HEARSAY-III:
A Domain-Independent Framework for Expert Systems

Robert Balzer
Lee Erman
Philip London
Chuck Williams

USC/Information Sciences Institute*
Marina del Rey, CA 90291

Abstract

Hearsay-III is a conceptually simple extension of the basic ideas in the Hearsay-II speech-understanding system [3]. That domain-dependent expert system was, in turn, a product of a tradition of increasingly sophisticated production-rule-based expert systems. The use of production systems to encapsulate expert knowledge in manageable and relatively independent chunks has been a strong recurrent theme in AI. These systems have steadily grown more sophisticated in their pattern-match and action languages, and in their conflict-resolution mechanisms [13]. In this paper, we describe the Hearsay-III framework, concentrating on its departures from Hearsay-II.

1. The Heritage From Hearsay-II

Hearsay-II provided two major advances -- the structuring of the workspace, called the *blackboard* in Hearsay, and the structuring of the search, via scheduling mechanisms. The blackboard provided a two-dimensional structure for incrementally building hierarchical interpretations of the utterance:

- *levels* which contained different representations (and levels of abstraction) of the domain (phones, syllables, words, phrases, etc.).
- a *location* dimension (the time within the spoken utterance) which positioned each partial interpretation within its level.

Knowledge sources (KSs), relatively large production rules, were agents which reacted to blackboard changes produced by other KSs and in turn produced new changes. The expertise was thus organized around the activity of building higher-level, more encompassing partial interpretations from several nearby lower-level partial interpretations (e.g., aggregating three contiguous syllables into a word) and producing lower-level ones from higher-level ones (e.g., predicting an adjacent word on the basis of an exisiting phrase interpretation).

Within this aggregation-based interpretation-building paradigm, Hearsay-II also provided a method for exploring alternative interpretations, i.e., handling search. Interpretations *conflicted* if they occupied the same or overlapping locations of a level; conflicting interpretations competed as *alternatives*. Thus, in addition to organizing activity around the interpretation-building process, Hearsay-II also had to allocate resources among competing interpretations. This required expertise in the form of critics and evaluators, and necessitated a more complex scheduler, which at each point chose for execution the action of one previously-matched KS.**

2. The Directions for Hearsay-III

To this heritage, we bring two notions that motivate most of our changes:

- Through simple generalization, the Hearsay approach can be made domain independent.
- Scheduling is itself so complex a task that the Hearsay blackboard-oriented knowledge-based approach is needed to build adequate schedulers.***

Our generalizations consist of systematizing the main blackboard activities:

- aggregating several interpretations at one level into a composite interpretation at a higher level,
- manipulating alternative interpretations (by creating a placeholder for an unmade decision, indicating the alternatives of that decision, and ultimately replacing the placeholder by a selected alternative), and
- criticizing proposed interpretations.

The complexity of scheduling is handled by introducing a separate, *scheduling blackboard* whose base data is the dynamically created activation records of KSs. These include both the domain-dependent KSs, which react to the regular, *domain* blackboard, and scheduling KSs, which react to changes on the scheduling blackboard as well. The organization of these activations (with agendas, priorities, etc.) is left to the application writer; Hearsay-III provides only the basic mechanisms for building expert systems. Thus domain KSs can be viewed as the legal move generators (competence knowledge) with the scheduling KSs controlling search (performance knowledge).

3. Blackboard Structure

In Hearsay-II, nodes on the blackboard, which represented partial interpretations, were called *hypotheses*. In Hearsay-III, we adopt the more neutral term *unit*. Hearsay-III provides primitives for creating units and aggregating them, i.e., associating them hierarchically. The blackboard is implemented in a general-purpose, typed, relational database system (built on top of INTERLISP), called *AP3*. AP3 has a pattern-matching language; this

*This research was supported by Defense Advanced Research Projects Agency contract DAHC15 72 C 0308. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any other person or agency connected with them.

**A good discussion of scheduling in Hearsay-II can be found in [5].

***This notion, in one form or another, is common to a number of others, for example, [6], [2], and [11].

is used for retrieval from the blackboard. AP3 also has demons; the triggering pattern which the application writer supplies as part of the definition of a KS is turned into an AP3 demon.

The blackboard levels of Hearsay-II have been generalized somewhat into a tree-structure of *classes*. Each unit is created permanently as an instance of some class. The unit is, by inheritance, also an instance of all superclasses of that class. The apex of the class tree is the general class Unit. The immediate subclasses of Unit are DomainUnit and SchedulingUnit; these classes serve to define the domain and scheduling blackboards. All other subclasses are declared by the application writer, appropriate to his domain. For example, in the SAFE application, which is a system for building formal specifications of programs from informal specifications [1], one of the subclasses of DomainUnit is ControlFragment, and it has subclasses Sequence, Parallel, Loop, Conditional, Demon, etc. The semantics of the unit classes other than Unit, DomainUnit, and SchedulingUnit are left to the application writer.

Any unit may serve to denote competing alternative interpretations. Such a unit, called a *Choice Set*, represents a choice point in the problem-solving. The Choice Set is a place-holder for the interpretation it represents; it can be dealt with as any other unit, including its incorporation as a component into higher-level units. Associated with a Choice Set unit are the *alternatives* of the choice. These may be explicit existing units or they may be implicit in a generator function associated with the Choice Set. When appropriate, a KS may execute a *Select* operation on a Choice Set, replacing it with the selected alternative. The Selection can be done in a destructive, irrevocable manner, or it can be done in a new *context*, retaining the ability to Select another alternative. Contexts are described more in Section 5.

Hearsay-II's location dimension (e.g., time-within-utterance in the speech-understanding domain) is not imposed on the Hearsay-III blackboard. The application writer may create such a dimension, either inherently in the interconnection structure of units or explicitly as values associated with the units. The flexibility of the underlying relational database system allows such constructs to have first-class status, for example, to be used in KS triggering patterns.

4. Scheduling

Hearsay-III retains Hearsay-II's basic sequencing of KS execution: When the *triggering pattern* of a KS is matched by a configuration of data on the blackboard, an *activation record* is created containing the information needed to execute the KS in the environment of the match. At some later time, the activation record may be selected and *executed*, i.e., the KS's *action*, which is arbitrary code, is run. The executing KS has available to it the blackboard data that triggered it, which usually serves as the initial focus for the activity of the execution.

Each KS execution is indivisible; it runs to completion and is not interrupted for the execution of any other KS activation. The effect of a KS execution is an updated blackboard. Indpendent activations of the same KS can pursue the same exploration by retrieving (potentially private) state information from the blackboard.

The scheduling problem is: given the current state of the system, select the appropriate activation record to execute next. The *separation of* KS execution from triggering allows for complex scheduling schemes (i.e., a large collection of activations may be available from which to select). To allow the application writer to use the Hearsay problem-solving features for building such schemes, several mechanisms were added in Hearsay-III:

- Each activation record is a unit on the scheduling blackboard. The application writer supplies, as part of the definition of each KS, code to be executed when the triggering pattern is matched; this code computes a scheduling-blackboard class (level) in which the activation record will be created.

- When executed, *scheduling KSs* are expected to make changes to the scheduling blackboard to facilitate organizing the selection of activation records. In addition to triggering on changes to the domain blackboard, scheduling KSs can trigger on changes to the scheduling blackboard, including the creation of activation records. The actions a scheduling KS may take include associating information with activation records (e.g., assigning priorities) and creating new units that represent meta-information about the domain blackboard (e.g., pointers to the current highest-rated units on the domain blackboard). The scheduling blackboard is the database for the scheduling problem.

- The application writer provides a *base scheduler* procedure that actually calls the primitive *Execute* operation for executing KS activations. We intend the base scheduler to be very simple; most of the knowledge about scheduling should be in the scheduling KSs. For example, if the scheduling KSs organize the activation records into a queue, the base scheduler need consist simply of a loop that removes the first element from the queue and calls for its execution. If the queue is ever empty, the base scheduler simply terminates, marking the end of system execution.

5. Context Mechanism

While Choice Sets provide a means for representing an unmade decision about alternative interpretations, we still need a method of investigating those alternatives independently. For that, Hearsay-III supports a context mechanism similar to those found in AI programming languages such as QA4 [10] and CONNIVER [9].

The method by which KS triggering interacts with the context mechanism allows controlled pursuit of alternative lines of reasoning. A KS triggers in the most general context (highest in the tree) in which its pattern matches. Execution of that KS occurs in the same context and, unless it explicitly switches contexts, its changes are made in that context and are inherited down toward the leaves.

Contexts are sometimes denoted as unsuitable for executing KSs -- a condition called *poisoned*. Poisoned contexts arise from the violation of a Hearsay constraint (e.g., attempting to aggregate conflicting units). In addition, a KS can explicitly poison a context if, for example, the KS discovers a violated domain constraint. A KS activation whose execution context is poisoned is placed in a wait state until the context is unpoisoned. Special KSs, called *poison handlers*, are allowed to run in poisoned contexts, and specifically serve to diagnose and correct the problems that gave rise to the poisoning.

A common application for the context mechanism arises when alternative interpretations lack good "locality". First consider the

example of SAFE's Planning Phase, which uses Choice Sets to represent alternative interpretations for control fragments. In the case of the input sentence

> "Send an acknowledgment to the imp and pass the message on to the host."

a Choice Set served well. The possible interpretations for this sentence include being put in parallel or in sequence with an existing structure; since all alternatives would be positioned identically in the existing aggregate structure, the Choice Set unit can be placed where the chosen interpretation eventually will go.

In some cases, however, locality is lacking. An example is the input sentence,

> "After receiving the message, the imp passes it to the host."

The possible interpretations for this include a demon ("The occurrance of x triggers y") and a sequence to be embedded in an existing procedure ("After x do y"). Since the demon interpretation resides at the same structural level as the procedure into which the sequence would be embedded, there is no convenient place to put the Choice Set representing these alternatives. Instead, the KSs producing these alternative interpretations put them in brother contexts, so that each can be pursued independently.

6. Relational Database

As mentioned earlier, the blackboard and all publicly accessible Hearsay-III data structures are represented in the AP3 relational database. In addition, any domain information which is to cause KS firing must also be represented in the database. This is because KSs are AP3 demons, and their triggering is controlled by activity in the database.

The AP3 database is similar to those available in languages such as PLANNER [7], but also includes strong typing for each of the relational arguments in both assertion and retrieval. These typed relational capabilities are available for modeling directly the application domain.

7. Implementation and Current Status

The Hearsay-III system is implemented in AP3, which in turn is implemented in INTERLISP [12]. AP3 was chosen as an implementation language because it already contained the mechanisms needed to support Hearsay-III (e.g., contexts, demons and constraints, and strong typing). In fact, the design of Hearsay-III's initial implementation was almost trivial, being largely a set of AP3 usage conventions. However, efficiency considerations have forced a substantial implementation effort.

Hearsay-III has been tested on two small applications: a cryptarithmatic problem and a cryptogram decoding problem. Three major implementation efforts are currently underway. The first of these, as described above, is the reimplementation of the SAFE system [1]. Second, Hearsay is being used as the basis for a system for producing natural language descriptions of expert system data structures [8]. Finally, the system is being used as the basis for a "jitterer" which automatically transforms a program so that a transformation chosen by a user is applicable [4].

The Hearsay-III architecture seems to be a helpful one. The separation of competence knowledge from performance knowledge helps in rapidly formulating the expert knowledge required for a solution. Preliminary experience with the larger applications now under development seem to bear this out, and seem to indicate that performance (scheduling) is a difficult issue. The flexibility that the Hearsay-III architecture gives toward developing scheduling algorithms will undoubtably go a long way toward simplifying this aspect of the overall problem-solving process.

Acknowledgments

We wish to thank Jeff Barnett, Mark Fox, and Bill Mann for their contributions to the Hearsay-III design. Neil Goldman has provided excellent and responsive support of the AP3 relational database system. Steve Fickas, Neil Goldman, Bill Mann, Jim Moore, and Dave Wile have served as helpful and patient initial users of the Hearsay-III system.

References

1. Balzer, R., N. Goldman, and D. Wile, "Informality in Program Specifications," *IEEE Trans. Software Eng.* SE-4, (2), March 1978.

2. Davis, R., *Meta-Rules: Reasoning About Control*, MIT AI Laboratory, AI Memo 576, March 1980.

3. Erman, L. D., F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys* 12, (2), June 1980. (To appear)

4. Fickas, S., "Automatic Goal-Directed Program Transformation," in *1st National Artificial Intelligence Conf.*, Palo Alto, CA, August 1980. (submitted)

5. Hayes-Roth, F., and V. R. Lesser, "Focus of Attention in the Hearsay-II System," in *Proc. 5th International Joint Conference on Artificial Intelligence*, pp. 27-35, Cambridge, MA, 1977.

6. Hayes-Roth, B., and F. Hayes-Roth, *Cognitive Processes in Planning*, The Rand Corporation, Technical Report R-2366-ONR, 1979.

7. Hewitt, C. E., *Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot*, MIT AI Laboratory, Technical Report TR-258, 1972.

8. Mann, W. C., and J. A. Moore, *Computer as Author -- Results and Prospects*, USC/Information Sciences Institute, Technical Report RR-79-82, 1979.

9. McDermott, D., and G. J. Sussman, *The CONNIVER Reference Manual*, MIT AI Laboratory, Memo 259a, 1974.

10. Rulifson, J. F., R. J. Waldinger, and J. A. Derksen, "A Language for Writing Problem-Solving Programs," in *IFIP 71*, pp. 201-205, North-Holland, Amsterdam, 1972.

11. Stefik, M., *Planning with Constraints*, Ph.D. thesis, Stanford University, Computer Science Department, January 1980.

12. Teitelman, W., *Interlisp Reference Manual*, Xerox Palo Alto Research Center, 1978.

13. Waterman, D. A., and F. Hayes-Roth, *Pattern-Directed Inference Systems*, Academic Press, New York, 1978.