Third Workshop on Knowledge and Reasoning in Practical Dialogue Systems, *Eighteenth International Joint Conference on Artificial Intelligence*, IJCAI-03, August 2003, 79-89.

The Gerona Knowledge Ontology and Its Support for Spoken Dialogue Tutoring of Crisis Decision Making Skills

David M. Fried, David C. Wilkins, Eugene Grois
Beckman Institute,
University of Illinois
405 N. Mathews Avenue, Urbana, IL 61801

Stanley Peters, Karl Schultz, Brady Clark
Center for the Study of Language and Information
Stanford University
210 Panama Street, Stanford, CA 94305

Abstract

This paper describes the Gerona knowledge ontology and the ways it supports spoken dialogue tutoring of crisis decision making skills. Gerona allows domain ontology models to be encoded and reasoned over. This paper describes how it has been used to encode three types of tutoring models that are relevant to simulator-based training: an expert model, a student critiquing model, and a question-answer model. These models provide SCoT-DC, a spoken conversational tutor, with domain specific knowledge needed for reflective tutoring.

1 Introduction

This paper describes the Gerona knowledge ontology and its support for spoken dialogue tutoring of crisis decision-making skills. Gerona allows three different tutor related models to be encoded that are relevant to simulator-based training: expert, student critiquing, and question-answer models [Fried and Wilkins, 2003].

It is important for an automated tutor to understand and have access to all of the information an expert might use in making a decision. A tutor needs to be able give expert explanations for decisions as well as pinpoint specific problems with a student's line of reasoning. This means being able to ask and answer questions pertaining to any of the knowledge reasoned over. To support this a domain ontology should make all domain knowledge explicitly available, and provide means for querying the ontology to answer domain specific questions and creating justifications. Gerona was designed to be a functional knowledge ontology; interpreters operating over the ontology generate expert decisions, critique student performance, and provide question/answer facilities for the creation of explanations and justifications.

The organization of this paper is as follows. Section 2 presents the proof-of-concept domain of ship damage control, and two tutoring systems for this domain. Section 3 describes the Gerona language, Section 4 shows its relevance to tutorial dialogue, and Section 5 presents future research directions and a summary of the paper.

2 Training for Crisis Decision Making

The proof-of-concept domain is whole-task training of crisis decision making for ship damage control. The damage control problem involves coordinating personal and other resources to address crises such as fire, smoke, flooding, and pipe rupture. The person being trained is called a Damage Control Assistant (DCA). It is difficult for human experts to become proficient in this task because there very few opportunities for realistic practice. It is well-known that practice is essential in order to achieve expertise. Practice is especially important in domains that involve decision making under high-stress, information overload, time-pressure, and incomplete information [Ericcson, 1993; Sniezek, et al, 2002]. Figure 1

I: Okay, you got a fire report on deck 1. What was the first thing you did?
S: I, uh, set fire and smoke boundaries.
I: Yes, and that was correct. Then what did you do?
S: I think I sent Repair 5 to fight the fire.
I: Was that the correct thing to do? Fight the fire?
S: I don't know. I guess not?
I: It wasn't. You left something out. What else did you need?
S: Uhm, if I sent Repair 5 –
I: You need to make sure mechanical isolation is complete before fighting the fire. You didn't have—you didn't order isolation.
S: So if I had gotten isolation, then I could fight the fire?
I: Yeah.

Figure 1 – An Example Damage Control Debriefing.

This is a dialogue between an instructor (I) and a student (S) about a damage control scenario involving a fire.

gives an example of a post-training session discussion between a student DCA and an instructor.

In a simulation-based training environment, a student interacts with a simulator-trainer from which reports are received about the evolving state of the world. An instructor who is providing feedback monitors these reports as well as the student's actions. The instructor may make recommendations or critiques, and may also answer student questions about specific events in the scenario or about general domain knowledge.

Such a simulator-trainer does not necessarily need to encode deep knowledge of the domain. Such knowledge is at the wrong level of detail, and it does not represent the knowledge used by human decision makers during a crisis. For instance, a firefighter does not need to know the combustion enthalpy of wood or the specific heat of water in order to fight a fire. Instead it should encode practical or "immediate" knowledge, and organize its knowledge around events. Examples of events are reports indicating changes in the simulated environment, student actions, and requests for explanation or justification of the student's behavior. Outputs are action recommendations, critiques of student performance, and the explanations and justifications requested by the user or tutor.

Explanation facilities are especially critical when tutoring. It is much more useful to know, for example, why a student action is incorrect than simply that it is incorrect.

2.1 Intelligent Tutoring Prototypes

Two research prototypes for training a DCA have been created and used with DCAs and are being refined along the lines described in this paper. The first of these is the DC-Train simulator-trainer [Bulitko and Wilkins, 1999].

DC-Train consists of a physical ship simulator, intelligent agents for simulation of ship personnel, an immersive multimedia interface for communication with a DCA, an expert model, a student critiquing model, and an interactive text-based tutor. Its input and internal representation is the Event Communication Language (ECL) described in Section 3.3 and it outputs an expert model, critiquing model, and justifications in the Causal Story Graph (CSG) formalism described in Section 3.4. The expert, critiquing, and justification models are currently encoded in C++, and are being replaced by the more explicit Gerona representation of GMOs and Meta-GMOs described in Sections 3.5 and 3.6. An executable version of the Gerona ontology yields an automated DCA expert who is referred to as the DCX. A DCX can solve damage control crises, critique other DCAs, answer questions about a crisis scenario, and provide justifications for its answers.

The second training system is SCoT-DC (Spoken Conversational Tutor[1]), which engages the student in a mixed-initiative reflective dialogue [Clark et al. 2002]. SCoT-DC is an intelligent tutor that engages DCA students in an after-action review of the student's performance during a DC-Train session. SCoT-DC includes modules for linguistic capabilities, including (i) speech recognition and speech synthesis, (ii) parsing text strings to logical forms and generating strings from logical forms according to an English grammar, (iii) context tracking and dialogue management. Other modules of SCoT-DC are (iv) a graphic interface manager and (v) a tutor. The system plans a reflective discussion of a student's performance in a DC-Train session, invites the student to interactive tutoring, and then asks a series of questions about actions during the DC-Train session, principles of damage control, and the student's answers in the reflective tutoring session. The intelligent tutor is the component of SCoT-DC requiring access to the damage control knowledge represented in Gerona. In section 4 we describe how Gerona enables a tutor to ask and respond to a wider range of questions. For example, requests for information and requests for confirmation; see [Shah *et. al* 2002] for discussion.

3 The Gerona Ontology

The Gerona Ontology facilitates the of encoding of expertise related to real-time crisis decision making. It allows the encoding of an expert model, a student critiquing model and a question-answer model for a crisis domain.

A Gerona ontology organizes its encoding of procedural doctrine around incoming and outgoing events to the DCA. These include events from the damage control simulator, the ship personnel simulator, the actions taken by the student DCA to solve a simulated crisis scenario, and questions asked by students during tutoring. The Gerona representation of procedural doctrine is both declarative and executable; it can be used directly to describe the system's expert knowledge or to justify its actions.

3.1 Motivation for the Design of Gerona

Gerona is a knowledge ontology that was specifically designed to meet the needs of a reflexive tutor that discusses a solved crises using spoken dialogue. In this section, we enumerate some unique needs of such a tutor, and describe how Gerona addresses those needs.

One important need of a spoken dialogue reflexive tutor is to have a comprehensive structured representation of everything that occurred during the solved crisis that is to be tutored. This includes all simulated crisis events, all personnel actions, all problem solving goals and actions related to each crisis and subcrisis, the expert actions that would comprise a solution to the crisis, the actions that the student took during the crisis, and a critique of each student action.. Gerona contains such a comprehensive structured representation. It is a graph structure called a Casual Story Graph and is described in Section 3.4

[1] http://www-csli.stanford.edu/semlab/muri/

Third Workshop on Knowledge and Reasoning in Practical Dialogue Systems, *Eighteenth International Joint Conference on Artificial Intelligence*, IJCAI-03, August 2003, 79-89.

A second important needs of a spoken dialogue reflexive tutor is the representation of the crisis decision making doctrine in a declarative form that can be reasoned over by a tutor. Moreover, it is helpful if there is a direct link between nodes of the CSG that contains all information about a solved crisis scenario and the specific doctrine that relates to that node. The Graph Modification Operators (GMOs) of Gerona satisfy these two important requirements and are described in Section 3.5. They are composed of rules that consist solely of a restricted clausal form called a G-Clause and these are also described in Section 3.5

Lastly, a spoken dialogue reflexive tutor needs to be able to handle a very wide range of questions. Consider student initiated dialogue, wherein a student can ask any question that comes to mind related to a 20 minute crisis scenario session that the student has just solved. Spoken dialogue by its unstructured nature provides a major challenge for a tutor. Gerona provides a mechanism for encoding a wide range of question-answer templates called Meta-GMOs. These are described in Section 3.6. To date, we examined past student-instructor dialogue and covered the range of their question asking behavior with about 100 specific question-answer templates. An example of a question answer template is as follows. "If retract [actions] in the past and assert [actions] in the past, then what is the correct [action] at a particular future time." A specific example of this as asked by the student could be as follows: "If early in the scenario I had turned firemain value 23 instead of firemain value 42, and had started firepump 3 instead of firepump 6, would I have had sufficient water pressure later on in the scenario when there was an explosion in engine room". This is an example of a question-answer template that requires time rollback of the CSG and the introduction of hypotheticals. M-GMOs allow for the encoding of such questions. The reason over the GMOs and CSG to produce the needed answer.

3.2 Related Research

There are many knowledge ontologies that encode domain-specific knowledge such as OpenCyc, Algernon, and Expect. Gerona is uniqe in terms of its scope: namely ship damage control. With respect to Intelligent Tutoring, it is has the advantage of being an executable ontology, wherein the execution yields an expert model, a student model, a question-answer model, and a justification model. And it is satisfies important needs of a spoken dialogue tutor as described in Section 2.

The Gerona ontology contains many of the basic elements found in production-based tutoring systems, such as ACT-R [Anderson and Lebiere, 1998], including a working memory and if-then rules. It differs in the specificity of these constructs. The working memory is a highly structured graph representation, and the if-then rules clauses are restricted to a clausal form called a G-Clause.

3.2 Components of a Gerona System

The major components of the Gerona Ontology are: an Event Communication Language (ECL), a Casual Story Graph (CSG), Graph Modification Operators (GMOs), Meta-Graph Modification Operators (Meta-GMOs), and Graph Clauses (G-Clauses). Their interrelationship is illustrated in Figure 2.

3.3 Event Communication Language

Consider a blackboard-style system which reasons about the DCA task using unconstrained FOPC. The system, when running, has a hodgepodge of facts, predicates, and functions in its working memory that it uses to reason. Facts may be introduced or retracted at any time, and unless conventions are introduced to restrict how they appear, the format of each could be arbitrary.

Such a system must contain code which conveys rules like "when personnel group P reports that there is a fire in compartment X, add the fact 'fire-report(P, X)' to the blackboard." Another piece of software reading the blackboard to, for example, do automated debriefing would have to know the exact format in which the FOPC system represented its information. If it instead looked for facts in the form of "report(fire, P, X)," the correct information would not be found.

The Event Communication Language (ECL) creates a standard way for enumerating and representing all types of information—events, goals, crises, facts, predicates, and functions—unique to a particular domain. The representation is, furthermore, as accessible to third-party programs as it is to the Gerona agent itself.

ECL resembles object-oriented knowledge representations; each item being represented is an instance of some class which dictates its properties. Suppose a fire report is received by the system: "Repair Locker 2 reports fire in compartment 3-300-1-Q." The ECL representation would have the class "damage report" and the properties "problem" (fire), "source" (Repair Locker 2), and "compartment" (3-300-1-Q).

For bookkeeping purposes, each ECL class is also assigned an "ECL number" that uniquely identifies it; for example, in the DCX system, a damage report is ECL number 6801. There are several hundred different ECL classes currently defined for DCX.

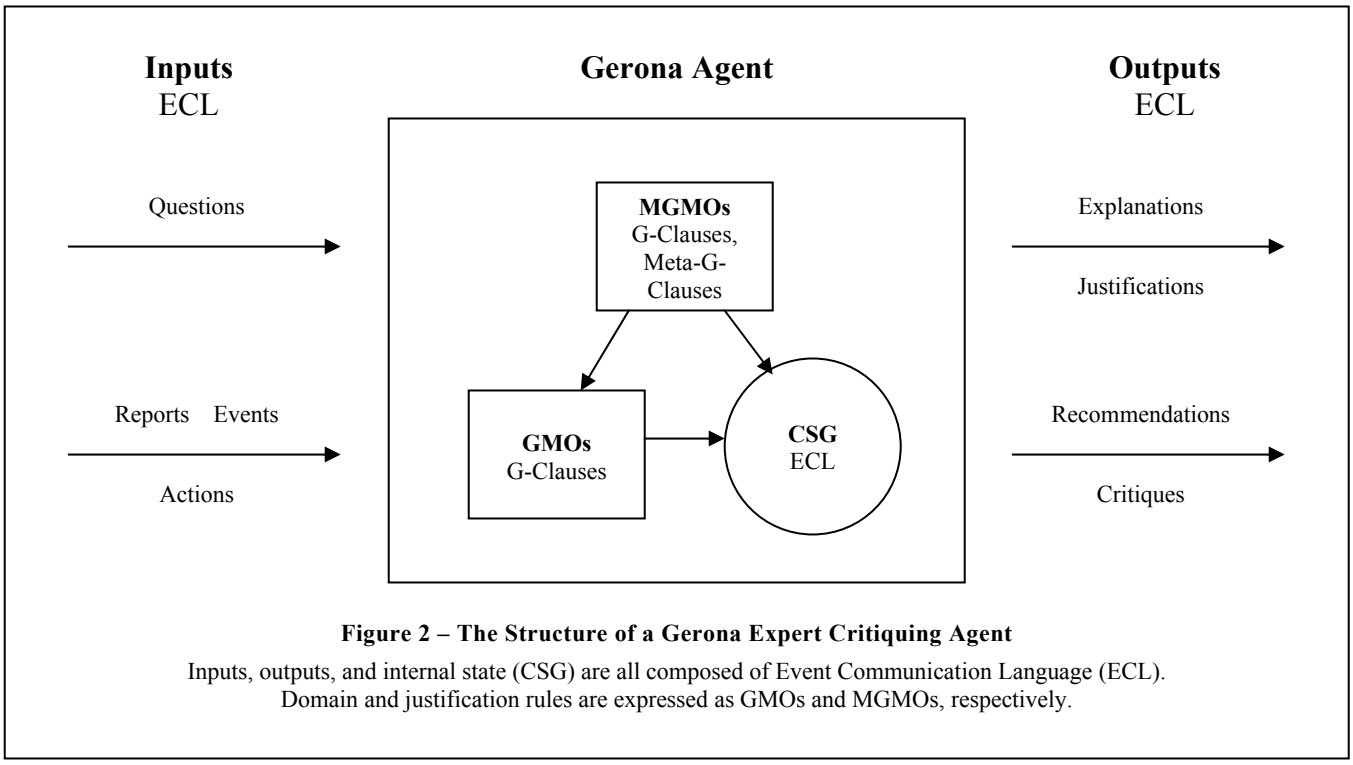

Figure 2 – The Structure of a Gerona Expert Critiquing Agent

Inputs, outputs, and internal state (CSG) are all composed of Event Communication Language (ECL). Domain and justification rules are expressed as GMOs and MGMOs, respectively.

The *type* of an ECL class describes what role it serves in the system; reports, actions, questions, and justifications are inputs and outputs, crises and goals are used to represent internal state, and world-state and world-info represent functions, predicates, and facts necessary for reasoning in the domain. Examples of various types of ECL classes are shown in Figure 3.

All that is necessary for ECLs to be accessible outside of a Gerona system is a table that lists the ECL classes and their associated ECL numbers and properties. This table can also include information such as how to translate an instance of each class into English in one or more contexts. When a tutoring program encounters a particular ECL item in the Gerona system's output, it can look up the ECL number and class and use it to determine how to express the information to the student.

3.4 Causal Story Graph

The Causal Story Graph (CSG) is a way to organize ECL instances associated with a particular scenario into a coherent world picture [Fried and Wilkins, 2000]. In production system parlance it also serves as the working memory, as no dynamic state information persists in the GMOs or Meta-GMOs. The CSG is organized as a tree, with each node being a separate, instantiated ECL item (see Figure 4). ECL items in the tree are arranged hierarchically, with crises at the top level; goal trees attached to each crisis; actions and their critiques attached to the lowest-level (most specific) goals; and reports attached to goals and actions to which they are most relevant.

Each CSG node is just an ECL instance, though some ECL types have an additional state associated with them in the CSG. For example, a node representing a goal

Example	Class	Type	Number	Properties
"Repair locker 5 reports fire in compartment 3-300-1-Q"	damage report	report	6801	source, compartment, problem
"Repair locker 3, enter space and fight fire in compartment 1-120-0-L"	fight fire	action	5120	target, compartment
Q: "Who has jurisdiction in compartment 3-300-0-E?" A: "Repair Locker 5"	jurisdiction	world-state function	4001	compartment, station
"Fire in compartment 01-128-2-A"	fire	crisis	8000	compartment

Figure 3 – ECL examples from the DCX system for the DCA training task.

The first column gives an example of information to be represented, and the following columns show what the corresponding ECL class looks like. Example instances are given as they would be spoken in English; the actual representation of ECL instances can involve as little as just the ECL number, a timestamp, and the values for each property.

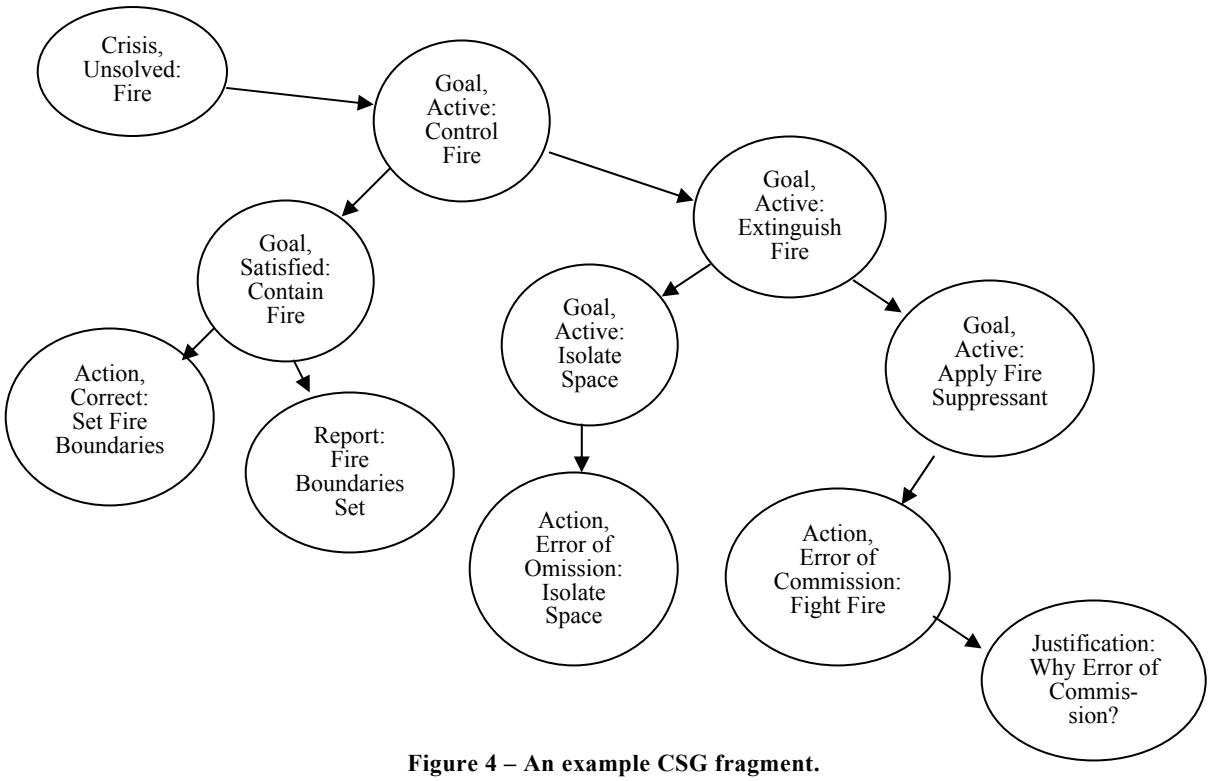

Figure 4 – An example CSG fragment.

This CSG could represent the scenario discussed in Figure 1. Note the crisis (fire), goal tree, and associated reports and actions. ECL instance properties are left out due to space constraints.

could have the state of "inactive," "active," "addressed," or "satisfied." A crisis would be "solved" or "unsolved." An action might be "pending," indicating that it is recommended by the expert system, "correct," indicating that the student performed the action in accordance with the domain rules, an "error of commission," indicating that the student ordered it but should not have, or one of several other similar states.

An example CSG fragment from the DCX system is shown in Figure 4. This fragment represents a fire crisis, with part of the associated goal hierarchy. The student has successfully contained the fire by setting fire boundaries (wetting down bulkheads to prevent fire spread). The student has also incorrectly ordered firefighting before ordering electrical and mechanical isolation—this is prohibited and results in the generation of both an error of omission and an error of commission. A justification is attached to one of the errors, indicating that during the tutoring session, clarification of exactly why the action was erroneous was requested.

The CSG can be used by programs other than the Gerona agent which creates it because it is immediately output to a file (in the case of DCX, a database) as it is created. In this way, an automated debriefing tool for tutoring has access to every recommendation and critique that the Gerona system generates. Such a program can also input a question ECL to the Gerona system and then read the answer to the question as a justification node in the CSG; this can be done offline, without the simulation running.

3.5 Graph Modification Operators

Graph Modification Operators map from events in the simulation (messages from agents or actions by the student DCA, encoded as ECL) to changes in the CSG. In this way, GMOs encode all expert and critiquing knowledge in the system.

A GMO evaluates a set of nested rules (in the form of IF-THEN statements) to determine what operations to perform on the CSG. The valid operations are creation, and modification of nodes. The conditional part of each rule reasons over non-CSG ECL items (like world-state and world-info facts, functions, and predicates) and the existence, state, and properties of ECL items in the CSG (like actions and goals).

Graph Modification Operators are written in the Gerona language, which uses a Prolog-like syntax, with variables beginning with capital letters and tokens beginning with lower-case or a number. Unlike Prolog, keywords are in all capitals and there is no backtracking; the latter difference makes evaluation of GMOs tractable.

Figure 5 shows an example of part of a GMO. The details will be explained below.

The Fundamental Syntax of Gerona: G-Clauses

Graph Modification Operators are composed of IF-THEN rules, where the premise and action are conjunctions of highly-constrained FOPC clauses called Graph-clauses or G-Clauses. The form of a G-Clause is as follows:

ecl-type(csg-operation, csg-state,
ecl-number, ecl-classname, ecl-properties,
csg-parent, gmo-variable)

To locate or verify a piece of information, the csg-operation is "find." To modify the CSG by adding a node, the operation is "create," and to change an existing node, the operation is "modify." Note that "find" can be used to check a relation, function, or existential as well as locate a node or nodes in the CSG.

To specify what type of information is to be retrieved or modified, the ecl-type is used. For example, to propose a new action, the syntax is:

action(create, pending, …)

To mark a goal as addressed, the syntax would be:

goal(modify, addressed, …)

As shown here, the third parameter (csg-state) is the state of the node in the CSG. A goal node in the CSG might be inactive, active, addressed, satisfied, or overridden. The state of an action node can be pending, correct, late, sub-optimal, late sub-optimal, error of commission, error of omission, or expired. Crises can be unsolved, solved, or overridden. Reports have no state.

State is ignored for non-CSG operations. An example of an operation where state is not used is retrieving the jurisdiction for a compartment:

world-state(find, _, 4302,
"best repair locker for compartment", …)

Again, as shown here, the next two parameters are the ECL number and class name. These are interchangeable, but the number is useful for shorthand notation and the class name for human comprehension.

It is also necessary to be able to specify new properties for a created or modified CSG node, parameters for a function or relation, or constraints for a search in the CSG or of an existential. This is done through the ecl-properties field. Here is a new node with its properties being bound:

crisis(create, unsolved, 8000, "fire",
[compartment <- Compartment], Parent, C)

A function with one input and one output (given a compartment, who is the best repair locker?):

world-state(find, _, 4302,
"best repair locker for compartment",
[compartment <- Compartment,
station -> Station], _, _)

Constraints on a search of the CSG (are boundaries set between decks 1 and 5?):

report(find, _, 6314, "boundaries set on bulk-heads", [above <= 1, below > 5], _, _)

To specify where a new node is placed, or to constrain a search in the CSG to children of a node (or nodes), use the csg-parent field

Finally, to specify a node (or nodes) to be modified, use the gmo-variable field. Also use this field to bind the results of a "find" or "create" operation on the CSG.

```
GMO 5120
FOR ECL 5120 "Fight Fire in Space"
   compartment -> Compartment
   target -> Station

   RULE 5120.fight-fire.critique.1
   IF
      goal(find, active, 7118, "Apply Fire Suppressant", [compartment = Compartment], _, G)
      AND action(find, pending, 5120, "Fight Fire in Space", [compartment = Compartment], _, A)
      AND goal(find, satisfied, 7110, "Contain Fire", [compartment = Compartment], _, _)
      AND goal(find, satisfied, 7116, "Isolate Space", [compartment = Compartment], _, _)
      AND world-state(find, _, 4302, "Best Repair Locker for Compartment",
                        [compartment <- Compartment, station = Station], _, _)
   THEN
      action(modify, correct, 5120, "Fight Fire in Space", [compartment <- Compartment, target <- Station], _, A)
      goal(modify, addressed, 7118, "Apply Fire Suppressant", [compartment <- Compartment], _, G)
   END RULE
   …
```

Figure 5 – An example GMO excerpt.

This excerpt describes the critiquing rule for determining if a student order to fight a fire is correct. Further rules in the same GMO would identify incorrect student actions, each rule corresponding to a different mistake.

Features of GMO Rules

Each GMO rule is given a unique name; in the example in Figure 5, that name is "5120.fight-fire.critique.1." Any time a node is created or modified, the name(s) of the rule or rules that fired are embedded in the node so that the CSG modification can be traced.

Only IF-THEN logic is allowed in GMO rules. Since there is no control logic (ELSE statements, WHILE loops, etc.), and since rules are order-independent, it is possible to analyze a partial excerpt of a GMO without losing any context. While this can create a small amount of redundancy in the logic of the rules, it allows for the use of GMO excerpts as a powerful justification tool, as explained in Sections 3.6 and 4.

Example GMO

To give an example of information that is encoded in a GMO, we will work from the expert rule and show how the GMO for that rule is created. Consider the following critiquing rule: "When the student orders firefighting, if the compartment has already been isolated, the fire has been contained, and the student chose the correct repair party for the compartment, then the student's action is correct." The expert has also given us a goal tree corresponding to the steps required to fight a fire; this includes the goals "Isolate Space," "Contain Fire," and "Apply Fire Suppressant."

The first part of the rule: "When the student orders firefighting," becomes the GMO header. In other words, the event that this rule is responding to is the student ordering firefighting—the only time this rule is invoked.

The next part of the rule is the precondition: "if the compartment has already been isolated, the fire has been contained, and the student chose the correct repair party for the compartment." This becomes three G-Clauses checking the satisfaction of the "Contain Fire" and "Isolate Space" goals and the value of the function "Best Repair Locker for Compartment." All three are ECL classes that have been defined for the DCA training domain—the first two are nodes in the CSG and the third is a function.

The final part of the rule is to translate "then the student action is correct." This means that an action node with a state of "correct" is created in the CSG.

The actual GMO is only slightly more complex, as shown in Figure 5. The additional lines check to see that the Gerona system has already suggested the action the student took, and verify that the student is addressing an unaddressed goal by ordering firefighting.

3.6 Meta-Graph Modification Operators

GMOs reason over the CSG in order to produce expert and critiquing behavior. Meta-GMOs reason over both the CSG and the GMOs to produce justifications of expert and critiquing behavior [Fried and Wilkins, 2003]. Also unlike GMOs, Meta-GMOs are domain-independent, so the same justification procedures can be used in any domain where Gerona is applied. Finally, Meta-GMOs can be used to answer questions about the scenario and about the domain in general that do not involve specific expert or critiquing actions.

Each Meta-GMO encodes one type of question-answer template. So far, around 100 such templates have been created and each has been given a unique question ECL class. If a question arises that cannot be handled by an existing question class, a new Meta-GMO can be hand-coded. One of the future research goals of Gerona is to determine how to learn Meta-GMOs automatically from transcripts of tutoring between student DCAs and human instructors.

Meta-GMOs are composed of G-Clauses just as GMOs are (see Figure 6). However, they can do several things that normal GMOs cannot. A Meta-GMO "sees" the entire history of the CSG instead of just the current moment. Find operations are extended to include searching after, before, or between times. The Meta-GMO can roll the scenario back or forward, looking to satisfy a particular condition, or even introduce hypothetical events into the system, firing the appropriate GMOs and making

```
MGMO 9003
FOR ECL 9003 "Why Error of Commission"
   error-of-commission-node -> EOC

   IF
      action(find, error-of-commission, ECLNum, _, _, _, EOC)
      g-clause(find, action([create, modify], correct, ECLNum, _, _, _, _), GC)
      roll-back(before-event, EOC, _)
      g-clause(justify-and-evaluate, GC, Justification)
   THEN
      justification(create, _, 9003, "Why Error of Commission",
                  [eoc-node <- EOC, justification <- Justification], EOC, _)
   END IF
END MGMO
```

Figure 6 – An example Meta-GMO.

This Meta-GMO describes the strategy for explaining an error of commission. The scenario is rolled back to when the error was committed. The correct rule is then extracted and evaluated, showing which preconditions were and were not met.

```
GMO 5120
FOR ECL 5120 "Fight Fire in Space"
   compartment -> Compartment
   target -> Station

   RULE 5120.fight-fire.critique.1
   IF
      goal(find, active, 7118, "Apply Fire Suppressant", [compartment = Compartment], _, G) = TRUE
      AND action(find, pending, 5120, "Fight Fire in Space", [compartment = Compartment], _, A) = FALSE
      AND goal(find, satisfied, 7110, "Contain Fire", [compartment = Compartment], _, _) = TRUE
      AND goal(find, satisfied, 7116, "Isolate Space", [compartment = Compartment], _, _) = FALSE
      AND world-state(find, _, 4302, "Best Repair Locker for Compartment",
                        [compartment <- Compartment, station = Station], _, _) = TRUE
   THEN
      action(modify, correct, 5120, "Fight Fire in Space", [compartment <- Compartment, target <- Station], _, A)
```

Figure 7 – A justification created by a Meta-GMO.

This is a justification for why firefighting was an incorrect action. It is an excerpt from the GMO given in Figure 5 and created by the MGMO in Figure 6 in response to a question of ECL class "Why error of commission?" The GMO header, rule precondition, and target G-Clause are extracted, and meta-programming is used to show which preconditions were met and which were not.

temporary modifications to the CSG.

Most importantly, a Meta-GMO can use Meta-G-Clauses, that search for G-Clauses in a GMO the same way G-Clauses can search for nodes in the CSG. A Meta-G-Clause looks like a G-Clause, but begins with "g-clause" as its data type:

```
g-clause(mgmo-operation, g-clause-spec,
    mgmo-variable)
```

The mgmo-operation is either find or justify (which will be described below). The g-clause-spec is a g-clause with partially-instantiated fields that will be used as a template to find matching G-Clauses. The mgmo-variable can bind G-Clauses that are found by a search, or specify which G-Clauses will be justified.

When a Meta-GMO is finished, it writes its answer into the CSG as a justification node.

Meta-GMO Justifications

The answer produced by a Meta-GMO when it responds to a question ECL has up to two parts. The first is a short answer, if one is required. For example, the question, "when were all stations manned and ready?" requires an answer like "at 3:06 of the scenario, when Repair Locker 2 reported that it was manned and ready." The second part of the answer (and the more interesting one) is an excerpt from the GMOs that justify the answer. The justification comes in one of several forms. If a justification is required for something that the Gerona system did, the excerpt contains the GMO header and the rule or rules that generated the behavior. If a general question is asked about the domain, such as "when is it appropriate to order firefighting?" then excerpts are produced from every GMO that generates the behavior.

If a justification is required for a critique of a student error, an excerpt of the correct rule is given, and meta-programming techniques are used to identify which rule preconditions were satisfied and which were not. Each precondition in the excerpt is annotated with its value (true or false) at the time the student action occurred. This is a general approach in logic meta-programming. [Hill and Gallagher, 1994; Barklund, 1995].

A justify operation can be done on a node or G-Clause. If it is done on a G-Clause, then that "target G-Clause," all rule preconditions and post-conditions preceding it in the GMO, and the GMO header are retrieved. If a node is to be justified, then the G-Clause that created the node becomes the target G-Clause and the same extraction technique is used. If meta-programming is needed, as in the example in Figure 7, then each precondition is evaluated as true or false. Justifications are created by G-Clauses and Meta-G-Clauses; the operation with meta-programming is called "justify-and-evaluate" while the operation without meta-programming is called simply "justify."

Meta-GMO and Justification Example

How might we simulate part of the instructor-student dialogue in Figure 1? A rule in the GMO in Figure 5 can classify the student's action as an error of commission. By sending the system a "Why Error of Commission?" ECL, handled by the MGMO in Figure 6, the justification in Figure 7 is produced, and placed into the justification node shown in Figure 4.

By choosing the preconditions that were not satisfied, a natural language tutoring system may translate this GMO excerpt: "When you ordered firefighting, it was judged to be incorrect because the compartment had not been isolated." (The other failed precondition corresponds to a recommendation by the system.) This is very similar to the actual instructor's comment: "You need to

make sure mechanical isolation is complete before fighting the fire … you didn't order isolation."

4 Support for Tutor Dialogue

In this section, we describe how the spoken conversational tutor for damage control (SCoT-DC) we are developing could use Gerona to structure the content of tutorial interactions. With a knowledge ontology consisting of the components presented so far, we have a number of facilities for reasoning about a domain. There are methods for reasoning like an expert, for critiquing a student's performance, and for gathering procedural explanations of doctrine based on the question-answer Meta-GMO templates. It is the last item of this list which is most interesting for dialogue, since it provides a clear interface to the domain knowledge in a way that is suitable for querying for information (e.g. "When should I fight the fire?").

The Meta-GMOs correspond directly to a large variety of Q/A templates for information about doctrine, represented as nodes in a CSG or as GMOs. Each node contains a type (e.g. goal, crisis, report), and some of them contain states as well (e.g. active, addressed, satisfied). The scope of what these nodes can represent is the entirety of the knowledge this representation gives to an artificially intelligent system using it. Unless a system is attempting to reason about a domain beyond what is specified in the ECLs and GMOs, the ECLs will provide a clear bound to the total knowledge held, and the GMOs will allow the tutor to actively reason within the realm of ECLs (using the Meta-GMOs).

Given the premise that a tutor only needs to reason about problems, states of a problem, and actions which modify a state(s) of a problem (all represented as ECLs in the CSG), the questions that can be dealt with are those concerning nodes and their states. Why a particular node has a particular state (e.g. why a goal was adopted and what action addressed it), what nodes are relevant in a given context (e.g. because they are the preconditions for taking the action prescribed for that situation), and when or how a node can change to a specific state, or be added to the graph. These relationships are embodied in the GMOs.

Since the CSGs and GMOs will contain, or can contain for any hypothetical situation, all the information necessary to make a step in reasoning, we think that any kind of question which has a procedural, step by step, answer can be represented by a Meta-GMO Q/A template. By defining the Q/A system using the same entities as the knowledge itself, the tutor obtains a direct mechanism for querying domain knowledge.

4.1 Asking questions

For a tutor to be able to pose meaningful questions it needs to have a method for identifying what kind of information it wants to elicit from the student (e.g. "After getting the report of fire, what was the first thing you should have done?"). Reasoning in terms of objects represented as nodes in a CSG supports a compact, systematic, and self-contained correspondence between the information the tutor desires and a question (Meta-GMO) that should elicit this information if the student knows it.

The key to Meta-GMOs providing a tutor with a simple mechanism to obtain the necessary domain knowledge required to ask a question and subsequently assess a student answer is that they wrap around the knowledge in a very explicit way. Staying closely tied to the notation for ECLs, CSGs, and GMOs (specifically G-Clauses), there is very little effort required to extend the language of GMOs to that of Meta-GMOs. Further, the ways in which GMOs are enabled to modify a CSG, is exactly what knowledge needs to be queried, and the Meta-GMOs are wrapped tightly around that.

Meta-GMO's seek out where a node could have been created or modified, and then hand back the G-Clause logic responsible. When relating to a student what was wrong with their action, the critiquing GMOs are important (Figure 5). A Meta-GMO will take the student action (ECL) and a problem state (CSG) and identify problematic preconditions for various action classifications to explain how an action was correct or incorrect (or suboptimal, or erroneous, etc.). It is exactly this close relationship between a node, its state in a CSG, and the domain knowledge (GMOs) that put it there which allows Meta-GMO's to easily facilitate finding and returning the necessary knowledge to answer some question about that node. Resulting from this, the tutor has a Q/A facility which, once a question is identified by a node, state, and some aspect of that node, the tutor has the question to ask, and the correct answer(s) ready to compare to the student's response.

The range of ways the tutor can phrase a question is an issue that is beyond the scope of this presentation. SCoT-DC (see Section 2) currently generates questions by filling sentence templates. Future versions may fill logical form (LF) templates for the sentence generator to map to sequences of words, or may employ more sophisticated sentence planning. Another issue is how to compare a student utterance to an answer returned by a Meta-GMO. An answer will come back as a sequence of G-Clauses embedded in one or more IF-THEN clauses to create a line of reasoning. The tutor needs a method for mapping both answers into a common structure where they can be compared. How to compare two explanations and identify whether a student answer matches the Meta-GMO answer, or whether there are important differences, could be very difficult. At the very least the comparison needs to identify unique elements in each answer. Extra elements in the student's answer could be a source of misconception and missing elements could indicate knowledge gaps. One tactic which a tutor could use is to try and make the student's explanation eventually match the

Meta-GMO explanation by going through the differences one at a time.

Given what we have discussed so far, the tutor should be able to ask questions about procedural doctrine and determine the degree of correctness of the student's response. How it responds to an incorrect answer is up to the tutorial strategies and tactics being applied, but the tutor now has a means for manipulating and dealing with answers to its questions.

4.2 Answering questions

Research about how human-to-human tutoring works has shown a positive correlation between high quality student questions and improved learning gain [Graesser 94]. Students that have practice at asking questions become better at figuring out their knowledge gaps and asking more pinpointed questions to correct themselves. This makes it important for an interactive tutor to be able to deal with student initiative in a robust way, and with Meta-GMOs the tutor can answer any question it can ask. To explain how the Meta-GMO representation can be used to answer questions is tied very closely to how it enables a system to ask questions and know the correct answer.

What we need to get out of a student utterance is: 1) what kind of ECL node(s) are they talking about; 2) is there a specific instantiated version of that node they are referring to; and 3) what are they asking about that node, or what aspect of the node is in question (e.g. its presence, its state, its relation to another node, etc.).

For example, in the sample dialogue from Figure 1 the student asks "So if I had gotten isolation, then I could fight the fire?". What we need to pull out of this is that the student is asking about isolation and firefighting, which map to two specific ECLs. They are asking about a specific firefighting node, the student's order to fight the fire (from context/earlier questioning). They want to know when it is correct to fight the fire (from "then I could…"). More specifically, they want to know how achieving isolation (from "gotten isolation") relates to firefighting and if in this situation, had the predicate for isolation hypothetically been satisfied, would their action have been correct.

Using something similar to the tutor's map into the Meta-GMO Q/A templates, a question can be picked out. This maps to a specific Meta-GMO: "Would condition [condition-ecl] be true or false if hypothetical event-sequence [event-sequence] occurred at time [time]?" Filling in the blanks: "condition-ecl" is a "correct action" classification for a firefighting action; "event-sequence" is an order to isolate the compartment (before fighting the fire) resulting in a report that "isolation has completed"; and the "time" is implicitly defined as being before firefighting was ordered.

The answer is found by running the Meta-GMO and the tutor can then use the answer to create a response. This particular Meta-GMO will take a CSG consisting only of reports and actions up to the time before the firefighting act in question was issued (based on the "time" parameter). It would then 'order' isolation, which the Gerona interpreter would simulate by activating the appropriate GMO to modify the graph. Since we are in hypothetical mode (there is no simulator to truly isolate the compartment) we assume an order-result, that the compartment was isolated. We could assume the opposite, that isolation was never achieved, but based on the student's question they are not interested in that case. The report of "isolation is complete" invokes the Gerona interpreter again. At this stage we have created the hypothetical situation suggested by the student and are now ready to see if it would be correct to fight the fire.

The Meta-GMO then 'orders' firefighting, and checks for the result. In this case the result is a "correct-action" classification for the action, and the tutor simply responds: "Yeah. [that would be correct]". How a response is created obviously depends on the current tutoring strategies and tactics, and doesn't have to be limited to turning a G-Clause answer directly into speech. Had the Meta-GMO shown us the action was still incorrect, then the tutor would perhaps figure out which preconditions were still not met, and then ask the student leading questions about the remaining issues. The algorithms for manipulating such an answer, interacting with current notions of student knowledge, and applying the correct tutoring tactics and strategies will create the scope of responses a tutor can give to a student question. What is important here is that the tutor has a viable method for understanding a student question and identifying the correct answer.

There is a major language problem with how students actually pose questions to a tutor which needs to be mentioned. The method described above works fine if the student poses a question clearly as a question. However, students often imply their questions through a subtle request for confirmation in their answer, or perhaps their refusal to answer points out they have some question they cannot even identify [Shah et al. 2002]. These are both points a human tutor would address through further questioning or perhaps hinting towards the solution. How to identify them in a student utterance, or respond with appropriate hints or questions is something we are currently addressing. It poses the larger problem that although we have the facility to respond to a very large variety of direct questions, the student may not ask those questions in a way we expect. More interpretation of a student utterance will be needed to figure out what a student wants to ask or needs to be asked when these situations arise. We hope that further study of human-to-human tutorial dialogue and collection of student utterances from our future experiments with SCoT-DC will help shed light on this issue.

5 Future Research and Summary

There are a couple of issues we plan to address with using the representation described to make a more robust Q/A system. Aside from noticing that there is quite a bit of work involved in comparing student explanations to Meta-GMO explanations, it is worth investigating what

types of answers a system will now have as well as what restrictions might be imposed.

If you think of a simple question, such as "Why should I order firefighting now?", there are two major categories of answers. The Meta-GMO representation will provide the procedural answer: "You should order firefighting to satisfy the goal of extinguishing the fire." However, the motivational answer is not available anywhere: "You should order firefighting because applying water to fire will put the fire out." This is one of the major problems inherent in representing knowledge as production rules, or sequences of IF-THEN clauses [Clancey 1987]. How to incorporate that kind of information into a knowledge ontology from which a number of uses are desired (i.e. for expert action generation and student action critiquing) is not entirely clear. Motivational knowledge would not be as useful for automated expert action generation or automated student action critiquing, but we would want it for creating justifications and appraising student explanations, since the procedural explanation cannot always suffice.

Another issue involves having a predefined list of Q/A templates that the tutor can ask questions about and answer questions for. This makes it possible for a student to ask a question and the tutor to have no possible way of answering it, as well as the tutor possibly wanting to find a piece of information but having no question with which to ask for it. The alternative to this is to have a dynamic question generation facility, and an interpreter which understands the scope of possible dynamic questions and can query a knowledge base for answers. We cannot say at this time whether one method is better than the other, intuitively it seems a dynamic system could have more capabilities, and that a predefined system puts a greater burden on the creator. Both kinds of Q/A systems, however, have a well-defined realm of knowledge determined by the underlying KR so it may turn out that both are equally as powerful, but it is the underling knowledge which restricts the Q/A capabilities. Future investigation will hopefully make this distinction more clear.

Currently, we have a working version of a spoken dialogue tutoring system; a working version of the expert-critiquing system with an associated text-based non-dialogue tutor; and a specification of ECLs, GMOs, and Meta-GMOs for the ship damage control domain [Fried and Wilkins, 2003]. The next steps are two in number. First, to implement interpreters for the Gerona-encoded expert, critiquing, and question-answering models that have been specified. And second, to base the dialogue system on the common knowledge ontology that is provided by the Gerona language.

References

[Anderson and Lebiere, 1998] Anderson, J. R., and Lebiere, C., *The Atomic Components of Thought*. 1998. Mahwah, NJ: Erlbaum.

[Barklund, 1995] J. Barklund. Metaprogramming in logic. In A. Kent and J. G. Williams (Eds.) *Encyclopedia of Computer Science and Technology*, volume 33, pp. 205-227. New York: Marcell Dekker.

[Bulitko and Wilkins, 1999] 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.

[Clancey 1987] Clancey, William J., *Knowledge-Based Tutoring: The GUIDON Program*. 1987. Cambridge: MIT Press.

[Clark *et al*., 2002] Brady Clark, Elizabeth Owen Bratt, Oliver Lemon, Stanley Peters, Heather Pon-Barry, Zack Thomsen-Gray, and Pucktada Treeratpituk. Proceedings of the International CLASS Workshop on Natural, Intelligent and Effective Interaction in Multimodal Dialogue Systems. June 28-29, 2002. Copenhagen, Denmark.

[Corbett and Anderson, 1992] Corbett, A. T. & Anderson, J. R. *Student modeling and mastery learning in a computer-based programming tutor*. In Proceedings of the Second International Conference on Intelligent Tutoring Systems. Montreal, 1992.

[Ericcson et al., 1993] Ericcson, K. A., Krampe, R. T. and Tesch-Romer, C. (1993). The role of deliberate practice in the acquisition of expert performance. *Psychological Review. 100*, (3), 1993, 363-407.

[Fried and Wilkins, 2001] Fried, D. and Wilkins, D. C., Design of Expert Session Summary Graphs for Critiquing Dialogues in DC-Train 4.0, Knowledge Systems Lab Report UIUC-BI-KBS-2001-0030. Beckman Institute, University of Illinois, Urbana-Champaign. March 2001. 29 pages.

[Fried and Wilkins, 2003] Fried, D. and Wilkins, D. C, A Knowledge Ontology that Supports Expert, Critiquing, and Tutoring Models: DCX 3.0, Knowledge Systems Lab Report UIUC-BI-KBS-2003-0001, Beckman Institute, University of Illinois, February 2003, 133 pages.

[Graesser and Person, 1994] Arthur C. Graesser and Natalie K. Person. Question Asking During Tutoring. *American Educational Research Journal*, pp. 104-137, Spring 1994, Vol. 31, No. 1.

[Hill and Gallagher, 1994] P. M. Hill and J. G. Gallagher. Meta-programming in logic programming. In D. M. Gabbay, C . J. Hogger, and J. A. Robinson (Eds.) *Handbook of Logic in Artificial Intelligence and Logic Program-*

ming - Volume 5: Logic Programming. Oxford University Press.

[Shah *et al*., 2002] Farhana Shah, Martha Evens, Joel Michael, and Allen Rovick. Classifying Student Initiative and Tutor Responses in Human Keyboard-to-Keyboard Tutoring Sessions. *Discourse Processes*, Vol. 33, No. 1, 2002.

[Sniezek et al., 2002] Sniezek, J. A., Wilkins, D. C., and Wadlington, P., Training for Crisis Decision Making: Psychological Issues and Computer-Based Solutions, *Journal of Management Information Science.* Volume 18, Number 4, 147-168, 2002.

[Wilkins and Sniezek, 2000] Wilkins, D. C., and Sniezek, J. A. (2000). The DC-SCS Supervisory Control Systems for Ship Damage Control: Volume 1- Design Overview, Knowledge Systems Lab Report UIUC-BI-KBS-2000-03. Beckman Institute, University of Illinois. December 2000, 125 pages.