Design Patterns for Balancing Behavior Responsiveness

Randolph M. Jones and Robert E. Wray
Soar Technology
3600 Green Court, Suite 600
Ann Arbor, MI 48105
rjones@soartech.com, wray@soartech.com

Keywords:
Interactive behavior models, Intelligent Systems, Responsiveness, Tradeoffs, Design patterns

ABSTRACT: *Models of intelligent, interactive, human behavior must be responsive. Three important dimensions to responsiveness are suitability, timeliness, and footprint. Responsive behavior models must use (usually significant amounts of) knowledge to make suitable decisions in a dynamic environment, but they must manage their knowledge and make these decisions in a reasonable amount of time, while using a reasonably small memory footprint. This paper examines approaches to representing knowledge in behavior-representation systems in ways that maintain the richness and suitability of the knowledge base, but ensure reasonable response times and memory footprints. We present design patterns that we identified during improvements to the timeliness and footprint of two behavior-representation systems. We present these systems as "case studies" to illustrate how the design patterns were used to improve responsiveness. Although the case studies were implemented in the Soar cognitive architecture, we generalize the design patterns to apply across behavior-representation architectures that rely on associative pattern matching for long-term knowledge retrieval and response.*

1. Introduction

A hallmark of intelligent, interactive, human behavior is that it is responsive to the environment (Newell, 1990). Particularly in complex and dynamic environments, intelligent behavior reflects an ability to react to situations and make good decisions quickly. Thus, for a human behavior model to be considered intelligent, it must also be *responsive* to the environment. We describe three interacting functional requirements associated with responsiveness: *suitability*, *timeliness*, and *footprint*. The degree to which a behavior model meets these requirements depends on the interaction between the model's computational architecture and knowledge representations. The three requirements also impose tradeoffs on the best knowledge representations for a behavior model. For example, the requirement for suitable decisions demands a large and rich knowledge base. The requirement for timely decisions demands efficient indexing and processing of knowledge. The requirement for a small memory footprint demands minimal intermediate data structures for accessing and using knowledge.

Design patterns (Gamma et al., 1995) are an approach to software engineering that fosters reusability at the conceptual level, as opposed to software modules that foster reusability at the implementation level. A design pattern is specified to a fine enough level of detail that it is easy for an engineer to apply to new problems. But it is at an abstract enough level that it can be applied independently of a particular software architecture or programming language. Based on our experiences building and optimizing a number of intelligent, interactive, human behavior models, we have defined design patterns that satisfice suitability, timeliness, and footprint tradeoffs. This paper illustrates the design patterns via two example behavior models that we have optimized for responsiveness. Although the behavior models we have improved were developed in the Soar architecture (Laird, 2012), we contend the design patterns are general to behavior modeling architectures that use associative pattern-matching engines to index and retrieve knowledge.

This work contributes a codification of design patterns that can be used to tune complex knowledge bases to particular behavior architectures and system requirements. These patterns thus enable the development of behavior models that are responsive along the three dimensions of suitability, timeliness, and footprint for practical use in operational systems.

2. Responsive Behavior Representation

The primary components of a behavior model are a computational architecture and the knowledge base the architecture operates on. The task environment provides the context that drives the architecture's knowledge retrieval and decision making. Retrieval patterns are determined in large part by the particular structure of the *knowledge representation,* which in turn can impact the efficiency of the architecture's execution. To ensure responsiveness, it is imperative not only to design appropriate *knowledge content*, but also appropriate knowledge representations. This section describes three responsiveness dimensions and their interactions.

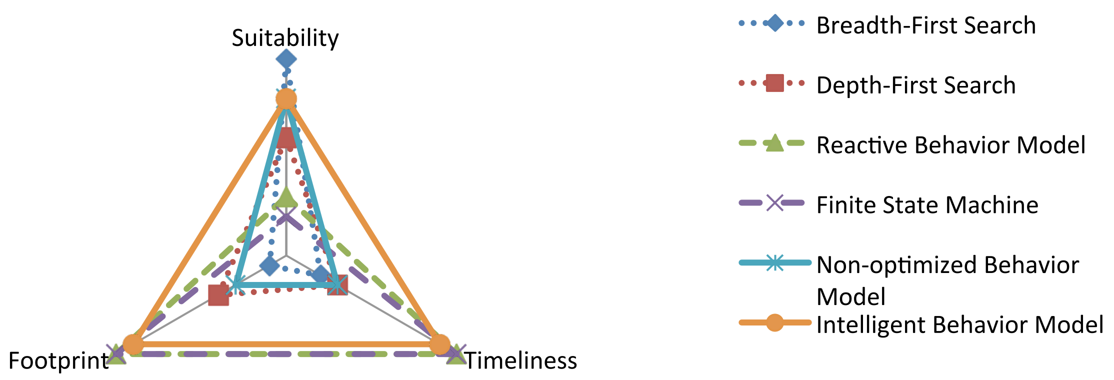

Figure 1. Dimensions of responsiveness. Search algorithms usually generate highly suitable results, but consume large amounts of time and memory. Simple behavior models are fast and use small amounts of memory, but they are limited in the range of responses they can generate. Intelligent behavior models must be responsive along all three dimensions.

2.1 Suitability

Suitability is the ability of a behavior model to generate good decisions, especially in response to a changing environment. Suitability represents the overall quality of behavior, as well as the ability to adapt intelligently as the situation changes. The primary goal for many behavior models is to make suitable decisions. But this goal must be achieved with solutions that also meet other requirements of the application.

A key contributor to suitable behavior is knowledge. A sufficient amount of appropriately represented knowledge is necessary to make suitable decisions in a wide variety of situations. Some behavior generation can be accomplished using lean knowledge content. However, more extensive knowledge is necessary for applications requiring responsiveness to a huge variety of exceptions and special cases, as well as an ability to take advantage of opportunities, to handle interruptions and unexpected events, and to coordinate with teammates, among other capabilities (Jones & Wray, 2006). But even a large knowledge base is insufficient if the knowledge is represented poorly.

2.2 Timeliness

Timeliness is a behavior model's ability to make decisions without "falling behind". Timeliness does not necessarily require hard guarantees of response time. It may be okay for a model to make some decisions slowly, as long as it can compensate or catch up later. On the other hand, for time-critical decisions, it is essential that the model deliver good enough decisions in a small enough amount of time.

Laird et al. (2011) argue that average response time/decision can be less important than maximum response time/decision. That is, problems in timeliness may not arise from being slow in general, but from occasional surges in activity. Laird et al.'s experiments used 50 msec as the target maximum allowable decision time for interactive behavior models. Note that we are distinguishing here between *decision cycle time* (the time an architecture spends processing new input before sensing the environment again) and *response time* (the time it takes for an architecture to generate a behavior-level response). A response may take several decision cycles, but both must be fast enough for the model to be responsive.

2.3 Footprint

Footprint is the amount of run-time memory consumed by a behavior model. Memory footprint impacts responsiveness because memory consumption produces processing delays in standard hardware architectures. Processing delays arise from the behavior architecture (processing intermediate knowledge structures for decision making) and from the operating system (managing memory allocation for memory-intensive systems). In the worst case, large memory footprints can cause behavior models to crash for freeze up, which could be seen as total unresponsiveness.

Footprint is also a concern for scalability. If an application requires hundreds or thousands of behavior models to run on a single machine, and each model consumes large amounts of memory, it may be necessary to reduce memory usage in order to meet the scalability requirements. In this way, footprint can impact the responsiveness of a scalable population of behavior models, even if it does not directly impact the responsiveness of a single behavior model.

2.4 Tradeoffs

Figure 1 illustrates how different types of behavior systems address tradeoffs between suitability, timeliness, and footprint. Past research (e.g., Jones et al., 1999; Laird, Jones, & Nielsen, 1998; Wray & Jones, 2005) demonstrates the strong role of knowledge to represent wide ranges of suitable responses in dynamic environments. Empirical results demonstrate the existence of knowledge-rich systems that address suitability, timeliness, and footprint (e.g., Jones, Furtwangler, & van Lent, 2011; Laird, Derbinsky, & Voigt, 2011). In many of the systems we have developed, the first priority has been to build systems that generate suitable responses. However, once the models become complex enough to start meeting suitability requirements, we almost always have to begin focusing on model inefficiencies in timeliness and footprint.

Knowledge representations that generate suitable decisions can be expensive. As an illustration, search-intensive planning and scheduling systems often guarantee high quality decisions. But they achieve this through processing that is quite expensive in terms of timeliness and footprint. Even though they generate highly suitable responses to wide varieties of situations, they are not responsive enough overall to serve as intelligent, interactive behavior models.

At another extreme are purely reactive behavior models, which maintain no internal state and guarantee rapid response times. While these systems are quite responsive in terms of timeliness and footprint, it has not been demonstrated that they can support the broad variety and richness of decision making that is necessary to make suitable decisions in complex human-reasoning applications. Thus, for most intelligent, interactive, human behavior models, we must trade off suitability, timeliness, and footprint to achieve the overall desired levels of responsiveness.

3. Case Studies

Although we have optimized many behavior models, the design patterns presented here were synthesized from a retrospective analysis of two case studies. In both of these cases we carefully documented the types of refactoring we identified, and abstracted those improvements into responsiveness design patterns

The strategy for both case studies was to start with a modeling architecture (in both cases, the Soar architecture, Laird, 2012) and then build a knowledge base that generates suitable decisions in a wide variety of situations and environments. We then profiled the system across scenarios and identified inefficiencies in model responsiveness. In the first case study (TacAir-Soar), we focused on timeliness inefficiencies. In the second case study (Dynamic Tailoring Monitor), we focused on footprint inefficiencies.

In both cases, we refactored the models to improve the responsiveness by orders of magnitude. More importantly for the work presented here, we cataloged the strategies we identified and generalized them into a set of design patterns for optimizing behavior model responsiveness in general. We continue to apply these patterns to new behavior models as we develop them.

3.1 TacAir-Soar

TacAir-Soar is the most complex, rich, sophisticated, and capable behavior model we have developed (Jones et al., 1999). TacAir-Soar is a model of fixed-wing combat aircraft pilots, and it contains the knowledge necessary to understand combat situations, make tactical decisions, and successfully perform a wide variety of air-to-air, air-to-ground, and other missions.

TacAir-Soar has been in development since 1992, using an iterative design and implementation process. An early version of TacAir-Soar included an initial set of air-to-air mission capabilities and cross-cutting mission capabilities such as situation understanding, mission planning, navigation, communication, and team coordination. Once the knowledge base was rich enough to generate suitable responses across the initial mission set, we began profiling the system on increasingly complex scenarios.

It was at this point that we sometimes ran into responsiveness issues. The issues largely had to do with timeliness. Example problems included situations where a number of targets suddenly appeared on radar, or managing intercept geometries that included numerous friendly and enemy agents. Through the development of a set of timeliness optimizations, we were able to enhance the model behavior and ensure that it remained responsive even in situations requiring significant attention to numerous details.

3.2 Dynamic Tailoring System Monitor

The Dynamic Tailoring System Monitor is a reusable behavior model that can be used to monitor learner behavior in a training application (Wray & Woods, 2013). The Monitor is configured with a set of instructor-defined patterns that represent expectations and constraints on the behavior of learners in a variety of situations. The Monitor checks these patterns against observed learner behavior and creates interpretations reflecting the status of observed behavior as compared to expected behavior. For example, the Monitor can indicate that a learner has completed some specific step in a procedure that is being practiced, or that the learner has violated some prescribed bound on behavior, such as not launching a weapon within a few seconds of reaching the launch acceptability region.

We have used the Monitor within a larger instructional support system (the Dynamic Tailoring System) to respond to unexpected learner behavior and to tailor learner scenarios in response to Monitor observations (Wray & Woods, 2013). We have also used the Monitor independently of other Dynamic Tailoring System components to support the delivery of feedback and direct instruction (Wray, Woods, & Priest, 2012).

Through a variety of experiments with the Monitor, we discovered that it developed an unreasonable memory footprint in certain environments, depending on the structure of the monitoring constraints used for model configuration. The memory inefficiencies arose in part because the monitoring activities result in the development of large numbers of intermediate results (behavior-level inefficiency) but also because the many-to-many matching requirements to compare observed and unexpected behaviors sometimes leads to exponential growth in the number of partial matches generated by Soar's pattern-matching algorithm (architecture-level inefficiency). We developed a set of scenarios to stress the footprint inefficiencies in the monitor and developed a number of strategies for identifying and eliminating these inefficiencies.

4. Timeliness Design Patterns

This section describes the individual design patterns we have identified and developed for improving the timeliness of a behavior model's responses.

4.1 Reduce input frequency

Frequent changes to perceptual input to the model lead to performance bottlenecks. When the world is not changing, an interactive behavior model does not usually have much work to do. When a model is informed of a change in the external environment, there is at least a possibility that something significant has happened. In order to generate responses that are suitable to changing input, the model must do some processing each time there are changes to the input.

Inefficiencies arise because the model cannot always know in advance whether a particular input change is actually significant, requiring a new response. This can sometimes only be determined by reasoning about the new input. This means that sometimes the model may waste time on input that has no ultimate impact. This is an opportunity for the model developer to build into the model (or into the model's I/O interface) assumptions about when input changes are significant.

There are a variety of ways that the frequency of model input can be reduced, subsequently reducing input processing. One strategy is to make input frequency a configuration parameter. For example, if the model builder knows that suitable responses can be generated when the input system is only updated once a second, then there is no reason to update the input system more frequently. This approach can also be used to set the update frequency of individual input parameters, rather than the input system as a whole. A similar approach is to round input values and only update them when their rounded values change. For example, when tracking fighter aircraft, it may not be necessary to track every meter of movement by the aircraft.

Both of these types of changes to input frequency could also be put under the control of the model itself, with sufficient knowledge to reason about situations in which input should be updated at higher or lower frequencies. For example, the TacAir-Soar system includes an attention mechanism that allows it to pay attention to certain geometric relations only for high-value targets. It can also instruct the input system to round different input parameters to different amounts, depending on the situation. The idea is that the model will select high-frequency input only in those situations where high-frequency input is essential. Additionally, the model can reflect on its own information requirements and determine situations in which some input information is not necessary to update at all.

4.2 Reduce input interpretation

After receiving new input updates, a behavior model may do early processing of the input. As a simple example, in the air combat domain one input element may be the angle of a target contact from the model aircraft's nose. One of the first things the model might do is translate that numerical input into a "left or right" designation. There are multiple other ways that numeric or symbolic input information can be "re-represented" for rich situation understanding.

This optimization design pattern identifies chains of input interpretation that may be superfluous. It limits input processing to occur only when it needs to, to the extent possible. Using the example above, there is no need to determine the left/right status of an input target if that status never actually gets used in any subsequent decision making. Again, to reduce the amount of input processing, the model can make its own decisions about which information needs to be computed.

Another optimization moves some types of input processing into the model's input interface, where it may be more efficiently computed. For example, an early version of TacAir-Soar used modeling patterns to continuously compute intercept geometry values, such as target aspect, lateral separation, and collision course. These mathematical computations can be computed much more efficiently if they are compiled into C code in the model's input application interface. Mechanical computations of this sort, which do not require the reasoning capabilities of the model, are candidates to be moved to a more efficient portion of the model architecture. These separable computations then also become modules that can be reused in other models.

4.3 Stabilize situation understanding

Additional sources of inefficiency in timeliness appear further along the decision-making pipeline. After a model generates intermediate interpretations of new input updates, those intermediate representations themselves feed into subsequent reasoning chains. Any degree to which the model can reduce the number of reinterpretations of intermediate representations will improve the timeliness of the model's responses.

One particular form of this inefficiency is "blinking". Blinking occurs when some entailment of the environmental situation straddles a boundary condition. Because the entailment keeps crossing the boundary, the model keeps changing its interpretation. This is particularly inefficient when each boundary change leads to a cascade of subsequent model interpretations.

The pattern for reducing blinking is to make boundary conditions conservative and fuzzy. As an example, the model may need to compute whether a target is in weapons range. If the target is right at weapons range, its status may change from second to second. However, the model could instead decide that the target is only "in range" when it is 1000 meters closer than the actual weapons range. And the target changes back to "out of range" only when it is actually outside of the weapons range. Because the boundary is now "fuzzy" and 1000 meters wide, there is much less opportunity for blinking. This in turn prevents the potential cascades of reinterpretation that can result from values that blink frequently. Determination of such fuzzy boundaries could also be put under control of the behavior model, or acquired by a learning system.

4.4 Fine-tune pattern matchability

Inefficiencies with timeliness of response can arise when the knowledge in a model is too general. This can be a tough tradeoff, because generality of knowledge is an attractive and effective way to ensure a model generates a broad range of suitable responses. Inefficiencies arise when knowledge retrieval is overly general. The solution is to contextualize the knowledge as much as possible, so its retrieval conditions only bring the knowledge to bear in those situations where it is actually relevant.

This particular pattern should be used conservatively. A significant limitation on contextualizing knowledge by hand to eliminate inefficiencies is that the model developer may also be eliminating the ability of the model to use the knowledge in novel situations that the model developer did not anticipate. One of the primary strategies for improving model response suitability is to generalize knowledge so it can generate intelligent responses in unexpected situations. Depending on the application, this ability to handle novel situations robustly may be much more important than eliminating the inefficiency of using general knowledge in situations where it turns out not to be useful.

Solving this tradeoff is currently an art, best handled by experienced model builders. However, there are opportunities to investigate learning algorithms and the use of default strategies so that the model can discover for itself when general knowledge should be used and when it should be contextualized to restrict its use. As an example, the Cascade system (VanLehn & Jones, 1993) falls back on general knowledge only when it cannot find suitable responses with its operational knowledge base. When it successfully applies some general knowledge, it immediately contextualizes the use of that knowledge so that it will not be used in over-general ways in the future.

4.5 Minimize response overhead

This design pattern exploits knowledge of the architecture on which the model is running. Every model architecture has some fixed processing overhead associated with each *decision cycle*. Ordinarily, a complete decision cycle starts with input updates to the model, and does some reasoning until the next input update. Because there is some architectural overhead associated with each cycle, it may make sense in some cases to put as much reasoning as possible into a single decision. The logic is that responses that can be generated in one decision cycle only incur the overhead once, where responses that take multiple decision cycles incur the overhead multiple times. Thus, the model developer can look for opportunities to do as much reasoning as possible in a single decision cycle, without needing to refresh the input.

One obvious tradeoff to this approach is the danger of putting so much processing in a single cycle that the decision cycle itself is too slow. If a single decision cycle is too expensive to be responsive, it would be better to split the reasoning across decision cycles and incur the additional overhead. This is consistent with Laird et al.'s (2011) observation that maximum decision cycle time can be a more important indicator of responsiveness than average decision cycle time (as long as average decision cycle time is sufficiently low).

Another danger of this pattern is damaging the ability of the model to generate suitable responses in dynamic environments. If there is significant processing within a single decision cycle, it may be the case that the model sometimes over-commits to a particular response. If the reasoning were split across multiple decision cycles, the model might generate a different response when the environment changes in important ways during the reasoning process. Reasoning chains that occur entirely within a single decision cycle do not get the chance to adapt as the environment changes (as reflected by updates to the input system).

4.6 Reduce uncertainty

A final design pattern for improving timeliness is to eliminate uncertainty in the model's reasoning. This pattern is related to variations of the patterns already discussed. A behavior model *may* be engaging in irrelevant reasoning any time it has to make guesses about its interpretations. When uncertainty is present, there is not necessarily any way for the model to know whether certain reasoning paths are going to be relevant, so it may have to reason "just in case".

Any intelligent behavior model should be capable of *some* reasoning under uncertainty, so it is not desirable to try to eliminate it in all cases. The pattern here is to look for *some* situations where uncertainty can be eliminated, especially when making the "best" decision may not be as important as making a "reasonable" decision in a more timely way. As an example, a wingman may not need to have continuous contact with a lead aircraft in order to stay in formation.

5. Footprint Design Patterns

This section describes the individual design patterns we have identified and developed for improving the memory footprint of a behavior model's responses.

5.1 Reduce accumulations

One essential part of intelligent behavior is situational understanding. One component of situational understanding involves keeping track of intermediate interpretations of the situation that feed into future understanding and decision making. The more sophisticated a behavior model is, the more complex its representation of situation understanding will likely be. Depending on the structure of the situation representation and the need to maintain a history of situation interpretations, this can lead to accumulations of information in a model's memory. In order to broaden the range of suitable responses generated by a model, it can be tempting to maintain arbitrary amounts of situation-understanding information, just in case it turns out to be relevant to future decision making.

However, unrestricted accumulation of situation representations will certainly lead to footprint problems. As an example, the Monitor keeps track of clusters of patterns that combine to trigger expectations and violations. (A "cluster" corresponds to a collection of actions related to some specific mission objective, like intercepting a bandit or flying formation as a wingman.) Some patterns are temporal, so the monitor must retain intermediate pattern results, just in case they become relevant in a future cluster. If there are numerous patterns and clusters being monitored, this leads to significant memory usage.

The design pattern for this situation is to eliminate as much historical information as possible, while maintaining the ability of the model to generate suitable responses. In the Monitor, we achieved this by deleting all intermediate pattern-matching results each time a full cluster was completed. The detection of a new cluster completion starts from scratch, but this is consistent with the monitor's functional specification.

Accumulated historical results are sometimes necessary for the model to function. In such cases, it may be necessary to introduce an external memory to the model. This modeling approach is well grounded. Humans do not maintain numerous intermediate results in short-term memory, so they use external aids (even pencil and paper) as temporary storage media. A behavior model use external memory, as well. If the external memory stores information more compactly than the behavior architecture does, the model's memory footprint will shrink. Behavior architectures may provide architectural support for more compact memories. For example Soar's semantic and episodic memories can be used for intermediate storage.

5.2 Use direct knowledge indices

Pattern-matching behavior architectures (like Soar and others) use sophisticated pattern-matching algorithms in order to ensure timeliness. These architectures support efficient many-to-many matches, so that long-term knowledge can be rapidly triggered by complex input updates and situation representations. In order to achieve timeliness, these pattern-matching algorithms cache intermediate results and partial matches. This is an architecture-level accumulation of representations, in contrast to the previous design pattern, which addressed behavior-level accumulations. Depending on the structure of the representation patterns, the accumulation of partial matches can be quite large, producing an unacceptable memory footprint.

One of the knowledge-representation patterns that can produce large numbers of partial matches involves generic indices to long-term knowledge. For example, suppose a model's long-term knowledge contains a hierarchical ontology of object types. There may be, for example, a representation of an FA-18 aircraft's properties, which is in turn an instance of a fixed-wing aircraft, which is an instance of an aircraft, which is an instance of a vehicle, which is an instance of a movable object, which is an instance of an object. The knowledge associated with the FA-18 can thus be indexed in a number of different ways. The FA-18 information could be retrieved by a "vehicle" query, or a "fixed-wing aircraft" query, or an "object" query. A pattern match consists of a number of queries that must all be satisfied. Thus, queries that return large numbers of objects can produces large numbers of partial matches. If a pattern contains multiple large queries, this leads to a combinatorial explosion that can consume massive amounts of memory.

In general, maintaining multiple indices to information in long-term memory is essential to generating suitable responses. Fortunately, the structure of the long-term knowledge does not increase memory consumption; the structure of the patterns to be matched does. Patterns that access the most general indices to knowledge are likely to generate large numbers of partial matches. Thus, a good design pattern is to ensure that a model's associative patterns access knowledge using the most specific indices possible. For example, if the model knows that it is looking for is an FA-18, then it should use the FA-18 index to memory, instead of a more general index. However, as discussed Section 4.4, there is a balance between generalizing the knowledge to cover many (and potentially novel) cases and specialization to reduce partial matching.

5.3 Exploit underlying architecture

The design pattern described above works when the model knows which specific indices to use for knowledge retrieval. But to generate a wide range of suitable responses, there will almost certainly need to be general patterns that operate with the most general indices to long-term knowledge. In such cases, it may be much more difficult to find ways to reduce the number of partial matches for the offending patterns.

In such cases there are still approaches that can reduce overall memory footprint. These approaches depend on having intimate knowledge of the particular pattern-matching algorithm being used by the behavior architecture. As mentioned previously, a pattern consists of multiple queries. From a functional standpoint, the order in which queries match does not matter. What matters is whether all the queries match or not. However, from the standpoint of memory footprint, the ordering of query matches is significant. Some query orderings produce large numbers of partial matches, while other orderings of the same queries may produce very small numbers of partial matches. As an artificial example, if a pattern were to match all the names in a phone book that have a first name starting with R and are on page 23 of the phone book, there will be many more partial matches if "starting with R" is the first query than if "on page 23" is the first query.

If a model developer comes across a particularly expensive pattern, architecture-specific strategies can change the ordering of query matches to reduce the footprint. In the Monitor, we split individual patterns into two or more chained patterns, to force Soar's matching algorithm to use a desirable ordering. Another Soar "trick" is to make clever use of negated queries, because negated queries do not generate any intermediate partial matches. The general lesson is to exploit the architecture's algorithms to reduce the knowledge-retrieval footprint.

6. Summary and Conclusions

We have deconstructed the concept of behavior-model responsiveness into three functional components: suitability, timeliness, and footprint. We argue that intelligent, interactive, human behavior models must be responsive along all three dimensions. They must generate responses that are appropriate and that adapt appropriately to changes in the environment. They must ensure that maximum and average response times are small enough for the responses to remain suitable. And they must ensure that responses remain suitable without exhausting the system's memory resources.

In general, the primary goal of behavior modeling is to develop systems that generate suitable responses across wide varieties of situations. Such systems require large amounts of knowledge, as well as appropriate reasoning strategies for bringing that knowledge to bear in efficient ways. Large knowledge bases *require* associative pattern-matching algorithms for intelligent retrieval (imagine the world-wide web without associative search engines). Thus, once a model becomes sophisticated enough, with a large enough knowledge base to be "intelligent", it may run into inefficiencies in timeliness and memory footprint.

Using two behavior-model case studies, we developed several techniques for optimizing the timeliness and footprint of models that are already suitable. We have additionally generalized these techniques into design patterns that apply beyond a specific behavior architecture. We intend these design patterns to provide a toolkit for future model developers to ensure that their intelligent, interactive behavior models remain responsive across all three dimensions of suitability, timeliness, and footprint.

7. Acknowledgements

This work is supported in part by the Office of Naval Research. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of Defense or Office of Naval Research. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. This work was supported, in part, by the Office of Naval Research project N00014-1-C-0170 Tactical Semi-Automated Forces for Live, Virtual, and Constructive Training (TACSAF). We would also like to thank colleagues who have contributed to the conceptual development and implementation of the design patterns described here including Ben Bachelor, Jeremiah Folsom-Kovarik, Josh Haley, Robert Marinier, Charles Newton, Brian Stensrud, and Angela Woods.

8. References

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Jones, R. M, Furtwangler, S., & van Lent, M. (2011). Characterizing the performance of applied intelligent agents in Soar. *Proceedings of the 2011 Conference on Behavior Representation In Modeling and Simulation (BRIMS).* Sundance, UT.

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P., & Koss, F. V. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine, 20*(1), 27–41.

Laird, J. E. (2012). *The Soar cognitive architecture*. MIT Press.

Laird, J. E., Derbinsky, N., & Voigt, J. (2011). Peformance evaluation of declarative memory systems in Soar. *Proceedings of the 20th Behavior Representation in Modeling and Simulation Conference*.

Laird, J. E., Jones, R. M., & Nielsen, P. E. (1998). Knowledge-based multiagent coordination. *Presence: Teleoperators and Virtual Environments, 7*, 547–563.

Newell, A. 1990. *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

VanLehn, K., & Jones, R. M. (1993). Integration of analogical search control and explanation-based learning of correctness. In S. Minton (Ed.), *Machine learning methods for planning*. Los Altos, CA: Morgan Kaufmann.

Wray, R. E., & Jones, R. M. (2005). An introduction to Soar as an agent architecture. In R. Sun (Ed.), *Cognition and multi-agent interaction: From cognitive modeling to social simulation,* 53–78. Cambridge, UK: Cambridge University Press.

Wray, R. E., Woods, A., & Priest, H. (2012). *Applying Gaming Principles to Support Evidence-based Instructional Design*. Paper presented at the 2012 Interservice/Industry Training, Simulation, and Education Conference, Orlando.

Wray, R. E., & Woods, A. (2013). A Cognitive Systems Approach to Tailoring Learner Practice. In J. Laird & M. Klenk (Eds.), *Proceedings of the Second Advances in Cognitive Systems Conference*. Baltimore, MD.

Author Biographies

RANDOLPH M. JONES, PhD, is a senior artificial intelligence engineer at Soar Technology. Dr. Jones co-founded Soar Technology in 1998, as he has led the science and engineering efforts for a wide variety of human behavior models and their applications. Dr. Jones received his BS in mathematics and computer science from UCLA, and he received his MS and PHD in information and computer science from the University of California, Irvine. Dr. Jones has held research and teaching positions at Carnegie Mellon, the University of Pittsburgh, the University of Michigan, and Colby College.

ROBERT E. WRAY, PhD, is a senior scientist at Soar Technology. His research encompasses many areas of artificial intelligence including knowledge representation and ontology, agent-based systems and agent architectures, artificial intelligence in education, machine learning, and cognitive science.