

Research in Progress

Artificial Intelligence Research at the Information Sciences Institute

William Mann

*University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291*

FOUNDED IN 1972 to develop and disseminate new ideas in computer science, the Information Sciences Institute [ISI] is an off-campus research center of the University of Southern California, with a combined research and support staff of over one hundred. The Institute engages in a broad set of research and application-oriented projects in the computer sciences. These projects range from basic research efforts, through development of prototype systems, to operation of a major Arpanet computer facility.

The Institute's AI research focuses on program synthesis, user interfaces, programming environments, natural language, and expert systems. AI researchers are supported by ten personal LISP workstations, several VAXs, two TOPS-20 systems, and a magnificent view of Marina del Rey.

Transformation Based Maintenance and Mappings Projects

Software specification and development continue to present an enormous problem to everyone involved with computers. We believe that the computer itself must play a far more significant role in the software development process than it does presently. The software designer's role should be streamlined to require only decision making and guidance, while the computer's is expanded to manipulation, analysis, and documentation. The key to such support is to capture

in the machine all crucial information about the processes of specification, design, implementation, and maintenance.

For several years we have been developing an alternative software paradigm based on this tenet (Balzer 1983). We envision a future user developing a high-level specification of what he or she wants a program to do, and transforming the specification into an efficient program, using a catalog of (proven) correctness-preserving transformations. Most debugging and all maintenance will be performed on the specification, which will have an operational interpretation suitable for testing.

Our specification language, called GIST, formalizes some of the constructs commonly used in natural language specifications, including descriptive reference ("the message from Bill"), nondeterminism ("a message from another site"), historical reference ("the last message the user looked at"), constraints ("never send multiple copies of a message to the same person"), and demons ("if a week passes without a reply to a request, inform the sender"). A GIST specification describes the behavior of both the program and its environment; this provides a natural way to specify embedded programs that interact in complicated ways with their environment. GIST's expressive power makes it possible to specify *what* behavior a program should exhibit without saying *how* it should be implemented.

We are also developing tools to support the specification *process*, i.e., to help bridge the gap between the specifier's

informal intentions and their formal expression in GIST. Unfortunately, specifications in any formal language tend to be difficult to understand. Hence, we have developed programs that explain GIST specifications in English (see Specification Validation project, immediately following).

The *Mappings* project is discovering transformations for translating GIST's high-level constructs into the lower-level constructs used in more conventional programming languages (London & Feather 1982), as well as criteria for selecting among alternative transformations. The user will guide this process by stating a high-level goal for the system to achieve, selecting a transformation from a catalog for the system to apply, or manually applying a transformation that the catalog lacks. We have developed problem-solvers that help find a sequence of transformations that achieves a given goal (Fickas 1982, Mostow 1983).

The *Transformation Based Maintenance* project is developing the system support needed to facilitate the automated implementation of specifications using transformations. To this end we have developed the system called POPART — Producer of Parsers and Related Tools — which produces a parser, lexical analyzer, pattern matcher, editor, pretty printer, attribute grammar mechanism, and even a transformation system for any context-free language input to it. Through the use of a *formal development structure*, which captures all the design decisions, assumptions, and optimization steps, optimizations can be replayed on a changed specification to produce a new implementation automatically (Wile 1983). That is, the new implementation is developed by analogy with the old. We believe that such an automated maintenance facility will be an important application of analogical reasoning techniques.

Personnel: Robert Balzer, Martin Feather, Jack Mostow, David Wile.

References

- Fickas, S. (1982) *Automating the transformation development of software* Doctoral dissertation, University of California, Irvine
London, P. E., & Feather, M. S. (1982) Implementing specification freedoms. *Science of Computer Programming*, 2, 91-131.
Mostow, J. (1983) A problem-solver for making advice operational. *AAAI-83*, Washington, D. C., August, 279-283
Wile, D. S. (1983) Program developments: Formal explanations of implementations. *CACM*, 26 (11).

Specification Validation

Regardless of the specification language used, formal program specifications can be difficult to understand. Yet, because a specification is frequently the means by which a customer communicates his desires to a programmer, it is critical that both the customer and programmer be able to examine and comprehend the specification. Two of the

major impediments to understandability are the unfamiliar syntactic constructs of the language and non-obvious interactions between widely separated parts of the specification. These interactions may cause the specification to denote behaviors that were unintended by the original specifier or not to denote behaviors that *were* intended. The Specification Validation project seeks to overcome these impediments by constructing tools to make specifications more understandable, both to specifiers and to those unfamiliar with formal specification languages.

One tool, the GIST paraphraser (Swartout 1982), addresses the syntax problem by directly translating a GIST specification into English. We have found the paraphraser to be useful in both clarifying specifications and revealing specification errors.

The paraphraser deals only with the static aspects of a specification. Our second tool addresses the more difficult problem of making non-local specification interactions apparent by simulating the dynamic behavior implied by the specification. Our approach has been to discover non-local interactions by using a symbolic evaluator (Cohen 1983) to analyze a specification. The symbolic evaluator gathers and integrates constraints from the specification. It discovers what sorts of behaviors the specification allows, and what is prohibited. Since a symbolic evaluator does not require specific inputs, it is possible to test a specification symbolically over a range of inputs that would require many test runs if specific inputs were employed.

The symbolic evaluator produces an execution trace, which details everything discovered about the specification during evaluation and which contains mechanically produced proofs justifying the evaluator's discoveries. Unfortunately, the trace is much too detailed and low-level to be readily understood by most people. To overcome that difficulty, we have constructed a trace explainer (Swartout 1983) that selects from the trace those aspects believed to be interesting or surprising to the user and uses that information to produce an English summary. The trace explainer employs a number of heuristics about what the user is likely to want to see to summarize the trace. In addition, since the mechanically produced proofs are often very detailed and appear unnatural, the explainer reformulates the proofs into a more understandable form. The use of these heuristics has resulted in quite readable output from the trace explainer.

Personnel: Bill Swartout, Don Cohen.

References

- Cohen, D. (1983) Symbolic execution of the GIST specification language. *IJCAI-83*, Karlsruhe, Germany.
Swartout, W. (1982) GIST English generator. *AAAI-82*, Pittsburgh, PA., 404-409
Swartout, W. (1983) The GIST behavior explainer. *AAAI-83*, Washington, D. C., August, 402-407

Consul and CUE

CONSUL and CUE comprise a single research approach to the problem of producing a natural user interface to computer services. We are focusing on two basic problems:

1. translating the user's description of what he wants to do into descriptions of what the system can do (and translating back the other way for explanation);
2. insuring that the functions and data structures of the individual services can be combined to perform the tasks that the user describes.

We believe that users may think of interactive computer services very differently than the builders of these services. A system that provides a natural interface must take these differences into account, making sure that the user's expressions of his service needs can be understood and, wherever possible, actually performed by the service functions that have been implemented. Similarly, when the user requires explanation of service requirements and capabilities, the system must be able to determine what the user is asking about in system terms and then provide him with an appropriate answer in user terms.

Our approach is to build a knowledge-based system consisting of detailed models of what users want to do and what the system can do. The knowledge base, implemented in KL-ONE, is used both by components that map between user and system descriptions and by those that acquire individual service function and data descriptions into the integrated system view.

The CONSUL project is responsible for fundamental knowledge representation and inference research, and for a natural language interface that allows the user to describe what he wants to do in English. CUE (for Consistent Underlying Environment) is responsible for integrating service functionality and for a form/menu/command interface that allows the user to describe what he wants to do in a more formal framework. Both projects cooperate to build the knowledge base.

The result of this research will be a single interface through which the user sees a completely uniform service environment for performing his tasks. The interface will allow the user to describe what he wants to do in terms of forms, menus, commands, or natural language—in whatever combination he deems appropriate for the task at hand.

We have implemented a demonstration system that exhibits this kind of interface in the realm of electronic office environments consisting of interactive services like electronic mail and personal calendar. Users interact with a workstation running CUE; commands and menu selections are interpreted and executed within the CUE machine. Natural language requests and any input that requires help or explanation is sent off to a separate CONSUL server machine for interpretation. The system currently works only on a limited number of test cases, and provides response times that are suitable for demonstration but not for real use.

Personnel: William Mark, Thomas Kaczmarek, Thomas Lipkis, Robert Fenchel, Norman Sondheimer, David Wilczynski.

References

- Kaczmarek, T., Mark, W., & Sondheimer, N. (1983) The Consul/CUE interface: An integrated interactive environment. *CHI-83*
- Kaczmarek, T., Mark, W., & Wilczynski, D. (1983) The CUE project. *Proceedings of SoftFair, July.*
- Mark, W. (1981) Representation and inference in the Consul system. *IJCAI 7*
- Schmolze, J., & Lipkis, T. (1983) Classification in the KL-ONE knowledge representation system *IJCAI 8.*
- Wilczynski, D. (1981) Knowledge acquisition in the Consul system. *IJCAI 7.*

Information Management

The *Information Management* project is attempting to build a computing environment which significantly reduces the effort required to create, integrate, and evolve services, and permits users to customize these services without detailed knowledge of how they were specified. The basis for such improvements lies in raising the level of specification and modification of these services.

We have identified and provided several recurring mechanisms that form the basis for the higher level specification and evolution dictions. The most important of these are descriptive reference, and coordination and automation rules. However, since the main activity in any computing environment is building and manipulating objects, we start with a comprehensive set of lower-level object definition, instantiation, and modification operations, and a database in which the mutable objects defined and manipulated by these operations persist. This systematizes session continuity and avoids having to use files of characters to communicate between services (the services coexist and share the database). Contexts provide a mechanism for hypothetical activity and the comparison of alternative futures.

Within this database, particular objects must be located. *Descriptive reference* (associative retrieval) has been provided as a *universal access mechanism*. This avoids having to create limited and ad-hoc access structures.

The "physics" of this world is defined by coordination and automation rules. These specify, respectively, the static data consistency and dynamic causal interactions in that world. The *coordination rules* define the *consistency among objects* that must exist within a single state. They also define how that data consistency can be restored if and when it is violated. For a subset of such rules, called Constraint Equations, the required restorations can be automatically computed from the consistency conditions. These rules allow the system to take responsibility for the maintenance of data consistency. This encourages the use of multiple representations (alternative views), which increases

comprehensibility. These rules facilitate evolution by allowing modifications to be expressed in terms of the most appropriate representation (view), and by ensuring that the effects are automatically propagated, unobserved and in the same state transition, throughout all related structures (*a la VISICALC*)

Automation rules, on the other hand, *define the effects* (i.e. observable events) *that are to result from a cause*. They specify the dynamic interactions that initiate state change, and provide a means for describing regularized interactions in situation/response form. This enables the system to perform these responses on the user's behalf. Such augmentation is an important form of evolution. But altering existing processing is also required. By specifying these alterations in terms of the observable behavior of the objects, rather than the code that generates that behavior, the vocabulary of the domain becomes a higher level diction for change (analogous to the adaptability of rule-based expert systems).

These mechanisms have been incorporated into a demonstration system. However, to actually validate these ideas, a system embodying them must be used for real work over an extended period. Hence, we are constructing a testbed version which contains all key services covering the vast majority of our work. These include program development and such administrative services as mail, document preparation, and personal databases. Our intent is to use these IM services to the exclusion of the corresponding external services and to evolve them as our needs change and we recognize new opportunities for integration.

Personnel: Bob Balzer, Dave Dyer, Neil Goldman, Matt Morgenstern, Bob Neches.

References

- Balzer, R., Dyer, D., Morgenstern, M., & Neches, R. (1983) Specification-based computing environments *AAAI 83*, Washington, D. C., August, 12-16
- Morgenstern, M. (in press) Active databases as a paradigm for enhanced computing environments. *Proceedings of 9th International Very Large Data Base Conference*, October
- Neches, R., Balzer, R., Dyer, D., Goldman, N., & Morgenstern, M. (in press) Information management: A specification-oriented, rule-based approach to friendly computing environments. *Proceedings of the IEEE Conference on Systems, Man, & Cybernetics*, January, 1984

Text Generation

This project is developing a theoretical framework and implementation to represent the relatively domain-independent parts of the skill of generating fluent multiparagraph English text. The text generation module, PENMAN, is designed to acquire information intended for a reader, organize it into a text plan with details down to the independent clause level, generate sentences according to the plan,

criticize the resulting text, and revise the text plan accordingly.

The project has concentrated on two areas: sentence generation and text planning.

A large systemic grammar of English, called NIGEL, is PENMAN's sentence generator. NIGEL is designed to be compatible with different knowledge representations, so that it can become a portable English grammar for several domains and computing environments. Michael Halliday, the founder of systemic linguistics, is participating in the definition of the grammar. Systemic notation has been augmented with a semantic notation which expresses the meaning of each grammatical feature in terms of the conditions under which it is chosen. These definitions of the grammatical features are embodied in "choice experts" which control sentence generation. Definitions currently exist for over half of the features of NIGEL's grammar of English.

A new technical description of how text is organized has been developed as part of the work on text planning, and verified on natural texts. Part of this description shows that texts express a large amount of their communication load in the relations which make them coherent, rather than explicitly in their clauses. This description is currently being elaborated into a text planning method.

These two emphases, representing respectively the execution and creation of plans for text, address the most critical needs for new technical knowledge for generating fluent multiparagraph natural language text.

Personnel: William Mann, Christian Matthiessen, Sandra Thompson, Michael Halliday.

References

- Mann, W. C. (1983) *An Overview of the PENMAN Text Generation System* (RR-83-114) Marina Del Rey, CA: USC Information Sciences Institute
- Mann, W. C. (1983) Inquiry semantics: A functional semantics of natural language grammar *Proceedings of the First Annual Conference: Association for Computational Linguistics*, European Chapter, September.
- Mann, W. C., & Matthiessen, C. M. I. M. (1983) NIGEL: A Systemic Grammar for Text Generation (RR-83-105) Marina Del Rey, CA: USC Information Sciences Institute (To appear in R. Freedle (Ed.) *Systemic perspectives on discourse: Selected theoretical papers from the 9th international systemic workshop*)

Explainable Expert Systems

The Explainable Expert Systems project is just getting underway at ISI. Its goal is to create a framework for building expert systems that will ease their maintenance and evolution and enhance their explanatory capabilities. Using current frameworks, it is difficult to create, modify, or extend expert systems; the explanations and justifications of

behavior that such systems can provide (if any) are often difficult to understand. In part, these problems stem from the fact that current frameworks force a system builder to integrate mentally many different kinds of knowledge, such as domain facts, heuristics, and control knowledge, and then to encode that knowledge at a uniform (and low) level of abstraction. Because the integration process occurs mentally, it is unrecorded. Yet it represents the "reasoning behind the system" and is exactly the knowledge required to justify and explain the system's behavior - both to users and to system builders. Further, this confounding of knowledge hinders maintenance and evolution.

We are constructing a framework for building expert systems that supports their evolution and explanation by encouraging the separation and explicit representation of the various kinds of knowledge that go into such systems and recording the integration of that knowledge for use in explanation and justification. This framework will extend the paradigm used in the XPLAIN system (Swartout 1983a). There will be four major aspects to our expert-system-building system:

1. a *knowledge representation framework* to explicitly represent the various kinds of knowledge that go into an expert system (e.g. domain facts, heuristics, control knowledge)
2. an *interpreter* to dynamically integrate the knowledge and produce expert behavior

3. a *compiler* to enhance the system's efficiency by statically integrating the knowledge that does not depend on the particulars of an individual session, and
4. an *explanation facility* that will use traces recorded by the interpreter and compiler to describe and justify the system's behavior.

There will be two major research areas. The first involves the relationship between compilation and interpretation of knowledge. The XPLAIN system essentially compiled all the knowledge it used. While it does seem that experts compile their knowledge as they become more expert, it seems they do not always use compiled knowledge; indeed, particularly difficult cases may require eschewing compiled methods and interpreting "deep" knowledge. The second research area will concentrate on how to represent other kinds of knowledge (such as efficiency knowledge) in the knowledge base.

Personnel: Bill Swartout.

References

- Swartout, W. (1983a) XPLAIN: A system for creating and explaining expert consulting systems *Artificial Intelligence*, 3, 21, 295-325.
- Swartout, W. (1983b) Explainable expert systems. IEEE: *Proceedings of MEDCOMP83*.

(Correction to the Research in Progress Section in the AI Magazine, Vol. 4 No. 4, Winter, 1983)

Jet Propulsion Laboratory
by
Steven Vere

The Winter 1983 issues of AI magazine contains a notice on AI research at JPL in which I am listed as a member of the Automated Problem Solving Group. My affiliation with the group ended in October, 1982, and I have formed a new "AI Research Group" at JPL.

NOTICE TO ALL MEMBERS

Recently, it has come to our attention that individual(s) have been misrepresenting themselves as AAI staff members in order to gain access to confidential information about personnel histories and salaries and corporate organizational structures. It is not the AAI's practice to want or need such information.

We do not know who these people are or what their intentions may be. So, please be advised if such individuals contact you, please note their names, addresses, and phone numbers and confirm the intent of their call with the AAI office (415-328-3123).

Thank you.