Haynes, S. R., Councill, I. G., Ritter, F. E. (2004). Responsibility-driven explanation engineering for cognitive models. In AAAI Workshop on Intelligent Agent Architectures: Combining the Strengths of Software Engineering and Cognitive Systems. 46-52. Menlo Park, CA: AAAI Press.

Responsibility-Driven Explanation Engineering for Cognitive Models

Steven R. Haynes, Isaac G. Councill, Frank E. Ritter

School of Information Sciences and Technology
Penn State University
University Park, PA 16802
shaynes@ist.psu.edu, igc2@psu.edu, frank.ritter@psu.edu

Abstract

We describe an approach for developing explanation facilities for cognitive architectures based on techniques drawn from object- and aspect-oriented software engineering. We examine the use of responsibility-driven design augmented with scenario-based techniques and class-responsibility-collaboration (CRC) cards to identify explanation behaviors for cognitive model elements, and discuss the explanation benefits derived from encapsulating model behaviors within aspects. Soar is used an example cognitive architecture, but the methods and results as illustrated would apply to any of the other architectures commonly used to development psychologically plausible intelligent systems.

Introduction

We are working to create a higher-level language for Soar with the central objective of providing model developers with tools to capture explanation content at design and development time. Part of the project involves studying what kinds of explanations users of cognitive models ask of such systems (Councill et al., 2003). For explanation facility development, we are using software engineering techniques to drive specification of the explanation content and behaviors required of different model elements. We are finding these techniques have broad application to the design of many aspects of cognitive architectures and intelligent agent architectures, including human-interface and model behavior development.

Principles of object-oriented design and programming suggest a responsibility-driven perspective on the tasks of decomposition, abstraction, and encapsulation that are central to the O-O development task. Granular system components exist because of the responsibilities they assume (and presumably carry out) and not because of any structural elements or data content present in the component, which only exist to serve the responsibilities. Responsibility-driven design and development (RD3) is considered a key driver and benefit derived from the concept of encapsulation, which is paradigmatic in OOP. An RD3 perspective on system design and development is a powerful tool to help manage the complexity of large, distributed, and high-functionality systems. For these reasons and some others that have emerged from our work, we conjecture that a responsibility-driven approach is also an appropriate means to design and build effective explanation facilities for cognitive models, intelligent agents, and other knowledge-based systems.

In addition to capturing object definitions and relationships through RD3, it may be also be desirable to capture broader aspects of model implementation during the process of design and development, including model behaviors and behavior patterns. We suggest that aspect-oriented programming techniques (Kiczales et al, 1997) can be employed to supplement explanations based on model object systems. Much interesting model behavior is not readily objectifiable without the aid of a strictly plan-based modeling architecture, for example CAST (Yen et al, 2001). Behaviors such as related actions or behavior protocols (e.g. attentional or communication behaviors) may cut across a model's object system. For such behaviors explanations based on strict object-orientation will be inadequate, thus we include aspect encapsulation within our general approach to capturing knowledge for explanations during design and development.

Problem Statement

Agent and cognitive modeling architectures often rely on distributed, localized units of problem solving structure, behavior, and knowledge as the basis for higher-order functionality (Yost & Newell, 1989). This creates problems for those interested in understanding how and why these higher-order behaviors come to exist. For example, users of an agent may need to understand why an agent behaves the way that it does in order to trust its behavior. Developers may need to understand how higher-order behavior emerges in order to duplicate, enhance, or extend it. The lack of transparency in complex cognitive models means that interested parties cannot easily comprehend and act to improve the behaviors they rely upon for their models to operate effectively. This issue raises important and difficult questions about how, when emergent, higher order

behavior supervenes on the functionality of lower-level components, these behaviors can be explained. This issue raises important and difficult questions about how these behaviors can be explained when they are emergent and the higher order behavior supervenes on the functionality of lower-level components.

Planned-based approaches to explanation (e.g., Cawsey, 1993) suffer a particular brittleness in implementation because they require identifying the range of explanation requests that might be posed in model use including predicting how atomic model elements might interact in response to novel situations — often we may want intelligent systems to make use of underlying model components in new ways unforeseen at development time. Such dynamic, adaptive behavior is after all emblematic of intelligent entities. Learning and self-modification of these structures pose serious problems for this approach. Explanatory approaches that rely on the existence of explanation plans are also challenged by the complexity of keeping an evolving model synchronized with its explanation plans throughout the development and maintenance lifecycle.

Responsibility-Driven Design and Development

The essence of good object-oriented design lies in its commitment to late binding of implementation details to requirements, and an early and intermediate focus on the behaviors needed to realize the capabilities that will meet stated requirements (Wirfs-Brock & Wilkerson, 1989). One of the tenets of the object-oriented approach is its ability to help manage the complexity of large system development. Explaining the purpose, structure, and behavior of intelligent systems is correspondingly complex and requires analogous methods and tools to help expose the rationale underlying how and why they work the way that they do.

According to Wirfs-Brock and Wilkerson (1989) responsibilities of objects consist of either the *actions* they carry out or the *information* they provide. An important point is that a responsibility is not necessarily realized or implemented entirely within the object tagged with the responsibility. The object may carry out its responsibilities by delegating all or part of the realization to other objects, systems, data sources, or even humans. This suggests an approach to explanation based on functional decomposition and assignment of behaviors to atomic model elements so that dispersed explanation responsibilities can be handled locally to the greatest extent possible.

RD3 for Explanation for Explanation Capabilities

We are taking a responsibility-driven perspective on the design and development of explanation facilities for cognitive models and other intelligent systems. In the case of Soar, this means we are identifying the explanation responsibilities assigned to each of the elements a developer might define in the course of creating a useful model. Because explanation is a user and use-centric capability (even if the user is another agent or model) it makes sense to analyze model and architecture components explanation responsibilities in terms of the scenarios describing the *who*, *what*, *how*, *where*, *when*, and *why* of an explanation request scenario (Haynes, 2001).

A decomposition approach to analysis of explanation responsibilities requires answering a number of questions about the model architecture including:

1. What are the architecture and model primitives?
2. What are the explanatory responsibilities and potential of each primitive?
3. What are the benefits and costs of realizing the explanatory potential of each primitive?

The purpose of this analysis is to assign specific, realizable explanation responsibilities and corresponding capabilities to the different elements of the target architecture and model structure. The analysis results in a ranking of the explanation benefit and cost from each primitive, which is useful to help prioritize the development of explanation capabilities.

A significant challenge to this decomposition-based, responsibility-driven approach is to somehow eliminate explanation dependencies between model elements. An element's explanatory capacity should be autonomous and context sensitive, able to account for the existence of events and other objects in its environment. The challenge, therefore, is determining how the different architecture and model elements can encapsulate explanatory capabilities based only on the inputs, outputs, and internal functionality of the object. As shown in the example that follows however, resolving this issue is a challenge in a cognitive architecture such as Soar. Explaining emergent, higher-level behaviors representative of Newell's (1990) knowledge level requires augmenting the model with heuristic devices that describe the intention underlying their creation. Though in a sense these devices share the functional rational that motivate explanation plans, which as discussed earlier introduce brittleness to explanation capabilities, the approach we are taking combines the use of elemental explanation responsibilities with behavioral aspects that help link these elements to the higher-level behaviors that are designed to enact. The figure below shows how behavioral aspects overlay the PSCM. The

scenario and instantiated CRC cards that follow describe how elements and aspects interact to produce more comprehensive and context-rich explanations of the model's runtime behaviors.

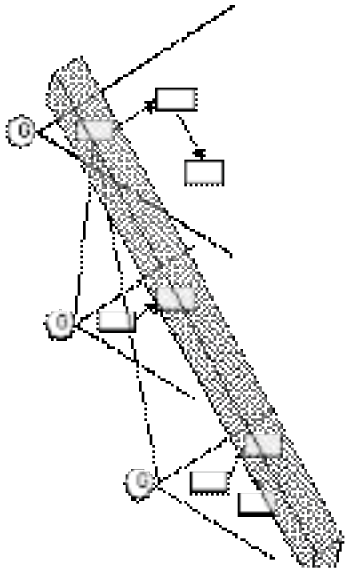

Figure 1 - Behavioral Aspect Explanation Overlay

In Figure 1 above, each *G* corresponds to a state within a goal stack, and each rectangle represents a unit action (an operator) carried out by a model. As shown, groupings of related actions (behavioral aspects) may cut across the object model of the system.

A Scenario for RD3 Explanation

We are using scenarios as the primary analysis approach to help identify the elements of the architecture and model (in this case, a Soar model) responsible for contributing to explanations of model structure and behavior. Scenario-based design techniques have been proposed as an effective input to object-oriented modeling of class responsibilities and as a technique for linking object models to user activities (Rosson, 1999). Scenarios ground responsibility analysis in concrete situations where an explanation is requested and analyzing the elements the architecture of a model that would be required to provide explanatory content. This involves drilling down into the model and through to the architecture to find where the model behavior can be explained. The purpose of this exercise is to enumerate a complete list of those elements of the architecture and model that may be relevant to explanation and then to develop a priori hypotheses regarding the most important explanatory elements.

Below we provide an example scenario showing how the approach is being applied on a Soar explanation architecture project.

A student working with D-Tank observes what she considers to be anomalous behavior in her model. The model was being pursued by an enemy tank when it suddenly executed three 90 degree turns before fleeing along an escape path. She wants to know about the decision cycles leading up the maneuver and why the model performed what seems to be an inefficient set of turns before fleeing.

Among the other explanation-seeking questions considered reasonable in situations such as this are:

What is the structure of the Soar model?
What knowledge is in the model?
How does the model represent and deviate from the real world?
What is in the model's world?
What is in the model's 'head'?
How does the model carry out behaviors to solve problems in the simulation?

The basic sequence on the problem-space level that caused this observed behavior consisted of the following:

State1 → IO Link (perceive enemy tank input) → Operator (attend) → Operator (turn) → Operator (turn) → Operator (turn) → Operator (flee)

Responsibility Analysis

Explaining how the model enacts behaviors to solve problems in the simulated world first requires tracking the flow of input-consideration-transformation-output that takes place in response to state changes and other events. Tracking, however, is only a prerequisite to then extracting detailed information about the entities involved in this flow, including how and why the entity is involved.

One of the features that would help the student understand the model and its behavior within the simulation would be to give them a 'state of the world' display. A comprehensive display with positions of all the agents in the game at the time the student was hit, along with a replay that can go back in time and play forward at different replay speeds. This display could be improved as an explanation device by allowing the user to filter irrelevant data, and could be further improved by providing annotations on the different model elements appearing on the display. VISTA (Taylor, et al., 2002), is a project moving in this direction, as well as a tool that we are using to support the creation of our environment. This trace explanation, however, only provides the information needed for the student to actively infer and construct her own explanation of why the observed behavior actually occurred.

Explaining how the model represents the world requires understanding the nature of the sense data, or perceptual input, both actual and potential and how the model

interprets this sense data. In Soar this problem is somewhat complicated by the fact that there are as many as three worlds to consider: the 'real' world, the simulated world, and the model's representation of the world. This is a critical aspect of responsibility modeling because it focuses on the design decisions made by the model developer as he or she understood the actions required from the model element in response to input/output or other state changes. It is further complicated by the need to account for other minds in the simulation, in other words, perceptions by a given model of what is inside the 'heads' of other models participating in the simulation.

Explaining the structure of the model requires identifying and describing all of the model's problem spaces, where a problem space consists of a set of operators and state representations for moving an initial state to a goal state. An impasse gives rise to a problem space where the initial state is characterized by a lack of knowledge and the goal state is achieved by acquiring the needed knowledge. However, an impasse is behavioral not structural. By definition there is no structure to provide the missing knowledge. Showing the problem space structure may therefore consist of enumerating the operators and impasses characterizing the space. The structure of the problem space is dynamic and therefore cannot be reliably determined at load time without an orienting structure to map the intended behaviors of the problem space to the underlying model elements designed to realize them.

Explaining what knowledge is in the model requires enumerating and explaining the different operators in the model and how they are organized (e.g., grouped into operator categories). Knowledge is represented as state augmentations; the model rules, the conditions under which they apply, how they are organized, and the states/operators to which they apply.

One of the issues with trace-type explanations is that it requires that the person requesting the explanation understand the meaning of the model element identifiers (names) chosen by the model developer. Also, while iterative calls to the next state or operator in the trace stack provides access to the context of the current model behavior, much of this information is model control knowledge that does not contribute to understanding system behavior relative to the domain and/or task (Clancey, 1983).

Model Elements and CRC Cards

We are using the CRC card technique (Beck and Cunningham, 1989) to capture the results of explanation responsibility scenario analysis. CRC cards are a simple tool for responsibility-driven, object-oriented design used for identifying classes (C), their responsibilities (CR), and their collaborating classes (CRC). In our case we substitute identified Soar architecture and model elements for classes, responsibilities are focused on the different types of explanatory content we might expect an element to provide, and identified collaborators remind us of the model elements that interact with the element under analysis to produce higher level model behaviors.

The Soar architecture and its models have a fairly large number of unique elements that potentially contribute to explanation of higher-order behaviors. Because of the potential cost of considering all of the elements in a given Soar model, it is important to be pragmatic at this stage focusing on those model elements considered prima facie as playing and important role in providing explanations. Using an explanation request scenario as the basis for analysis, we consider for each of these model elements the explanation responsibilities and collaborators. An initial list of these elements appears in Table 1 below. It is similar to, but larger than, the ontology included in the VISTA toolkit (Taylor, Jones, Goldstein, & Frederiksen, 2002).

Table 1 - Soar Model Elements

PSCM Level	Symbol Level
Problem Space	Rule
Decision Cycle	Elaboration Rule
Goal Stack	Operator Proposal Rule
State	Operator Application Rule
Operator	Operator Selection Rule
Operator Preference	Attribute
Impasse	Value
	Identifier
Events	Memory Structure
Operator proposal	Working Memory Element
Operator selection	IO Elements
Elaboration	Soar Commands
Operator application	Preferences
Impasse	File
Other (PSCM)	**Other (Symbol)**
IO Link (input)	Goal
IO Link (output)	Production
	Match Set

In our case we substitute identified Soar architecture and model elements for classes. Responsibilities are focused on the different types of explanatory content we might expect an element to provide. Identified collaborators remind us of the model elements that interact with the element under analysis to produce higher level model behaviors.

General concepts that have emerged from the responsibility analysis include the following responsibilities for all higher-level (i.e., PSCM) Soar components except where

variability is noted in the CRC cards below. Each component should have the ability to:

1. Identify itself
2. Enumerate its components
3. Provide a definition of itself
4. Provide its purpose as the developer understood it
5. Identify its relations and dependencies within an aspect
6. Describe how it is designed to work
7. Describe how it is designed to be used
8. Identify and describe key design constraints that impacted the component's final form

A specific example of CRC card analysis for the Soar model elements participating in the D_Tank understanding scenario is provided below (summarized somewhat for simplicity).

Table 2 - Example CRC Cards

Class	**State$_1$**
Explanation Responsibilities	Provide a layout of the current model's decision environment and representation of the world and problem-solving context Identify itself (top state) List its knowledge of the world at the time the event occurred (the working memory elements in the top state) List its behavioral aspects as the set of possible behaviors that were anticipated in the problem space (possible operators)
Explanation Collaborators	All operators that can exist within the state (e.g. turn, attend) All elaborations that can occur within the state All real and potential impasses (from superstates and to substates) Behavioral Aspects (e.g. attention, flee behavior)
Notes & Issues	States may be used to group related operators and elaborations, but this is not required. Behavior aspects may encapsulate or cut across states to provide additional grouping. Impasses provide partial information regarding the purpose of each state other than the top state.

Class	**IO Link (input)**
Explanation Responsibilities	Identifies all percepts available to the model (an enemy tank, for instance)
Explanation Collaborators	Attend operator (or Attention behavior)
Notes & Issues	An operator is required to explicitly perceive information on the input link. If an important object is not attended the oversight may be a useful component of an explanation.

Class	**Operator (attend)**
Explanation Responsibilities	Identifies the percepts from the input link that are recognized and remembered by the model, and identifies it's behavioral aspect context if applicable
Explanation Collaborators	IO Link (input) Attention behavior State$_1$
Notes & Issues	If an attend operator is not selected while there is information on the input link, the information is not perceived by the model.

Class	**Operator (turn)**
Explanation Responsibilities	Identifies the direction of the turn as well as it's behavioral aspect context if applicable
Explanation Collaborators	Preference (helps choose the Operator) Selection Rule (chooses the operator among set of acceptable ops) Elaboration (augments the Operator or the context) State (may provide some behavior context for the operator)
Notes & Issues	The Turn operator can operate within many behavioral contexts including wandering, attacking, fleeing, etc. Multi-purpose operators such as this particularly benefit from aspect encapsulation.

Class	**IO Link (output)**
Explanation Responsibilities	What actions the model performs or attempts to perform on the simulation environment
Explanation Collaborators	Action operators (e.g. turn)
Notes & Issues	An operator is required to create an action on the output link

The elements in the table suggest that it is possible for an explanation of a Soar model to be created from the components, using their explanation responsibilities and how they work with their collaborators.

Discussion

Explanations are *service responsibilities* in a model or other intelligent systems. In other words, providing an explanation is not central to the functionality desired of the agent, unless the desired functionality explicitly includes tutoring or advice giving. Service responsibilities present difficulties to cost benefit analysis because they are not easily linked to the critical development path; a model can be delivered to meet functionality requirements without

including unstated service responsibilities such as explanation or model maintainability.

Certain desirable attributes of well-engineered systems, for example, encapsulation, are only facilitated by the use of better tools, not ensured by them (Wirfs-Brock & Wilkerson, 1989). Ensuring that a design exhibits encapsulation or ensuring that a design is explainable requires a commitment to this goal on the part of model developers and their sponsors. This commitment generally obtains from recognition of the benefits accrued from achieving identified objectives. Providing intelligent systems with explanation facilities entails additional development overhead (time and costs) and so must be justified by reference to the benefits gained. These benefits potentially include longer-term time and cost savings; model quality, comprehensibility, maintainability, extendibility, and re-targetability; and enhanced model usability, broadly defined. Further research and development is required however to show that these benefits are actualized in model development and use.

Encapsulated explanation functionality is derived from the knowledge of model object inputs, outputs, and transformations, which are most clear at the time the model is being created or maintained. This suggests that populating the model element explanation content 'slots' should be encouraged or mandated as part of the model development task. Important questions remain regarding the cost-benefit ratio of efforts to include explanation facilities as a standard component in cognitive models. In particular, the costs of additional explanation knowledge engineering must be justified with respect to the usability benefits derived from their implementation. One way to help leverage these efforts for additional benefit may be to use explanation knowledge engineering to support core knowledge base analysis and development efforts. Using explanation analysis to reflect on the evolving form of the core knowledge base may play a role in helping to extend, refine, and evaluate the knowledge driving the model's behavior. After all, a model that is difficult or impossible to explain may be so because of some repairable lack of coherence rather than simply its innate complexity.

One of the objectives of this research is to identify the analytic "sweet spot" that minimizes the role of explanation plans, in form of behavioral aspects, while maximizing the contribution of atomic model elements to explanation functionality. Behavioral aspects play an important context-setting role in explaining the different runtime scenarios considered by a model developer. However, over-reliance on these plans introduces an inflexible and burdensome task into the model development and maintenance lifecycle – that of synchronizing programmed model behaviors with potential explanation capabilities.

Conclusion

Developing explanation facilities for cognitive models, intelligent agents, and other knowledge-based systems is a hard problem requiring techniques and tools to act as focusing principles for the design process. Our experiences developing explanation facilities for Soar suggest that the responsibility-driven approach using scenarios as the fundamental unit of analysis with CRC cards as the analysis product may be an effective means of meeting these challenges.

Acknowledgements

Support for this report was provided by the US Office of Naval Research, award number N00014-02-1-0021, and by the Office of Navy Research through a subcontract from Soar Technology, contract VISTA03-1. Mark Cohen and Kevin Tor also participated in discussions related to the scenario creation and CRC cards.

References

Engelmore, R., and Morgan, A. eds. 1986. *Blackboard Systems*. Reading, Mass.: Addison-Wesley.

Beck, K. and Cunningham, W. (1989). A Laboratory For Teaching Object-Oriented Thinking. In *Proceedings of OOPSLA'89*, October 1-6, 1989, New Orleans, LA.

Cawsey, A. (1993). Explanation and Interaction: The Computer Generation of Explanatory Dialogues. Cambridge, MA: MIT Press.

Clancey, W. J. (1983). The Epistemology of a Rule-Based Expert System - A Framework for Explanation. *Artificial Intelligence, 20*(3), 215-251.

Councill, I. G., Haynes, S. R., & Ritter, F. E. (2003). Explaining Soar: Analysis of existing tools and user information requirements. In F. Detje, D. Doerner, & H. Schaub (Eds.), *Proceedings of the Fifth International Conference on Cognitive Modeling*. 63-68. Bamberg, Germany: Universitats-Verlag Bamberg.

Haynes, S. R. (2001). Explanations in Information Systems: A Design Rationale Approach. Unpublished Ph.D. dissertation, Departments of Information Systems and Social Psychology, London School of Economics.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C. Lopes, C., Loingtier, J.-M. & Irwin, J. (1997). Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming*, 220--242.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33*(1), 1-64.

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

Ritter, F. E. (2003). Soar. In L. Nadel (Ed.), *Encyclopedia of cognitive science*. vol. 4, 60-65. London: Nature Publishing Group.

Rosson, M. B. (1999). Integrating Development of Task and Object Models. *Communications of the ACM,* 42(1), 49-56.

Taylor, G., Jones, R. M., Goldstein, M., & Frederiksen, R. (2002). VISTA: A generic toolkit for visualizing agent behavior. In *Proceedings of the 11th Computer Generated Forces Conference*. 29-40, 02-CGF-002. Orlando, FL: U. of Central Florida.

Wirfs-Brock R. and Wilkerson, B. (1989). Object-oriented design: a responsibility-driven approach. In *Proceedings of OOPSLA'89*, October 1-6, 1989, New Orleans, LA., pp. 71-75.

Yen, J., Yin, J., Ioerger, T. R., Miller, M. S., Xu, D., & Volz, R. A. (2001, August 2001). CAST: Collaborative Agents for Simulating Teamwork. In *Proceedings of the Joint Conference on Artificial Intelligence* (IJCAI-01), Seattle, WA.

Yost, G. and Newell, A. (1989). A Problem Space Approach to Expert System Specification. *IJCAI*, 621-627.