Integrating Different Knowledge Representations in an Intelligent System: Standardization Allows Diversity?

Ronnie W. Smith
Department of Mathematics
East Carolina University
Greenville, NC 27858, USA
rws@cs.ecu.edu

Abstract

This paper proposes that for a complex intelligent system, the choice of a knowledge representation language for the interaction processing component and the domain reasoning component should be left unconstrained provided that some type of common knowledge representation language for communicating goals, actions, and inputs is available. This common language would be used for both components for information exchange. In this way, it may be possible to design a domain-independent interaction processing component that is applicable to a genre of interactions such as advisory, database query, or task-assistance interactions.

Designer Knows Best

This paper argues for the position that it is unwise to constrain all components of an intelligent system that engages in sophisticated human-computer interaction to using the same knowledge representation language. Specifically, if we are interested in separating interaction from domain reasoning so that a standard set of interaction modules can be used with a variety of application domains, then we should not presume to constrain the internal knowledge representation used by the designer of the domain reasoning component. After all, one of the lessons that has been learned through the years in developing computational representations of algorithms and knowledge is that different representations are appropriate in different circumstances. As evidence we have the never-ending evolution of programming languages as well as a variety of programming paradigms. For knowledge representation (KR) we also have a variety of languages that are in use, and effective KR in multi-modal systems for human-computer interaction will probably require a variety of these languages.

The arguments presented here are based on personal experience in the development of a natural language dialog system known as the Circuit Fix-It Shop (Smith & Hipp 1994), a system that communicates with users via speech in order to assist a user in the repair of an electronic circuit. While only using one communication modality[1] it is still a rather complex system.[2] A major emphasis of the research has been to develop a unified architecture that integrates a variety of necessary dialog processing capabilities (problem-solving, subdialog processing, expectation usage, user-model usage, and mixed-initiative behavior). In developing this unified architecture the focus has been on general principles of task-oriented dialog rather than domain-specific aspects that would enhance system performance in circuit repair. Consequently, an important aspect of the architecture is the definition of the communication interface between the general dialog processing component and the domain-specific reasoning component as discussed below.

Interaction Processor/Domain Processor Interface

A Message-Passing Interface

In our model the process controlling the interaction (in the case of the Circuit Fix-It Shop, the dialog processing component) must control the processing done by the domain reasoner. This is done via sending messages to the domain reasoner in the form **[OpCode,Input]** where **OpCode** specifies the operation to be performed by the domain reasoner and **Input** is any additional information the domain reasoner might need (such as the description of a user input providing domain information). The list of operations that the domain reasoner should be able to perform include the following.

- Determine the next domain goal based on the domain reasoner's strategy for completing the task.
- Determine the next domain goal based on helpful information provided by the user that is not directly relevant to the domain reasoner's strategy for completing the task.
- Determine information relevant to the perceived user strategy for task completion.

[1]Keyboard input of text is allowed, but this was not used experimentally.

[2]The software includes over 17000 lines of Prolog, over 7600 lines of C, and 560 grammar rules.

- Provide a list of expected responses based on the current domain action of the user.
- Reset domain reasoner status to the values it had when an already completed action was performed.[3]
- Process user input about a domain action.

Common Interlingua: Uses and Limitations

A common language is of course needed for exchange of information about goals, actions, and inputs between the dialog processor and domain reasoner. The Goal and Action Description Language (GADL) described in (Smith & Hipp 1994) is used for this purpose. It uses a Prolog-style syntax for specifying predicates and propositions that describes goals, actions and inputs using [Object,Property,Value] triples for basic information description.

While useful as the common interlingua between the dialog processor and the domain reasoner, it is not convenient as an internal knowledge representation language for all forms of knowledge required in the two components. If natural language is used as one of the communication modalities, the interaction processor also needs to represent information about speech acts, linguistic constraints, dialog context, mental states, and user knowledge while the domain reasoner may have its own special needs. For example, in the Circuit Fix-It Shop the domain reasoner must maintain information about relative suspicion of circuit trouble spots as the potential cause of error as well as structural and behavioral descriptions of the circuit. We do not claim that the knowledge representation formalism that is appropriate for the Circuit Fix-It Shop is necessarily the most appropriate knowledge representation for other task-oriented application domains (e.g., general scheduling problems and other types of diagnostic and repair problems such as medical diagnosis).

Summary View

- There are general principles of interaction that cross domain boundaries.
- Consequently, it may be possible to design a general architecture for the interaction component in a particular interaction genre such as database query or task assistance. Of necessity, a general interaction component will specify a standard interface and knowledge representation language via which communication with the domain processing component must occur.
- However, within an interaction genre, there may be such a disparity across domains in the type of internal reasoning that is needed that it renders it impractical to require a specific knowledge representation language to be used across all domains.
- Thus, all that can be required is that the information a domain processor must exchange with the interaction processor must be represented in a common interlingua. Whether all knowledge in the domain processor is represented in this interlingua or whether the domain processor will use other knowledge representation languages internally and simply transform knowledge as needed to the interlingua should remain a designer choice.

[3]This is for cases of miscommunication when it is later determined that this action actually was *not* completed.

Acknowledgments

This work has been supported by National Science Foundation Grant IRI-9501571.

References

Smith, R., and Hipp, D. 1994. *Spoken Natural Language Dialog Systems: A Practical Approach.* New York: Oxford University Press.