Multi-Agent Architectures that Facilitate Apprenticeship Learning for Real-Time Decision Making: Minerva and Gerona

David C. Wilkins
Center for Study of Language and Information
Stanford University
Stanford, CA 94305
dwilkins@stanford.edu

David Fried
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
fried@uiuc.edu

Abstract

This paper describes the Minerva and Gerona agent architectures, which have been designed to facilitate apprenticeship learning in real-time decision making domains. Apprenticeship is a form of learning by watching, which is particularly useful in multi-agent knowledge-intensive domains. In this form of situated learning, human and synthetic agents refine their knowledge in the process of critiquing the observed actions of each other, and resolving underlying knowledge differences. A major design feature of Minerva and Gerona is their method of knowledge representation of domain and control knowledge, both static and dynamic. Their representations facilitates reasoning over domain and control knowledge for the purpose of apprenticeship learning. This ability to reason over domain and control knowledge plays a central role in solving the global and local credit assignment problems that confront an apprenticeship learner.

Introduction

Apprenticeship learning is a powerful method that agents use in a multi-agent setting to refine their knowledge (Wilkins88, Tecuci98). Apprenticeship is particularly effective in knowledge intensive domains. This paper describes the key challenges for apprentices in the domain of real-time decision making. It describes the Minerva and Gerona multi-agent architectures, and shows how their design assists with meeting the challenges of apprenticeship.

In an apprenticeship learning situation, there are two problem-solving agents, either synthetic or human, with differing levels of competence. This is shown in Figure 1. One of the agents actively solves a problem and the other agent watches. The watching agent employs apprenticeship techniques to improve the agent with the lesser competence, which can be the either the active or watching agent.

The first major challenge of apprenticeship is *global credit assignment*, which is the problem of determining when an observed action of the active agent suggests a knowledge difference between the two agents. This is not easy, because there are *natural variations in solving a problem in a community of agents*. This places two difficult design requirements on the performance element of the watching agent. The first design requirement is that that performance element must be able to compute the range of actions consistent with acceptable problem solving styles. Only then can it know when an observed action is not part of the normal variation and thus a knowledge difference exists between the agents. The second design requirement is that the watching agent's performance element must be able to *recompute its preferred next best action* based on the observed action that was taken by the active agent, rather than the action that it would have taken. These global credit assignment problem has major implications for the representation of control and scheduler knowledge within the performance element.

The second major challenge of apprenticeship is *local credit assignment.* This is the problem of determining the specific and best knowledge repair to make in the agent with lesser competence, based on an unexplainable action of the active agent. The range of possibilities is quite large, ranging from missing or wrong knowledge, and domain or control knowledge.

The remainder of this paper is organized as follows: The next section presents the Minerva architecture, which was developed in the domain of medical diagnosis and ship damage control. The following section presents the Gerona architecture, which was developed in the domain of decision making for ship damage control.

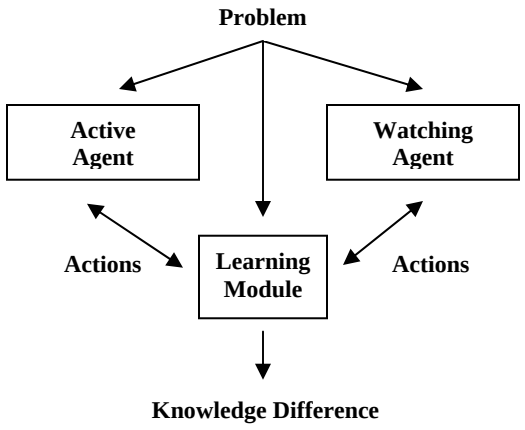

Figure 1. The Multi-agent apprenticeship learning paradigm. Agents can be synthetic or human. Either agent can be improved in the above scenario, depending on the apprenticeship mode.

Minerva Multi-Agent Architecture

Minerva is a multi-agent architecture for real-time decision making. The accomplishment of the Minerva project is that it started with a classical expert system architecture and modified its method of knowledge representation and inference to create a multi-agent architecture. The Minerva architecture supports a broad range of multi-agent explanation, critiquing, and learning capabilities, especially apprenticeship learning. Minerva is an evolutionary descendant of the Mycin and Neomycin architectures. In this section, we will review these systems and explain how they led to Minerva.

In the beginning was Mycin (Buchanan and Shortliffe, 1986). Mycin's major components were a rule base and a backward-chaining inference engine as shown in Figure 2. It had some but limited multi-agent capabilities. Specifically, the Guidon and Tieresias augmentations to Mycin allowed it to transfer expertise between itself and another agent, but only when Mycin was the active agent. The Guidon knowledge-based tutor queried the watching agent (a human novice) on what it believed from watching Mycin solve problems, and through a Socratic dialogue improved the watching agent's performance. Tieresias allowed a watching agent (a human expert) to track down a knowledge difference within a Mycin-like system and make a repair. The limitation of Mycin as a multi-agent system is that its domain rules were an admixture of domain, control, and scheduling knowledge.

Neomycin redesigned the representation and inference method of Mycin to provide greater multi-agent capabilities (Clancey, 1984). Neomycin separates strategy knowledge from the domain rule knowledge, as shown in Figure 2. This allows the same domain knowledge to be applied under many different problem-solving strategies, such as rule out hypothesis, group and differentiate hypotheses, and review of systems. A community of experts has these variations.

The Odysseus apprenticeship learning program worked in conjunction with Neomycin, and refined Neomycin's knowledge by watching a human expert (Wilkins, 1988).

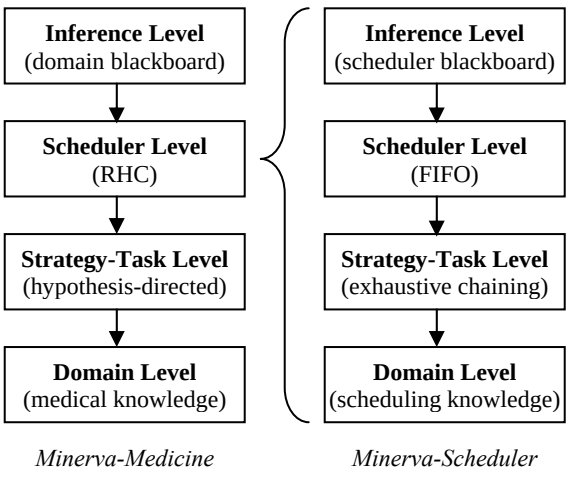

Figure 3. Use of Recursive Heuristic Classification for scheduling in Minerva.

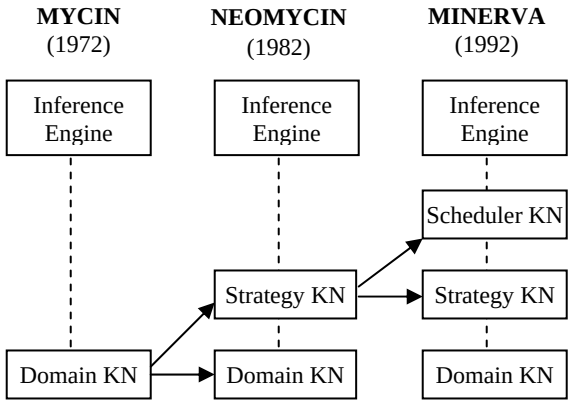

Figure 2. Improving Support for Multi-Agent Reasoning in Decision-Making Shells

Odysseus analyzes each action of a human expert diagnosing a patient. It solve the global credit assignment problem, by unifying the rules in the domain and strategy layers to generate all reasoning chains, which have with the expert's observed action as the root of the chain. If no reasoning chain is found that connects the observed action to an active goal, it assumed that there is a difference between Neomycin's domain knowledge and the knowledge of the expert.

Odysseus addresses the local credit assignment problem using the *metarule chain completion method*. This method involves generating all reasoning chains from the unexplained action to an active goal, and allowing one unification clause failure in the creation of this reasoning chain. The method successfully improved the Neomycin knowledge base. But it had a major weakness. In a large knowledge base, an "explanation" can too frequently be generated, since its not possible to distinguish between good and poor explanations. Thus, the explanation failures that signal the need for a knowledge base repair become too infrequent.

Minerva is a reworking of Neomycin to further increase multi-agent capabilities (Park, et al, 1992; Donoho, 1993). In Minerva, scheduler knowledge is separate from domain and strategy knowledge and placed in a separate layer, as shown in Figure 2. In Minerva, all explanation chains are generated as before by unifying domain and strategy knowledge. But then the scheduler reasons over these chains and gathers evidence for and against each chain using a process called *recursive heuristic classification* (Park, Donoho, Wilkins, 1993), as shown in Figure 3. The evidence against the chains is probabilistic, and after combining evidence this leads to all chains having a probabilistic weight. When an observed action is not in the top third of the strongest chains, an explanation failure is said to have occurred. Minerva and Odysseus use the local credit assignment method.

In Minerva, the embedded scheduler knowledge base is completely generated using induction over solved cases. A training example is generated for each action taken within an expert session, not just for each case (Donoho, 1993). An alternative method of accomplishing scheduling is by Envisionment of the alternatives (Bulitko and Wilkins, 2003).

The Gerona Real-Time Agent Architecture

The Gerona Agent Framework creates event-driven agents for real-time decision-making (Fried, et al, 2003). Its major components are shown in Figure 4. Information from other agents is sent to Gerona using an Event Communication Language (ECLs). Domain and control knowledge is encoded in Graph Modification Operators (GMOs); these are comprehensible to domain experts and are directly executed by the Gerona interpreter (Dhuse, 1992). All dynamic knowledge is encoded in a Causal Story Graph (CSG), which consists of a structured and annotated set of ECL statements. Meta-GMOs allow the agent to answer questions about its domain and control knowledge. MGMOs reason over the GMO and CSG structures, which have an explicit representation. The remainder of this section describes the ECL, GMO, CSG, and MGMO components. We then describe Gerona-DCX, a Gerona agent for damage control decision making with apprenticeship capabilities.

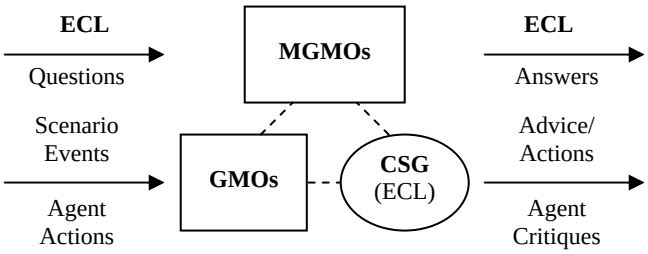

Figure 4. Organization of a Gerona Agent, showing inputs, outputs, and dependences.

An *Event Communication Language* (ECL) defines the entire range of language statements that can be exchanged with another agent. An ECL language encompasses the vocabulary of messages, orders, queries, and answers that can be passed between a Gerona agent and other agents, including communications about its own internal state and domain-specific knowledge. ECLs are organized into classes, and each class has a unique name and number.

An ECL class and an example instance in Gerona syntax is as follows:

```
report( 6801 "damage report (fire/flood/smoke)",
        [ source, compartment, problem ] )

report( 6801 "damage report (fire/flood/smoke)", 2:43,
        [ source = "Repair2",
          compartment="02-128-0-Q",
          problem = "fire" ] )
```

In this example, the ECL class is “damage report” with the ECL number of 6801. A particular agent, repair party 2, issues an ECL communication at time 2:43 that there is a fire observed in compartment 02-128-0-Q.

Each ECL class defines a set of properties that are assigned specific values when the class is instanced (an instance of an ECL class used for communication is also called an "ECL message"). An ECL class can be represented as a tuple $<Y, N, \Pi>$, where Y is the type of information represented (action, report, function, question, etc.), N is the ECL number, and Π is the set of properties, π_i. An ECL instance is represented as $<Y, N, T, P>$, where T is the timestamp that the instance was created, and P is a mapping from each of the π_i to one or more allowed values.

The *Causal Story Graph* (CSG) explicitly stores all of a Gerona agent's ECL communication with other agents and its beliefs about the other agents and their actions. It comprises all dynamic information required by Gerona to operate. A CSG is simply a collection of ECL instances, arranged as a tree, with some additional state and history information associated with each instance. The CSG gets its name because it associates information both by causal and logical relationships and as a chronological log (or story) depicting the agent's beliefs at every point in its execution.

A CSG node can be represented by the tuple $<Y, S, N, T, P, D, R, O, H>$, where the additional element S represents the state (which can have different values based on the data type), D is the unique *Node ID*, R is the Node ID of the node's parent in the CSG, O is a pointer to the operator that created the node or modified it to its current state (useful in allocating local credit), and H is a history of the previous states of the node.

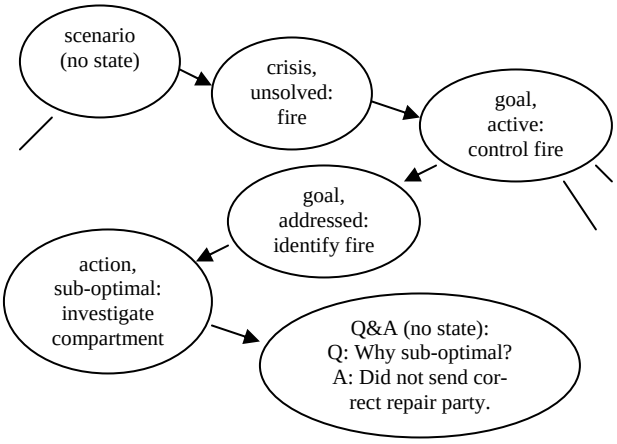

Figure 5. Sample CSG excerpt from a Gerona-DCX agent. The Gerona system is observing a human novice problem solver. It has critiqued the novice’s action “of investigate compartments” as suboptimal, and explained the reason for the critique.

The CSG also stores credit assignment information. The sub-optimal action node in Figure 5 represents a critique of an order given by another agent to investigate a compartment. In this case, the O component of the CSG node would later allow the system to locate the specific conditions that led to the critique (see the section on MGMOs below). Action nodes with states *correct*, *error-of-omission*, *error-of-commission*, and *late* are critiques of other agents. Action nodes with state *pending* are recommendations of correct actions by Gerona to other agents.

***Graph Modification Operators* (GMO)** encode all information relevant to receiving and responding to communication with other agents. Each GMO responds to a class of incoming messages and as its name suggests describes the graph operations to apply to the CSG. A single GMO may modify one or more existing CSG nodes or create a single new CSG node. Global credit assignment information is stored in the CSG, and GMOs encode all knowledge for assigning global credit.

A GMO consists of two parts. The first is a header that describes the communications it responds to. This includes a single ECL class and a set of conditions that the properties of an incoming message must satisfy. Using the example of the "damage report" ECL class described above, we might have a separate GMO for this report when the *problem* reported is "fire" and one for when *problem* is "flooding".

The second part of the GMO is a collection of nested if-then rules, which are evaluated in parallel, with any discrepancies between outputs handled in the simplest way possible. Because of this arrangement, the effects of a rule depend only on its own preconditions and post-conditions. Rules are composed of *Graph Clauses,* also called *G-Clauses*. G-Clauses are simple, compact elements that describe information queries or graph modifications. Their functionality is based on three types of operation:

- *find* operation – search for previously-recorded communications and inferences or query static information.
- *modify* operation - modify one or more CSG nodes.
- *create* operation – create a CSG node.

The G-Clauses become the building blocks of GMO rules along with a few simple control structures, including negation, disjunction, and for-each loops. Execution is forward only without backtracking; if more than one result could be bound to a variable, the set of possible results is bound. Evaluation of a GMO can be shown to be worst-case $O(E^2)$ – and usually $O(E)$ – where E is the number of previous events in the scenario; GMO evaluation is also in the efficiently-parallelizable class *NC*. Figure 7 illustrates what the flow of execution in a Gerona system looks like.

***Meta-GMOs* (MGMO)** allow a Gerona agent to answer questions from other agents about its own beliefs. MGMOs are different from GMOs in that they operate both over the

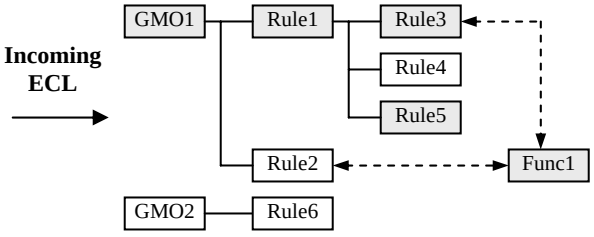

Figure 7. Execution in Gerona. An incoming ECL message triggers a GMO. Rules whose preconditions are met are shown in grey. Note that rules 1 and 2 are evaluated in parallel, as are 3, 4, and 5. Func1 is a relation, also implemented in Gerona, that is queried by rules 2 and 3.

```
GMO 6820.fire FOR ECL 6820 "Alarm in compartment "
WHERE alarm = "fire"
LET compartment -> Compartment

   RULE 6820.fire-alarm.interpret.4
      "Fire alarm triggers new crisis and goals"
   IF NOT crisis(find, unsolved, 8100, "Fire",
                    [compartment = Compartment], _, _)
   THEN
      crisis(create, unsolved, 8100, "Fire",
             [compartment <- Compartment], _, C)
      goal(create, active, 7100, "Control Fire",
            [compartment <- Compartment], C, G1)
      goal(create, …)
      …
   END RULE
END GMO
```

Figure 6. Part of a GMO, from Gerona-DCX, for handling a fire alarm. The first G-Clause determines whether a fire crisis has already been inferred for the compartment. The second and third G-Clauses create the new crisis and top-level goal (additional G-Clauses would create additional sub-goals).

CSG the GMOs themselves, creating new CSG *answer* nodes containing the results of each question. Because they can locate specific conditions that have or haven't been met, MGMOs make possible local credit assignment for knowledge differences between Gerona agents and other agents.

Examples of an MGMOs are shown in Figures 9 and 11. MGMOs may use special primitives and operations not found in GMOs. A *Meta-G-Clause* locates G-Clauses just as G-Clauses locate nodes in the CSG. The *justify* operation, when used in either G-Clauses or Meta-G-Clauses, allows the system to collect all the possible preconditions that could lead to a particular (real or hypothetical) CSG modification (this process is shown in Figure 8). A variation, called *justify-and-evaluate* uses meta-programming to determine why a particular effect did not or would not have happened. MGMOs also have special primitives that allow the context of the agent to be rolled forward and back in order to ask questions about what the agent did at some previous time. Hypothetical events may even be proposed – invoking the appropriate GMOs and creating a temporary

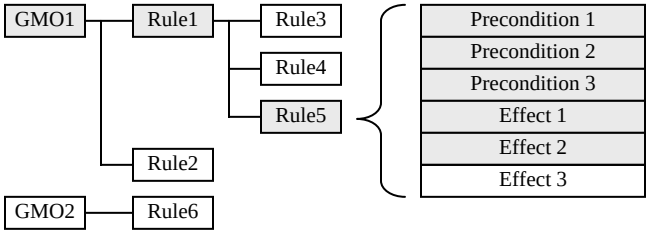

Figure 8. A graphical illustration of justification in a Gerona agent. The system is trying to justify Effect 2 in Rule 5. Shaded elements are those whose logical conditions are included in the justification.

CSG – in order to answer "what if" questions. The following are examples the very broad range of question types that can be answered by our set of about 100 MGMOs:

- Why did the agent believe a goal needed to be satisfied, propose an action, or generate a critique of another agent?
- At a particular time, what communications and beliefs did the agent generate? Why?
- Why doesn't the Gerona agent believe a particular action by another agent was correct?
- How would some hypothetical event be critiqued by the system, in some hypothetical situation? Why?
- What does the Gerona agent believe is the definition of a particular predicate or function ?
- When in general would the agent make a specific action recommendation?
- How in general does the agent believe a specific goal can be addressed or satisfied?
- When did the agent believe some condition first become true or false?

Example Interaction with a Gerona Agent

The version of Gerona that allows it to function as a watching agent for apprenticeship learning in the domain of ship damage control is called Gerona-DCX (Fried, et al, 2003). Gerona-DCX is a component of the DC-Train immersive trainer in the domain of ship damage control (Bulitko and Wilkins, 1999). In our experiments to date, Gerona-DCX has been employed as the more competent or gold-standard agent, and the active agent has been a novice problem solver.

As part of a post-session debrief, Figure 9 and 10 illustrate Gerona critiquing the active agent's action of sending the wrong repair team to fight a fire. This MGMO rolls the scenario back to just before the novice made the error, determines all the places it could modify the CSG to mark the order in question correct, and then uses *justify-and-evaluate* to determine why those operators failed to be evaluated. What is notable about MGMOs is their small size, which in some cases can generate pages of feedback concerning an action that an active agent could take or has taken.

Perhaps this answer given in Figure 10 isn't enough for the novice agent to determine what the critical knowledge difference is between it and the Gerona agent. Another possible question Gerona could ask of the novice agent is "When in general is it appropriate to order isolation?" The MGMO that generates the answer to this question is shown in Fig 11, and one answer (of several) is shown in Fig 12.

Power and Learnability

The computational power of a particular knowledge representation is of interest because simpler representations tend to be easier to learn (we learn C4.5 rules, not C++ programs). Gerona lacks an explicit control layer, but it is still expressive; it can mimic the operation of a multi-tape Turing machine to compute any function in $TISP(O(n^2),O(n))$.

```
MGMO 9002 FOR ECL 9002 "Why Sub-Optimal Action"
LET action-node -> ActionNode
RULE 9002.1 "Explain why the action isn't correct."
IF g-clause( find, action([create, modify], correct,
        ActionNode.ecl, _, _, _, _), _, CorrectGClauses)
   AND roll-back(before, ActionNode, _)
   AND g-clause(justify-and-evaluate, CorrectGClauses,
        ActionNode, Justification)
THEN
   answer(create, _, 9002, "Why Sub-Optimal Action",
      [action-node <- ActionNode,
       justification <- Justification], ActionNode, A)
END RULE
END MGMO
```

Figure 9. The entire MGMO that answers why a particular action by another agent was critiqued as sub-optimal. It uses a Meta-G-Clause to locate G-Clauses that recognize an action as correct, establishes a context just before the other agent sent the order, and then evaluates the various preconditions for the action being correct to see which one failed.

```
trigger( 5170, "Isolate Space",
      [ compartment -> Compart, target -> Station ] )
      ***SUCCESS***
goal( find, active, 7116, "Isolate Compart for Firefighting",
      [ compartment = Compart ], _, G) ***SUCCESS***
world-state(find, _, 3360, "Can Isolate Compartment",
      [compartment = Compart], _, _) ***SUCCESS***
world-state(find, _, 4302,
      "Best Repair Locker for Compartment",
      [compartment <- Compart, station = Station], _, _)
      ***FAILED***

   "In order for ordering a station to isolate a space to
   be correct, the goal of isolating the space should be
   active, the compartment should be able to be isolated, and the station should be the best repair
   locker for the compartment – you failed to meet this
   last condition."
```

Figure 10. A justification created by the *justify-and-evaluate* operation in the MGMO in Figure 9, and stored in an *answer* node in the CSG. The system is capable of creating justifications as both GMO excerpts and in English.

However, its power is also reasonably limited, as it cannot compute any function in $TIME(\omega(n^3))$ or in $SPACE(\omega(n))$.

Furthermore, Gerona rules can be shown – at least in theory – to be PAC-learnable if the ECL vocabulary is already defined. The proof of learnability stems from the "Learning to Take Action" paradigm (Khardon 1999). The proof requires that the size of learned GMO rules be bounded, which is a very reasonable assumption to make.

Related Research

A similar real-time agent that can operate in multi-agent environments is SPARK (Morley and Myers, 2004), which attempts to combine an "elegant, semantically grounded"

agent framework with mechanics that scale up to real-world problems. SPARK provides a full-featured programming environment, where Gerona's computational power is limited for the purposes of transparency and learnability. SPARK does, however, include reflective tools similar to Gerona's MGMOs that allow interesting questions to be answered about an agent's performance.

The Disciple system (Tecuci 1998) is another system that acquires knowledge through multi-agent interactions. Unlike Spark and Gerona, Disciple is not primarily designed as a real-time agent. However, it does feature a powerful mechanism for improving its rule knowledge; this mechanism involves asking questions about specific examples with which the Disciple agent is having difficulty. By using examples rather than exposing a human expert directly to rule knowledge, it is unnecessary for experts to be familiar with formal logic while helping the agent to learn.

```
MGMO 9300 FOR ECL 9300 "When action"
LET action-ecl-number -> ActionECLNumber
RULE 9300.1 "Determine when action is appropriate."
IF g-clause( find, action(create, pending,
          ActionECLNumber, _, _, _, _), _, GClauses)
   AND g-clause(justify, GClauses, _, Justifications)
THEN  answer(create, _, 9300, "When action",
          [ action-ecl-number <- ActionECLNumber,
            justifications <- Justifications ],
            miscellaneous-questions, _)
END RULE
END MGMO
```

Figure 11. The entire MGMO that determines the conditions under which an action is appropriate. The Meta-G-Clause finds operations which could create a proposal to perform the action, and the *justify* operation creates the excerpts that will serve as answers (see Figure 12).

```
trigger(6801, "damage report (fire/flood/smoke)",
       [casualty = "fire", source -> Src,
        compartment -> Compart])
goal(find, inactive, 7116, "Isolate Compartment",
       [compartment = Compart], _, G)
world-state(find, _, 3360, "Can Isolate Compartment",
       [compartment = Compart], _, _)
world-state(find, _, 4302,
       "Best Repair Locker for Compartment",
       [compartment <- Compart, station -> Station], _, _)
goal(modify, active, 7116, "Isolate Compartment",
       [compartment = Compart], _, G)
action(create, pending, 5170,
       "Electrically and Mechanically Isolate Space",
       [compartment = Compart, target = Station], G, _)

   "When a fire report is received from a station in a
   compartment, and the goal of isolating the com-
   partment (if necessary) is not active, and the com-
   partment can be isolated, and the best repair locker
   for the compartment is some station, then mark the
   goal as active, and recommend ordering the station
   to electrically and mechanically isolate the com-
   partment."
```

Figure 12. One of several possible sets of circumstances in which the action of isolating a compartment is appropriate. This GMO excerpt would be one of several generated by the MGMO in Figure 11.

Acknowledgements

We gratefully acknowledge the feedback on the Gerona design by Greg Dhuse, Eugene Grois, and Karl Schultz. This research was supported, in part, by ONR MURI grant N00014-00-1-0660.

References

Clancey, W. J. 1985. Heuristic Classification. *Artificial Intelligence*. 27:289-350.

Bulitko, V. V. and Wilkins, D. C., "Automated Instructor Assistant for Ship Damage Control," *Proceedings of the Eleventh Conference on Innovative Applications of Artificial Intelligence*, IAAI-99, Orlando, July 18-22, 1999, 778-785.

Bulitko, V. and Wilkins, D.C., "Qualitative Simulation of Temporal Concurrent Processes Using Time Interval Petri Networks," *Artificial Intelligence*, Volume 144, Issue 1-2, 95-145, March 2003

Buchanan, B. G. and Shortliffe, E. H. 1984. *Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley.

Donoho, S., 1993. Similarity Based Learning for Heuristic Classification, M.S. Thesis, Dept of Computer Science, Univ of Illinois.

Dhuse, G, 2004. Glint: An Interpreter for Graph Modification Operators of the Gerona Language for Crisis Decision Making, M.S. Thesis, Department of Computer Science, Univ. of Illinois.

Fried, D. M., Wilkins, D. C., Grois, E., Peters, S., Schultz, K. and Clark, B. 2003. "The Gerona Knowledge Ontology and Its Support for Spoken Dialogue Tutoring of Crisis Decision Making Skills," Workshop on Knowledge and Reasoning in Practical Dialogue Systems, *Eighteenth International Joint Conference on Artificial Intelligence*, Aug 2003.

Khardon, Roni. "Learning to Take Actions." *Machine Learning*, Volume 35, Number 1, pp. 57-90, 1999.

Morley, D., and Myers, K. "The SPARK Agent Framework", *Third International Conference on Autonomous Agents and Multiagent Systems*, Volume 2, 714-721.

Park, Y. T. and Donoho, S. and Wilkins, D. C., "Recursive Heuristic Classification," *International Journal of Expert Systems*, Volume 7, Number 4, 1994, 329-357.

Tecuci, G. 1998. *Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool, and case Studies.* London, England: Academic Press.

Wilkins, D. C. 1988. Knowledge base refinement using apprenticeship learning techniques. In *Proceedings of the 1998 National Conference on Artificial Intelligence.* 646-651.

Wilkins, D. C. and Ma, Y. 1994. "The Refinement of Probabilistic Rule Sets: Sociopathic Interactions," *Artificial Intelligence*, Volume 70, Number 1, 1994, 1-32.