

Fig. 4. The action '*checkSineWaveSign*' evaluates the resource '*sineWaveValue*' to evaluate its sign, which is stored as +1 or -1 in the resource '*sineWaveSign*'. By dynamically computing Δt, update times focus on the region where the sinewave crosses zero, leading to increased precision while keeping the number of evaluations low.

action '*updateSineWaveValue*', which also occurs at discrete times. As shown in Fig. 5 if the action '*updateSineWaveValue*' updates only every 4.00 seconds, then the stored value of the resource '*sineWaveValue*' would not provide sufficient resolution for any but the coarsest decisions as to when it has crossed zero.

To ensure that each action has sufficiently-contemporary assessments of the environment, additional constructs are also included in the simulation's models of actions and resources. First, each update to a resource value is time-stamped as to when it was last updated (this is annotated in the resource blocks shown in Fig. 2). Second, each resource 'knows' which action(s) can be called to update its value to the current time when required (this linkage between resource and actions-that-set-it can be dynamically changed through the course of the simulation to reflect new strategies or courses of action in response to context). Third, each action 'knows' which resources it needs to 'get' as its assessment of the environment. With this information, when an action (such as '*checkSineWaveSign*' in this example) attempts to get a resource value that is "too old" ('*sineWaveValue*' in this example), the simulation engine scans the aspects of the environment that the action needs and calls on other actions to update resources accordingly (in this example, if the stored value of '*sineWaveValue*' is too old, the simulation engine will call the action that can set its value to the immediate time, '*updateSineWaveValue*').

This effect is shown in Fig. 6 for the case where the action '*updateSineWaveValue*' declares for its internal purposes a series of next update times (shown in green) that does not provide the currency in the resource '*sineWaveValue*' needed by the other action, '*checkSineWaveSign*'. So, when '*checkSineWaveSign*' updates according to the next update time calculation described earlier, the simulation engine automatically first calls '*updateSineWaveValue*' anytime the resource '*sineWaveValue*' is considered too old. The resulting record of '*sineWaveValue*' is shown in red, and provides the

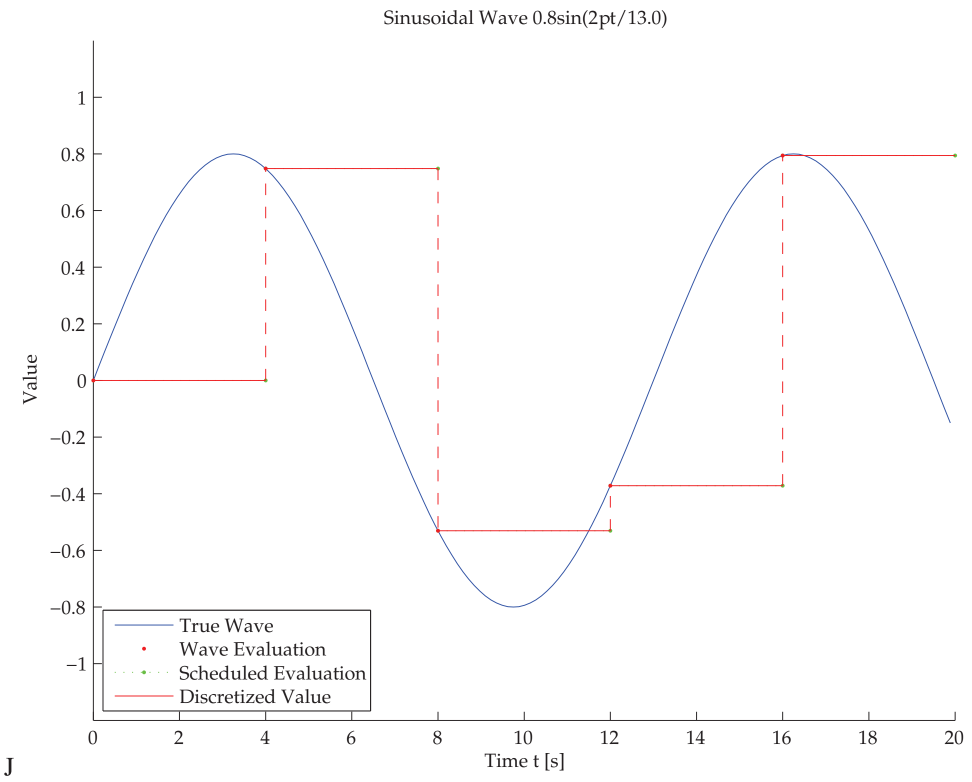

Fig. 5. Comparing the true value of a sine wave to a the value of the resource '*sineWaveValue*' when action '*updateSineWaveValue*' is called every 4.00s.

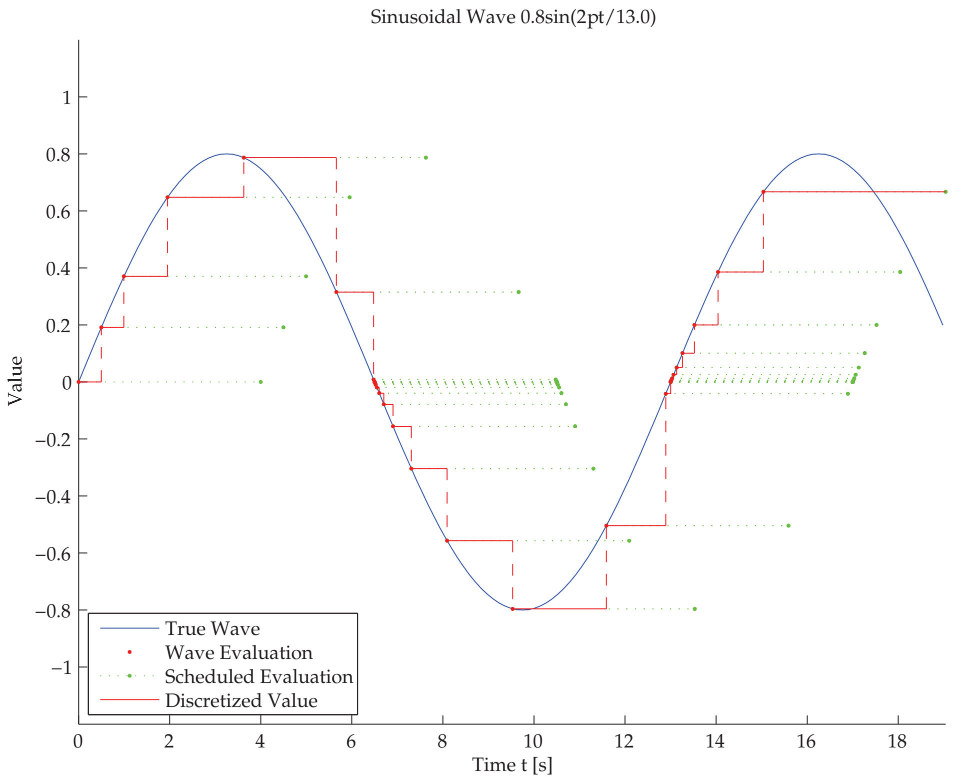

Fig. 6. While the action '*updateSineWaveValue*' schedules itself for evaluation every 4.00s, the action '*checkSineWaveSign*' needs its outcome to be reflected in the resource '*sineWaveValue*' at additional times. The simulation engine recognizes this need and calls for '*updateSineWaveValue*' to execute when needed by other actions.

temporal resolution required by all the actions that sample it.

To further improve computational efficiency, each action can also define a maximum 'age' for each of the resource values that it samples (gets) from the environment. If this maximum age is set to 'zero,' then the actions setting these resources will be invoked. However, it is often more realistic to set this maximum age to model the temporal variance in the environment. For example, a pilot's decisions when controlling an aircraft are based on a range of information sampled via a scan pattern potentially lasting several seconds; thus, it is inaccurate based their decision on an assessment of the environment where every resource value is exactly contemporary. Thus, temporal variance in the simulation can serve as a surrogate for temporal variance in the real work. In addition, where it is conceptually appropriate to specify such maximum 'ages,' profound improvements in computational efficiency can be achieved, as every update to an action that interacts with another (through their 'get' and 'set' relationships to a resource value) will no longer always trigger updates of the other.

In complex work models, calling of 'Action A' may identify an interaction with 'Action B,' which may have an interaction with 'Action C,' and so on. Thus, the simulation engine assembles the list of action-dependencies required to establish immediate values in required resources. These heuristics prevent circular-dependencies within this list:

1) An action may not trigger an interaction with itself, even when it 'gets' and evaluates a resource that it then 'sets'.
2) Any action that does not require other actions to update (due to its aspects of the environment being sufficiently current) is immediately executed, so that it can establish current resource values needed by other actions.
3) Any action that is earlier in the list of action-dependencies cannot identify actions already in the list after its earlier listing.

Consider, for example, the case where the simulation advances to the next update time of Action A and finds that Action A requires Actions B and C to be updated and a temporary list of action-dependencies at that point contains C-B-A. Action B is evaluated next and is found to invoke no dependencies; it is immediately executed, leaving the list of action-dependencies to be C-A. Action C is then evaluated and is found to invoke Action D (list of action-dependencies is enlarged to D-C-A), and finally Action D is evaluated and is found to circularly invoke Action A (the list of action-dependencies becomes A-D-C-A). At that point, the simulation engine recognizes that no further evaluation is required as the dependencies of A either have been resolved (the case of Action B) or will be resolved by the end of the execution of the list. The actions are executed in the order A, D, C and finally A again; the double-execution of 'A' illustrates resolution of a circular dependency.

Using these constructs, each action is completed the instant it is called. An action can be listed as having a duration for the sake of tracing what 'work' was happening in the environment at each point in time. However, the notion of an action having a duration does not change the dynamics of the simulation.

VI. Other Timing Parameters

The previous section described the timing parameters describing actions, and how they are used by the simulation engine to propagate time forward. This section describes additional timing parameters considered by the simulation engine that describe aspects of resources and of the agent models.

A. Time Parameters in Resources

When an action sets a resource value, a specified '*setting delay*' defines how long from current time until the new value will be listed in the resource. This replicates, for example, inertia in a resource or delay between an initiating action and its conclusion. For example, when an aircraft landing gear is deployed it simply takes time before the locks engage and the landing gear is truly "down." This is implemented in the simulation engine by a dynamic record of 'future updates' within a resource that will be invoked once the simulation clock catches up to them.

'*Resource unavailability*' is defined by two parameters marking the start and end times during which a resource is "not available." This construct can be used by actions whose dynamics may be driven by resource availability. For example, in communications between pilots and air traffic controllers, the voice frequency can be listed as 'unavailable' for another to 'speak' on (set) during the duration of a voice communication initiated previously. The construct of (non-)availability is not absolute, however; continuing the example, a high priority voice communication may interrupt a lower priority one.

B. Timing in Representing Agent Behavior

While the knowledge needed to conduct the work is described in a work model external to any agent model, during simulations agents serve as the executers of actions. The default perfect agent model executes actions exactly and immediately; where appropriate, the simulation engine can also accommodate the addition of agent behavior as applied to all actions. This is similar to Laughery and Corkers concept of 'first-principle modeling of human-performance' in which the same aspects of human performance are applied by the simulation framework to all tasks to which they are assigned [9]. These extensions are discussed in more detail in [5], [8].

Of interest here are the timing dynamics which an agent model can add to the execution of actions. For example, an agent may add a simple '*execution delay*' which postpones the execution of its actions; this delay may be fixed or vary with workload or number of resources that the action gets and/or sets. Likewise, an action may be listed as having a duration during which it 'occupies' an agent and the number of actions currently 'occupying' the agent can be recorded and traced: when a new action is commanded it is added to this list and an action whose completion time has passed is dropped. Further examining this list of active actions, if the effective taskload 'requirement' of each is parameterized, the taskload of actions active within the agent can be summed, recorded and traced, with potential workload limits or saturation events flagged.

VII. Conclusion

The objective of this work, as noted in the introduction, is to simulate (and thereby predict) the behavior of real, complex, heterogeneous systems involving teams of agents in which situated cognition is fundamental to performance. In contrast to previous multi-agent approaches in which behavior and knowledge was encapsulated within agent and component models, this paper describes detailed models of the collective work (and interaction) of multiple agents towards clear work goals in which the work has inherent patterns and structures as established by the physical environment and by a procedural environment as defined by established work practices, procedures and regulatory requirements. At its most atomic, these simulations represent work as actions that evaluate the current situation, and then change environmental resources. Any number of agents may perform the work, with agent knowledge represented as the ability to perform the actions assigned them from the work model, perhaps 'managing' their actions in the face of excessive taskload and interruptions.

Thus, these models are not intended to validate any specific theory of situated cognition, but instead are driven by their engineering utility in supporting analysis and design of complex, multi-agent systems. Given the emphasis on dynamic simulation here, a unique aspect of this newly developed simulation engine is the explicit representation of time. Each action must be able to report its next update time. For actions describing work on and within the environment, this captures the natural frequency of the environmental dynamics; for actions describing teamwork, this mirrors the triggers and timing of inter-agent communication and coordination; and for decision actions selecting strategies, this mirrors the dynamics of affordances in the environment, the agent's attention to these dynamics, and the required precision in timing.

In models where work involves intricate responses to changes in the environment, the next-update-time declaration represents an interesting component of situated behavior. A perfectly-timed decision occurs at the instant the context changes. However, this presents two problems: first, computationally, in a simulation of multiple actions being performed asynchronously, such an exactly-contemporary decision requires synchronous updates of context and decisions at very fine time intervals, driving up computational load; second, conceptually, in real multi-agent complex systems such decisions are rarely perfectly timed, instead often being based on predictions of when context may change, and thus have some inherent dynamics. Thus, the next-update-time of a decision action is often well-modeled as a prediction, based on attributes of the work environment, of when the relevant contextual changes will occur. This prediction may be conservative (based on a worst-case estimate) or not, as reflects the trade-off between catching a contextual change quickly and requiring repeated interpretation of context. This provides a representation of the temporal aspects of situated cognition by modeling the triggering of decision actions based on properties of the environment (assuming the agent's sampling behavior is accommodated to these properties), as a timing aspect of attention to the affordances of the environment.

Similarly, a novel aspect of the simulation engine described in this paper examines the temporal relationships between actions created when one sets the value of an environmental resource at one time which another then samples ('gets') at a later time. In large models, these interactions can involve many actions at any particular time. If a 'getting' action requires exact, instantaneous knowledge of the resource's value, then the simulation engine will automatically command the 'setting' action first to establish a temporally-correct view of the environment. In addition, this simulation engine allows the modeler to specify the maximum allowable age of the resource value when assessed by an action beyond which it will be automatically updated; this provides another mechanism for increasing the fidelity of the temporal descriptions of actions' relationships to resources, as well as enabling computational efficiency.

Further timing constructs noted here allow for delays between the initiation of an action and when the values of environmental resources are changed, as well as noting times that an action makes a resource unavailable to other actions. Likewise, agent models may add their own temporal dynamics to actions, such as delays, and may need constructs such as the duration for which they are occupied with an action to enable models of their taskload and task management.

VIII. Acknowledgments

The models and simulation framework discussed here have been developed through the support of the NASA Aviation Safety Program's Intelligent, Integrated Flight Deck project, under the technical supervision of Michael Feary and Paul Schutte.

References

[1] A. P. Shah and A. R. Pritchett, *Lecture Note in Computer Science*. Berlin: Springer, 2005, ch. Environment Analysis: Environment Centric Multi-Agent Simulation for Design of Socio-Technical Systems, pp. 65–77.

[2] S. M. Lee, A. R. Pritchett, and K. M. Corker, "Evaluating transformations of the air transportation system through agent-based modeling and simulation," in *7th FAA/Eurocontrol Seminar on Air Traffic Management Research and Development*, 2007.

[3] A. R. Pritchett, S. M. Lee, and D. Goldsman, "Hybrid-system simulation for national airspace system safety analysis," *Journal of Aircraft*, vol. 38, no. 5, pp. 835–840, 2001.

[4] A. R. Pritchett, S. Y. Kim, S. K. Kannan, and K. M. Feigh, "Simulating situated work," in *2011 IEEE Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)*, Submitted.

[5] A. R. Pritchett and K. M. Feigh, "Simulating first-principles models of situated human performance," in *2011 IEEE Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)*, Submitted.

[6] K. Vicente, *Cognitive work analysis: Toward safe, productive, and healthy computer-based work*. Lawrence Erlbaum Associates Mahwah, NJ, 1999.

[7] E. Roth, *Handbook of Cognitive Engineering*. Oxford University Press, 2011, ch. Cognitive Work Analysis.

[8] A. R. Pritchett, *Handbook of Cognitive Engineering*. Oxford University Press, 2011, ch. Computer Simulation of Aviation Safety.

[9] K. Laughery Jr and K. Corker, *Computer modeling and simulation of human/system performance*, 2nd ed. New York: Wiley, 1997, pp. 1375–1408.