Redux: Example-Driven Diagrammatic Tools for Rapid Knowledge Acquisition

Douglas J. Pearson
ThreePenny Software, LLC
Seattle, WA 98103, USA
douglas.pearson@threepenny.net

John E. Laird
EECS Department, University of Michigan
Ann Arbor, MI 48109, USA
laird@umich.edu

Keywords:
Knowledge acquisition, behavior acquisition, diagrams, visualization, example-driven, human behavior models, rule based.

ABSTRACT: *The last ten years has seen a revolution in the complexity and realism of human behavior models (HBMs). However, the cost of developing realistic HBMs continues to increase as much of the detailed and complex knowledge must be manually encoded to produce realistic behavior. The focus of this project is on reducing the cost of acquiring, validating and maintaining the knowledge used in realistic HBMs. Our approach is to develop tools that allow subject matter experts (SMEs) to specify behavior using abstract scenarios represented as diagrams. The SME can graphically describe the conditions under which actions and goals should be pursued, together with the associated reasons for those decisions. The system, guided by the expert's choices, analyzes and automatically generalizes from the example scenarios, alerting the SME to inconsistencies and missing knowledge. The system incrementally generates an executable HBM whose behavior the SME can view and modify during development. By moving the language of discourse from symbolic programming languages to annotated diagrams, the SMEs specify knowledge directly without requiring the intervention of a knowledge engineer to translate between the representations.*

1. Introduction

The cost of developing realistic human behavior models (HBMs) continues to increase as much of the detailed and complex knowledge must be manually encoded to produce realistic behavior. The focus of this project is on reducing this cost by developing tools that allow subject matter experts (SMEs) to specify behavior using example scenarios represented as diagrams. The distinguishing features of this approach are:

- Changing the language of discourse for developing, validating and maintaining HBMs from symbolic languages to diagrams.
- Driving the development process through example scenarios, where an SME walks through ideal behaviors, recording reasons for decisions and describing appropriate goals, actions and methods to pursue.
- Generalizing the examples through direct guidance from the SME in selecting features for when a particular course of action is appropriate, coupled with heuristics to assist in this selection process. This ensures that general purpose behaviors are acquired rather than simple, scripted scenarios.
- Generating and analyzing rules automatically to determine how well they cover the examples specified by the SME. Where differences arise, the SME is prompted to correct inconsistencies or fill in missing knowledge.
- Managing the knowledge during all stages of development from acquisition, through development, validation and maintenance within a single, unified environment.

We discuss these methods in the context of an ongoing project to develop realistic human behavior models of soldiers engaged in close quarters combat within buildings. To date our work has focused on the challenge of acquiring new behaviors as distinct from acquiring new internal or external representations of the environment, which we will pursue at a later point within this project.

2. The Challenge

The typical approach to knowledge acquisition and construction of a human behavior model consists of:

1. Review of relevant domain specific literature by the development team.
2. Interviews with a subject matter expert (SME) describing the overall task domain and then specific example scenarios with descriptions of decisions and actions.
3. Prototype knowledge-base development based on notes taken by knowledge engineering team
4. Intermediate evaluation of the HBM by the SME
5. Continued development cycles with knowledge engineers (KEs) adding new behaviors that are then reviewed by the SME for accuracy and completeness.
6. Validation of the final model by the SMEs
7. Extension and maintenance of the model during its useful life, adding behaviors to cover new tasks.

The most costly parts of the development process are usually steps 3, 5 and 7--the phases where knowledge engineers encode the behaviors previously described by the SMEs. Based on our experience of building large scale HBMs in the tactical air combat and MOUT domains [1, 2] 75%-90% of the effort was spent developing tactical and mission-specific knowledge. This experience directly motivated our current effort to build tools that allow SMEs to more directly encode their knowledge through examples of behaviors represented as diagrams.

3. A New Approach

The core of this approach is to minimize the role of the knowledge engineer as much as possible, to let the SME enter knowledge in a format friendly to his natural thinking, and to translate that representation automatically to an executable format.

The outline of our approach is:

1. The expert (SME) lays out a training scenario as a sequence of situations represented as diagrams, similar to a storyboard for a movie.
2. The expert then steps through each situation in the scenario, defining the desired behavior for the entities within this specific scenario.
3. The behavior defined in the example scenario is automatically generalized to cover more than the specific scenario being described by the SME.
4. Rules are automatically generated from the example scenarios and these rules are analyzed to determine how well they cover the library of training examples.
5. The training scenarios are saved in a library that can be examined (and modified) by other SMEs. The library of examples can be used for regression testing, as well as to determine if additional scenarios need to be considered by the SMEs.

The overall structure of the Redux (Rapid Behavior Acquisition from Diagrams Using Examples) system is shown in Figure 1 and in the rest of this paper we will describe these elements in more detail.

3.1 Behavior capture using diagrams

The biggest problem with current approaches is that there is a vast disconnect between the language used by the SMEs for describing behaviors and the language used by the knowledge engineers for building the HBMs. A long tradition of psychological research supports the idea that for some problem solving and thinking, diagrams are essential [3, 4]. Our hypothesis is that by specifying behavior through diagrams, the SMEs will be able to more directly encode their knowledge greatly reducing the time to develop HBMs.

The knowledge to be acquired falls into three categories:

- Goal knowledge: the goals and objectives of the entities, such as clear a room, defend a room, or retreat to safety.
- Behavior knowledge: the knowledge that determines which goals and actions should be taken to achieve the current goals given the current situation. This includes relevant doctrine and tactics.
- State knowledge: the features of the environment that are relevant to generating behavior, such as the weapons available to an entity, the location of doors and windows, available sight lines, areas under enemy control.

At this stage of our research project, the SME is able to specify new goal and behavior knowledge in terms of the existing state representation. The SME does this, not by directly writing code, but by stepping through diagrammatic representations of the example situations in the task domain. The current tool assumes that the state representation is developed in collaboration with the SME prior to behavior specification. We hope to relax this assumption in the future

.

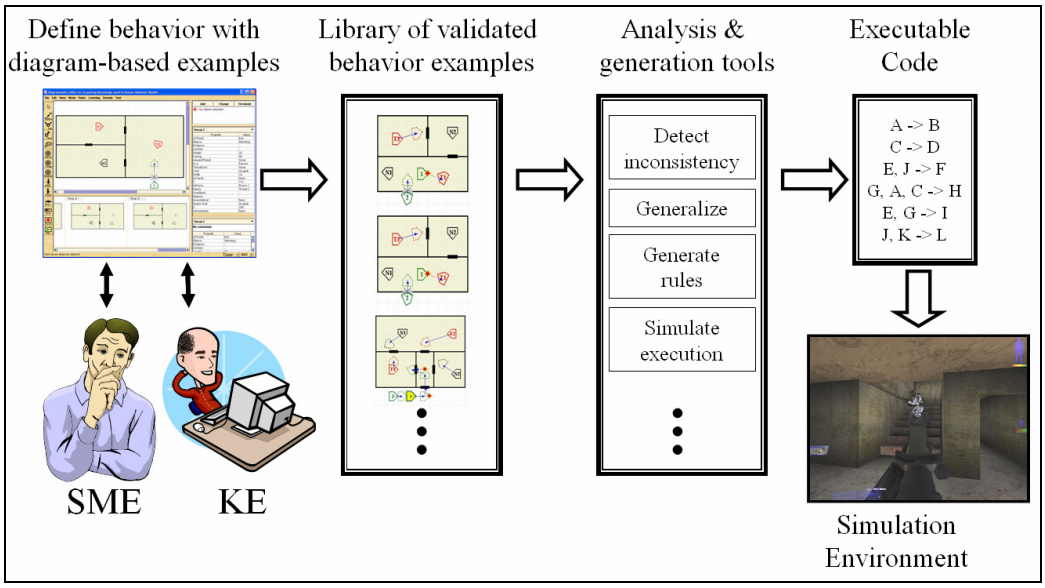

Figure 1. Redux System Architecture

3.2 Scenario specification by the expert

The first step in entering new knowledge is for the SME to create a specific situation that is the first step of a scenario. In our example domain, the SME lays out a series of rooms, doors, walls, kitchen appliances etc. The SME can also place participants, including friendly, opponent and neutral forces (see Figure 2).

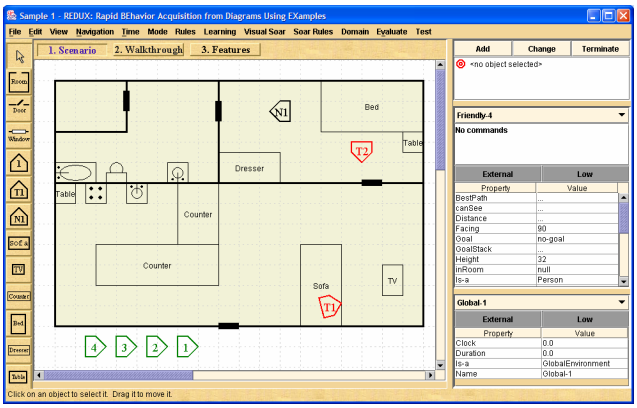

Figure 2. Scenario Specification

3.3 Example behavior specification

Once the first state of scenario is defined, the SME selects the appropriate behavior for each entity for its current task. These tasks can include situations assessment (e.g. determining defensive strong points), high-level tactical goals (e.g. defend a room) or low-level behaviors (e.g. move to a door). Redux then automatically creates the next state of the scenario, moving entities if appropriate, and the process repeats, with the SME specifying appropriate behavior step by step.

The majority of the information that the SME must have access to in specifying behavior is well suited to visual representations, such as the layout of a room, the positions of individuals, their actions etc. The physical aspects of the situation are directly represented in the diagrams and can easily be seen and selected by the SME. However, there are also more abstract data structures that are internal to the HBMs, such as abstract situational awareness, current objectives and individuals' attitudes concerning other members of the team, that are represented as attribute-value feature vectors. All of an object's or entity's features are available through menus as shown on the right-hand side of Figure 2 and 3. For some of these concepts, graphical representations may be possible that would be easier to use than the purely symbolic attribute-value representations.

Behaviors are defined by selecting specific actions from an available palette (e.g. add-new-goal or move-to-location). The user then parameterizes this action by clicking in the visual display (e.g. clicking on the room to be cleared or on the door to be moved to). The user can add new goal concepts, together with the parameters they require. The behaviors being defined are not limited to external, physical behaviors. The set of actions can include internal state changes, such as new situational awareness information (e.g. that a particular door is the most likely access point for an enemy).

The expert builds up a sequence of actions for the entities step by step. This forms a series of states (S_0, S_1, …). These states do not need to occur at fixed time intervals (e.g. every 10 seconds) but instead occur at points that the expert deems important. For example, the SME might define a series of small, detailed moves when defining the behavior for entering a door, but use longer moves for advancing down a corridor. Redux automatically determines the duration of each state by computing the maximum duration of all of the actions (A_0, A_1, …) within that

state. For example, a person walking 10 ft at 2 ft/s would take 5 seconds. The time at each state is then computed from the sum of the duration of all previous states:

$$time(S_n) = \sum_{i=0}^{n-1} duration(S_i)\ ;\ duration(S_i) = \underset{j=0}{\overset{m}{Max}}\, duration(A_j)$$

This approach to time allows the SME to provide an appropriate level of detail to different parts of the scenario.

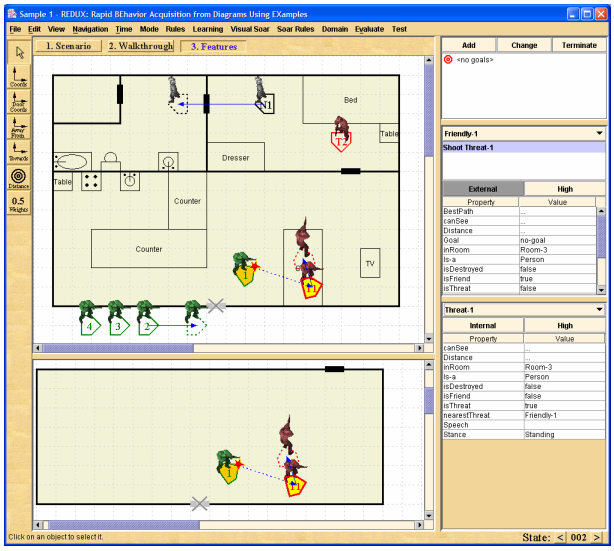

Figure 3. Behavior Specification with Entity View

The SME can also choose to view the situation in multiple ways. Most notable is the entity's view that shows what the selected entity can sense directly through all modalities. We plan to extend this view to show situational analysis, such as determining lines of fire, escape routes etc. Figure 3 shows an example of this (the lower window is the entity view).

In many domains, the entities should select actions with a certain amount of unpredictability. For example, when soldiers are training against computer controlled opponents, it is important that the opposing forces do not always follow the same tactics in a given situation or they will be too easily defeated. Redux supports this by allowing the SME to define multiple acceptable actions in a particular state and assigning weights to each path. The different actions become branches in the scenario, which allows the SME to efficiently describe multiple training examples within a single scenario.

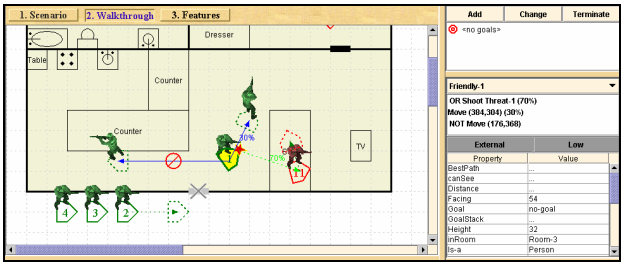

Figure 4. Example of conditional and negated actions

In addition to having the SME specifying multiple actions, the actions themselves can have multiple, mutually exclusive outcomes. For example, when one agent shoots at another, the outcome could be a fatal hit, a wounding hit, or a miss. This leads to additional branches in the scenario (which the SME can ignore if they are not tactically distinctive).

The SME can also define actions that should explicitly not be taken in a particular situation. Figure 4 shows an example of two weighted alternatives (30% to move, 70% to shoot) and that the entity should not withdraw in this situation. Sometimes it is more efficient to specify actions that should not be taken so that exceptions can be made to more general rules.

3.4 Example Generalization

If the knowledge base included only specific annotated examples, the knowledge would be very brittle, covering the specific training scenarios but little else. One of the major roles of a knowledge engineer in traditional knowledge acquisition is to generalize from the examples so that the knowledge covers a broader collection of situations.

Generalization is such a key area that we address it with multiple techniques. We will describe two methods we are using in this section and another proposed method in the future work section.

Our first approach to generalization is to have the SME explicitly step through the scenario, marking the features in the environment that are relevant to each decision. The more features that the SME selects, the more closely tied the acquired knowledge will be to the specific scenario. If the SME selects only a few features, the knowledge generated from the scenario will be very general and will cover many similar situations.

For example, when firing at an opponent a subset of the important features might be:

Table 1. Example relevant features

Object	Feature	Value
<target>	IsThreat	true
<target>	IsAlive	true
<shooter>	CanSee(<target>)	true
<shooter>	Nearest-Threat	<target>
<shooter>	Goal	Eliminate-Threats

The process of selecting these relevant features is potentially time consuming and error prone. Our second generalization technique uses a series of heuristics to reduce both the time spent and errors made. These heuristics attempt to identify features that are likely to be relevant to a decision and then the SME can further specialize or

generalize these automatically selected features. To continue this example, the heuristic associated with shooting someone might be to include:

Table 2. Approximate list of relevant features

Object	Feature	Value
<target>	IsThreat	true
<target>	IsAlive	true
<shooter>	CanSee(<target>)	true
<shooter>	Goal	<current-goal>

That is to say that whenever a "shoot" command is issued, these features will be included in the default set of relevant features (e.g. to shoot someone you should be able to see them). We currently assume that the set of included features is domain specific and is developed in consultation with the SME prior to behavior acquisition. The important aspect of these heuristics is that their predictions do not have to be correct, just close. The SME will review and adjust the set of features, removing ones that are not in fact relevant or adding others (e.g. the Nearest-Threat condition in this example). This initial 'guess' at the feature set reduces the workload for the SME.

This example also serves to demonstrate how the number of features selected affects the generality of the knowledge that is acquired. If the Goal feature is removed then the knowledge gained will apply any time the entity can see a threat, not just when the current task is to eliminate those threats. Conversely, if additional features are added (e.g. that the shooter is carrying a certain weapon) then the newly acquired knowledge will apply to a smaller range of situations.

A major research issue is how to correctly handle the situation where the important features relevant to a decision are not currently represented within the domain. We have not yet directly addressed this problem although our approach will be to allow the user to define new internal structure that the HBM can then use for its reasoning. For example, the SME might determine that a particular decision relies on the last known location of an enemy. If that property is not currently represented in the task domain, we must present a method for its inclusion together with a method to visualize this new piece of information. We expect that developing such techniques for dynamically extending the set of concepts will be one of the more challenging aspects of our future work on this project.

3.5 Automatic Rule Generation and Analysis

After the SME has defined the scenario, specified the desired behavior and reasons for that behavior Redux can automatically generate a set of rules based on the SMEs choices. This generation process is currently a direct mapping from the important feature sets. The rule generated from the example shown in Table 1 would be:

If (<target> ^isThreat true ^isAlive true) &
(<shooter> ^canSee <target> ^nearest-threat <target>
^goal eliminate-threats))

then
(<shooter> ^select-action <action>)
(<action> ^name shoot ^target <target>)

These rules are executed within Redux and compared to the behavior that the SME specified. If the SME did not accurately specify the list of important features for each decision then the behavior produced by the rules will not match the desired behavior that the SME specified. For example, if an entity shoots an opponent in a crowded room the SME should indicate that the reason for this was because of the target being an enemy (not a friend or neutral). If the SME forgets to do so, then when Redux simulates the entity preparing to shoot it will determine that the entity cannot uniquely decide which target to select and will prompt the SME for further clarification.

Redux also can determine that a rule is likely to be over-general or over-specific during the selection of important features by the SME. This is done by checking whether the rule created from the feature set would also apply to the state immediately preceding or immediately after the current state. If the rule does match in those states this is usually a sign that the rule is over-general and additional features should be specified so it only matches in the correct state. To continue our example, if the SME forgot to include the (<shooter> ^CanSee <target>) feature then this rule would match in the state before the shooter moved into the room. Figure 5 shows how the tool displays an error with the red stop light, signaling that the SME should correct the rule. This alert is only provided as advice to the SME as there are valid situations where a match will occur in neighboring states.

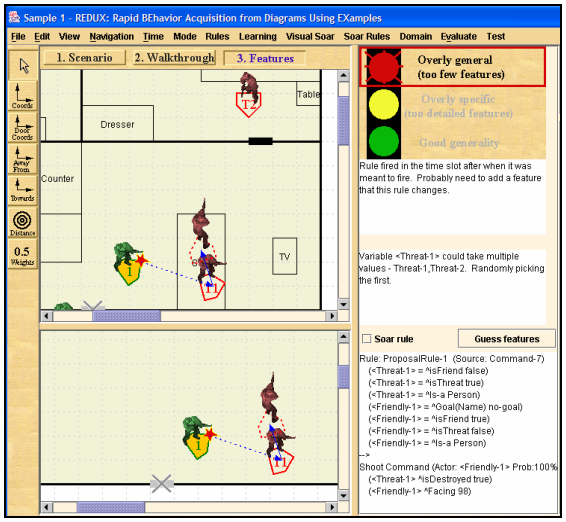

Figure 5. Immediate Detection of Errors

Rules that are likely to be over-specific can also be determined by categorizing certain features of the domain as being highly specific features. For example, it is unlikely that an entity will move to exactly the same location in two different scenarios, so a rule that includes an entity's exact position is probably over-specific.

This ability to detect errors in the knowledge base during the creation and storage of examples can save an enormous amount of time. In a traditional knowledge acquisition process, such an error is often not recognized until the knowledge engineers have invested substantial effort and the SME may have to be contacted again to explain what the correct behavior should be.

3.6 Rule Assisted Knowledge Acquisition

As rules are built up from prior situations and scenarios, Redux can use these during the knowledge acquisition process to control behavior of the agents, even before the SME has specified actions. Thus, the knowledge acquisition process becomes more of a collaboration between the tool and the SME, with Redux being able to generate behavior for familiar situations. This simplifies the SMEs job to being one of verifying behavior and filling in the blanks – places that the tool does not yet have sufficient knowledge to generate behavior.

3.7 Traceability and Validation

An important problem in any effort to acquire behavior models from experts is how to verify that the acquired knowledge has been accurately encoded in the HBM. By formally capturing the training scenarios as diagrams, we can both validate that the rules generate the desired behavior in all example scenarios as well as tracing the source of each piece of the knowledge base back to the specific diagram drawn by the SME that lead to its inclusion. If errors are detected, then the point of discussion is a specific concrete example as opposed to an abstract rule. Thus, an SME brought in to verify the system can examine both generated behavior and the original examples, and if there are disagreements, they can be readily settled by either modifying the scenario or generating new scenarios, with new correct behavior.

This approach compares very favorably to standard knowledge acquisition processes, where the final knowledge base is validated by running a series of test cases and having the results inspected by the SMEs. This testing can be expensive if the number of test scenarios is large and performing manual comparisons of the results is a potentially error prone process. Worse, when errors are discovered and changes are made to the knowledge base, the only way to reliably validate the new model is to repeat all of the tests and inspections again. Unfortunately, this final phase of regression testing is rarely done in current systems because it is prohibitively expensive. However, in our approach, the examples are always there and regression testing is a core part of the methodology.

3.8 Maintenance of the Knowledge Base

Knowledge acquisition typically focuses on the initial creation of a knowledge base. In practice with large scale HBMs, there is invariably a need to include new knowledge after the delivery of the model. A significant motivation for this project is that the SMEs for one of our behavior models (TacAir-Soar [1, 2]) have been frustrated by their inability to add new missions and tactics quickly and cheaply.

Our example-driven approach allows new knowledge to be added through the addition of new example scenarios. We hope that the tools are sufficiently easy to use that in many cases these additions can be made by the SMEs directly, without the involvement of knowledge engineers at all. The SMEs will maintain the library of example diagrams, rather than maintaining the underlying code. As new examples are added or existing examples are modified, the automatic analysis and validation steps described above will help ensure that changes do not break existing behaviors and introduce errors.

4. Evaluation

We are still in the relatively early stages of this project and are still developing the suite of tools but the first, preliminary, results are encouraging. We used Redux to define a simple scenario where a single entity moves through a series of rooms and takes up a defensive position. The time this took using the tool are as follows:

- Defining the scenario (22 rooms and doors) – 4.5 minutes
- Specifying the walkthrough (19 states) – 5 minutes
- Generating and validating a set of rules (12 rules) – 4.75 minutes

Total time: 14.25 minutes to move from scenario to rules for a single tactic developed by an expert Redux user. To put times such as this in context, we need to run comparative trials to determine how well an SME can use the tool without the help of a KE and how long it takes to manually encode a given tactic without use of the tools.

Once Redux is complete, we will evaluate it based on these trials:

1. Baseline trial. In order to create a baseline for comparison, we will have an SME specify, in advance, the requirements for a series of tasks in the MOUT Close Quarters Combat domain. This specification will follow the standard model of using written documents to convey the requirements to the knowledge engineer. The engineer will then use their exist-

ing development tools to complete as many of the tasks as possible within the available time.

2. Comparison trial. An SME, working in conjunction with a knowledge engineer, will use the new tool set to complete the same set of tasks. We will then compare the time taken to complete the tasks and the quality of the solutions.

We may also design and conduct additional comparative trials to evaluate how well an SME can function working alone with the tools or how well a novice user performs with the new tools.

5. Domain Independence

Although our examples and evaluation domain both focus on the close quarters combat domain, the suite of tools is largely domain independent. The only requirement for a domain is that we can build a visual representation of the task. In physical tasks this representation is typically a two-dimensional top-down view of the problem domain, but the tool only assumes that some such visual representation can be found (e.g. a 3-dimensional view or even a purely internal representation would also suffice).

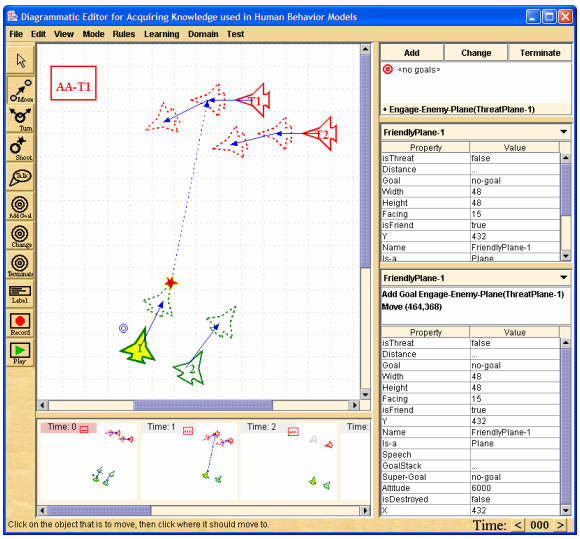

Figure 6. Air Combat Domain

In order to demonstrate that Redux is indeed domain independent, we applied the tool to the air combat domain in a matter of hours (see Figure 6).

6. Related Work

Visual Programming is the use of graphics to create computer programs [5]. There are a number of visual programming languages (e.g. ARK [6], VIPR [7], Prograph [8]) where the program code is visually represented and modified directly by the user. These languages focus on general purpose programming languages and tasks and therefore the elements represented are basic programming elements like classes, objects, methods, iterations and branches. These visual programming systems do not use inference or learning in the development of the underlying code, instead relying on the user to directly input all of the new knowledge.

Programming by Demonstration is a variation on visual programming, where the user demonstrates the desired behavior on sample data. For example, Peridot [9] allows a user to draw a desired interface and then "use" the prototype interface while the underlying behavior is induced by the system. Mondrian [10], Chimera [11] and Metamouse [12] are all examples of systems where the user demonstrates a sequence of graphical editing commands and the system learns new compound graphical procedures. The focus of these systems is also to learn general purpose programming languages (such as LISP). Although they use machine learning techniques to generalize from the sample instances, there is very little transfer of the learned knowledge to new situations. Learning how one dialog box functions has, appropriately, little effect on how another dialog box will operate. In the knowledge acquisition task for our work, transferring knowledge between different related scenarios is an important goal. Our use of a rich knowledge representation and complete AI architecture facilitates this transfer.

Visual programming focuses on the internal representations of the agent while programming by demonstration focuses on the external or task domain representations. Our approach combines these by representing both certain elements of the agent's internal goals and sensed state with external representations of the task domain. This combination gives the user greater insight into the agent's behaviors and reasoning allowing for better transfer of knowledge to the system.

Visual programming has also been used for specifying simple robotic behaviors, both in the game *MindRover* (www.mindrover.com) and in the legged robotic toy *Wonderborg* by Bandai. These systems show that visual programming can be used effectively by consumers to program simple behaviors without having to use computer languages. Our goal is to expand this type of programming to the complex behaviors required in HBMs.

Knowledge acquisition by assembling primitive components (e.g. [13, 14]) typically focuses on the acquisition of new concepts in the representation language and constraints between those concepts. Our work can also be seen as the composition of primitive components. But in contrast, we have focused initially on the acquisition of behaviors described in terms of an existing representation language rather than on extending the representation.

7. Future Work

There are many avenues for future research on this project, including:

During example generalization, the SME may fail to identify all critical features or might include some that aren't really important to the decision to select a particular be-

havior. To improve the quality of this feature selection process, we will use machine learning techniques to analyze the scenarios. The learner will identify commonalities across scenarios as well as propose possible specializations to the SME.

We currently test for inconsistencies and errors within a scenario. As each scenario is created, the rules generated from that scenario can be tested against all scenarios in the example library (not just the current scenario) to see if they produce contradictory behavior. This extension will allow us to detect interactions between scenarios that are typically difficult and time-consuming to identify and correct.

As we have mentioned earlier, the current approach assumes that the state representation is largely constant and developed in advance of behavior specification. A key future goal is to relax this assumption by providing tools to extend the representation language. The tools must be simple enough for the SME to make these additions, while being efficiently represented so the tool scales well to large and complex domains.

8. References

[1] Jones, R. M., Laird, J. E., Nielsen P. E., Coulter, K., Kenny, P., and Koss, F. Automated Intelligent Pilots for Combat Flight Simulation, *AI Magazine* , Spring 1999, Vol. 20, No. 1, pp. 27-42.

[2] Robert E. Wray, John E. Laird, Andrew Nuxoll, and Randolph M. Jones. Intelligent opponents for virtual reality trainers. In *Proceedings of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC) 2002*, Orlando, FL, Dec 2002.

[3] Larkin, J. and Simon, H. Why a diagram is (sometimes) worth 10,000 words. *Cognitive Science* 11:65-99, 1987.

[4] Shah, P., & Miyake, A. The separability of working memory resources for spatial thinking and language processing: An individual differences approach. *Journal of Experimental Psychology: General, 125*, 4-27, 1996

[5] Myers B., "Taxonomies of Visual Programming and Program Visualization," *Journal of Visual Languages and Computing*, Vol. 1, No. 1, March 1990, pp. 97 - 123.

[6] Smith, R. The alternate reality kit: An animated environment for creating interactive simulations. In *Proc. 1986 IEEE Workshop Visual Languages*, pp. 99-106, 1986.

[7] Citrin, W., Hall, R., and Zorn, B. Programming with visual expressions. In *Proc. 1995 IEEE Symposium Visual Languages*, pp. 294-301, 1995

[8] Cox, P. T. and Pietryzkowsky, T. Using a pictorial representation to combine dataflow and object-orientation in a language-independent programming mechanism. In Glinert, E. P., editor, *Visual Programming Environments: Paradigms and Systems.* IEEE Computer Society Press, Los Alamitos, CA, 1990.

[9] Myers B., "Taxonomies of Visual Programming and Program Visualization," *Journal of Visual Languages and Computing*, Vol. 1, No. 1, March 1990, pp. 97 - 123.

[10] Lieberman H.., “Mondrian: A Teachable Graphical Editor”. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge MA, 1993, Chapter 16.

[11] Kurlander D., “Chimera: Example-Based Graphical Editing”. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge MA, 1993, Chapter 12.

[12] Maulsby D. and Witten I.., “Metamouse: An Instructible Agent for Programming by Demonstration”. In Allen Cypher, editor, *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge MA, 1993, Chapter 6.

[13] Clark P., Hayes P., Reichherzer T., Thompson J., Barker K., Porter B., Chaudhri V., Rodriguez A., Thomere J., Mishra S., Gil Y. Knowledge entry as the graphical assembly of components. In *Proceedings of the International Conference on Knowledge Capture 2001,* Victoria, B.C., Canada, pp 22-29.

[14] Handschuh S., Staab S., Maedche A. CREAM: creating relational metadata with a component-based, ontology-driven annotation framework. In *Proceedings of the International Conference on Knowledge Capture 2001,* Victoria, B.C., Canada, pp 76-83.

Author Biographies

DOUGLAS J. PEARSON is a Software Architect and Founder of ThreePenny Software, LLC. He received his Ph.D. in Computer Science from the University of Michigan in 1996 where his thesis focused on machine learning of planning knowledge. He now runs a small software company developing commercial quality applications in a wide range of different fields.

JOHN E. LAIRD is a Professor of Electrical Engineering and Computer Science at the University of Michigan and Associate Chair of the Computer Science and Engineering Division. He received his Ph.D. in Computer Science from Carnegie Mellon University in 1983. He is one of the original developers of the Soar architecture and leads its continued development and evolution. His current research includes extending Soar through the addition of reinforcement and episodic learning. He is a founder of Soar Technology and is a Fellow of AAAI.