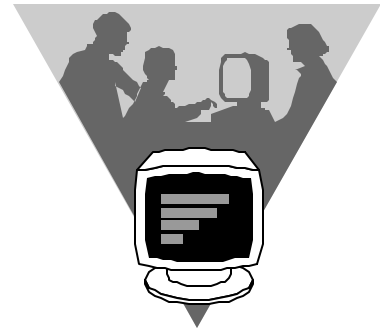


Knowledge-Based Expert Systems

Cyber Object - Expert Systems

Cyber Object specializes in Knowledge-based expert system solutions for the telephony industry. This approach dramatically increases an applications functionality while keeping it user friendly. Until recently, the expert system was implemented on host-based applications. Cyber Object has implemented the powerful CLIPS Expert System into an Internet JAVA client that allows for its use over a distributed network. Cyber Object has incorporated this proven Object Oriented Expert System in the Intelligent Customer Advocate System - ICAS.



Knowledge Based Expert Systems

Knowledge-based expert systems, or simply expert systems, use human knowledge to solve problems that normally would require human intelligence. These expert systems represent the expertise knowledge as data or rules within the computer. These rules and data can be called upon when needed to solve problems. Books and manuals have a tremendous amount of knowledge but a human has to read and interpret the knowledge for it to be used. Conventional computer programs perform tasks using conventional decision-making logic -- containing little knowledge other than the basic algorithm for solving that specific problem and the necessary boundary conditions. This program knowledge is often embedded as part of the programming code, so that as the knowledge changes, the program has to be changed and then rebuilt. Knowledge-based systems collect the small fragments of human know-how into a knowledge base that is used to reason through a problem, using the knowledge that is appropriate. A different problem, within the domain of the knowledge base, can be solved using the same program without reprogramming. The ability of these systems to explain the reasoning process through back-traces and to handle levels of confidence and uncertainty provides an additional feature that conventional programming doesn't handle.

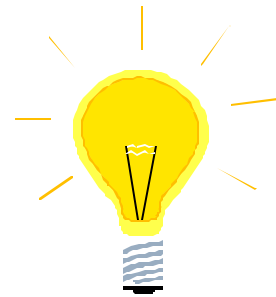
Most expert systems are developed via specialized software tools called shells. These shells come equipped with an inference mechanism (backward chaining, forward chaining, or both), and require knowledge to be entered according to a specified format (all of which might lead some to categorize OPS5 as a shell). They typically come with a number of other features, such as tools for writing hypertext, for constructing friendly user interfaces, for manipulating lists, strings, and objects, and for interfacing with external programs and databases. These shells qualify as

languages, although certainly with a narrower range of application than most programming languages. For more detailed information on expert system shells, see the "Expert System Shells at Work" series by Schmuller PC AI, (1991, 1992).

What are Expert Systems?

Conventional programming languages, such as FORTRAN and C, are designed and optimized for the procedural manipulation of data (such as numbers and arrays). Humans, however, often solve complex problems using very abstract, symbolic approaches that are not well suited for implementation in conventional languages. Although abstract information can be modeled in these languages, considerable programming effort is required to transform the information to a format usable with procedural programming paradigms.

One of the results of research in the area of artificial intelligence has been the development of techniques that allow the modeling of information at higher levels of abstraction. These techniques are embodied in languages or tools that allow programs to be built that closely resemble human logic in their implementation and are therefore easier to develop and maintain. These programs, which emulate human expertise in well-defined problem domains, are called expert systems. The availability of expert system tools, such as **CLIPS**, has greatly reduced the effort and cost involved in developing an expert system.



Rule-based programming is one of the most commonly used techniques for developing expert systems. In this programming paradigm, rules are used to represent heuristics, or "rules of thumb," which specify a set of actions to be performed for a given situation. A rule is composed of an if portion and a then portion. The if portion of a rule is a series of patterns which specify the facts (or data) which cause the rule to be applicable. The process of matching facts to patterns is called pattern matching. The expert system tool provides a mechanism, called the inference engine, which automatically matches facts against patterns and determines which rules are applicable. The if portion of a rule can actually be thought of as the whenever portion of a rule since pattern matching always occurs whenever changes are made to facts. The then portion of a rule is the set of actions to be executed when the rule is applicable. The actions of applicable rules are executed when the inference engine is instructed to begin execution. The inference engine selects a rule and then the actions of the selected rule are executed (which may affect the list of applicable rules by adding or removing facts). The inference engine then selects another rule and executes its actions. This process continues until no applicable rules remain.

What is CLIPS?

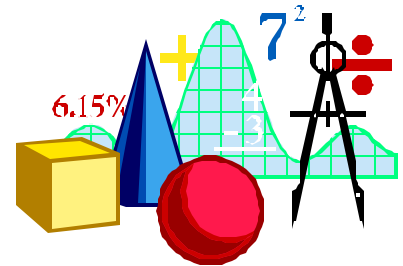
CLIPS is a productive development and delivery expert system tool that provides a complete environment for the construction of rule and/or object based expert systems. CLIPS is being used by over 5,000 users throughout the public and private community including: all NASA sites and branches of the military, numerous federal bureaus, government contractors, universities, and many companies. The key features of CLIPS are:

- **Knowledge Representation:** CLIPS provides a cohesive tool for handling a wide variety of knowledge with support for three different programming paradigms: rule-based, object-oriented and procedural. Rule-based programming allows knowledge to be represented as heuristics, or "rules of thumb," which specify a set of actions to be performed for a given situation. Object-oriented programming allows complex systems to be modeled as modular components (which can be easily reused to model other systems or to create new components). The procedural programming capabilities provided by CLIPS are similar to capabilities found in languages such as C, Pascal, Ada, and LISP.
- **Portability:** CLIPS is written in C for portability and speed and has been installed on many different computers without code changes. Computers on which CLIPS has been tested include IBM PC compatibles, Macintosh, VAX 11/780, and Sun 3/260. CLIPS can be ported to any system that has an ANSI compliant C compiler. CLIPS comes with all source code that can be modified or tailored to meet a user's specific needs.
- **Integration/Extensibility:** CLIPS can be embedded within procedural code, called as a subroutine, and integrated with languages such as C, FORTRAN and ADA. CLIPS can be easily extended by a user through the use of several well-defined protocols.
- **Interactive Development:** The standard version of CLIPS provides an interactive, text oriented development environment, including debugging aids, on-line help, and an integrated editor. Interfaces providing features such as pull down menus, integrated editors, and multiple windows have been developed for the Macintosh, Windows 3.1, and X Window environments.
- **Verification/Validation:** CLIPS includes a number of features to support the verification and validation of expert systems including support for modular design and partitioning of a knowledge base, static and dynamic constraint checking of slot values and function arguments, and semantic analysis of rule patterns to determine if inconsistencies could prevent a rule from firing or generate an error.

- **Fully Documented:** CLIPS comes with extensive documentation including a Reference Manual and a User's Guide.

The History of CLIPS

The origins of the C Language Integrated Production System (CLIPS) date back to 1984 at NASA's Johnson Space Center. At this time, the Artificial Intelligence Section (later the Software Technology Branch, Client/Server Systems Branch, and now the Information Technology Office) had developed over a dozen prototype expert systems applications using state-of-the-art hardware and software. However, despite extensive demonstrations of the potential of expert systems, few of these applications were put into regular use. This failure to provide expert systems technology within NASA's operational computing constraints could largely be traced to the use of LISP as the base language for nearly all expert system software tools at that time. In particular, three problems hindered the use of LISP based expert system tools within NASA: the low availability of LISP on a wide variety of conventional computers, the high cost of state-of-the-art LISP tools and hardware, and the poor integration of LISP with other languages (making embedded applications difficult).



The Artificial Intelligence Section felt that the use of a conventional language, such as C, would eliminate most of these problems, and initially looked to the expert system tool vendors to provide an expert system tool written using a conventional language. Although a number of tool vendors started converting their tools to run in C, the cost of each tool was still very high, most were restricted to a small variety of computers, and the projected availability times were discouraging. To meet all of its needs in a timely and cost effective manner, it became evident that the Artificial Intelligence Section would have to develop its own C based expert system tool.

The prototype version of CLIPS was developed in the spring of 1985 in a little over two months. Particular attention was given to making the tool compatible with expert systems under development at that time by the Artificial Intelligence Section. Thus, the syntax of CLIPS was made to very closely resemble the syntax of a subset of the ART expert system tool developed by Inference Corporation. Although originally modelled from ART, CLIPS was developed entirely without assistance from Inference or access to the ART source code.

The original intent of the prototype was to gain useful insight and knowledge about the construction of expert system tools and to lay the groundwork for the construction of a fully usable tool. The CLIPS prototype had numerous shortcomings, however, it demonstrated the feasibility of the project concept. After additional development, it

became apparent that sufficient enhancements to the prototype would produce a low cost expert system tool that would be ideal for the purposes of training. Another year of development and internal use went into CLIPS improving its portability, performance, and functionality. A reference manual and user's guide were written during this time. The first release of CLIPS to groups outside of NASA, version 3.0, occurred in the summer of 1986.

Further enhancements transformed CLIPS from a training tool into a tool useful for the development and delivery of expert systems as well. Versions 4.0 and 4.1 of CLIPS, released respectively in the summer and fall of 1987, featured greatly improved performance, external language integration, and delivery capabilities. Version 4.2 of CLIPS, released in the summer of 1988, was a complete rewrite of CLIPS for code modularity. Also included with this release was an architecture manual providing a detailed description of the CLIPS software architecture and a utility program for aiding in the verification and validation of rule-based programs. Version 4.3 of CLIPS, released in the summer of 1989, added still more functionality.

Originally, the primary representation methodology in CLIPS was a forward chaining rule language based on the Rete algorithm (hence the Production System part of the CLIPS acronym). Version 5.0 of CLIPS, released in the spring of 1991, introduced two new programming paradigms: procedural programming (as found in languages such as C and Ada;) and object-oriented programming (as found in languages such as the Common Lisp Object System and Smalltalk). The object-oriented programming language provided within CLIPS is called the **CLIPS Object-Oriented Language** (COOL). Version 5.1 of CLIPS, released in the fall of 1991, was primarily a software maintenance upgrade required to support the newly developed and/or enhanced X Window, MS-DOS, and Macintosh interfaces. Version 6.0, released in the spring of 1993, added fully integrated object/rule pattern matching and support features for rule-based software engineering.

Because of its portability, extensibility, capabilities, and low-cost, CLIPS has received widespread acceptance throughout the government, industry, and academia. The development of CLIPS has helped to improve the ability to deliver expert system technology throughout the public and private sectors for a wide range of applications and diverse computing environments. CLIPS is being used by over 5,000 users throughout the public and private community including: all NASA sites and branches of the military, numerous federal bureaus, government contractors, universities, and many private companies.

Acknowledgment

As with any large project, CLIPS is the result of the efforts of numerous people. The primary contributors have been: Robert Savely, previous branch chief of the STB and now chief scientist of advanced software technology at JSC, who conceived the project and provided overall direction and

support; Chris Culbert, current chief of the Information Technology Office, who managed the project, wrote the original CLIPS Reference Manual, and designed the original version of CRSV; Gary Riley, who designed and developed the rule-based portion of CLIPS, coauthored the CLIPS Reference Manual and CLIPS Architecture Manual, and developed the Macintosh interface for CLIPS; Brian Donnell, who designed and developed the CLIPS Object Oriented Language (COOL), coauthored the CLIPS Reference Manual and CLIPS Architecture Manual, and developed the previous MS-DOS interfaces for CLIPS; Bebe Ly, who was responsible for maintenance and enhancements to CRSV and is now responsible for developing the X Window interface for CLIPS; Chris Ortiz, who developed the Windows 3.1 interface for CLIPS; Dr. Joseph Giarratano of the University of Houston-Clear Lake, who wrote the CLIPS User's Guide; and Frank Lopez, who wrote the original prototype version of CLIPS.

Many other individuals contributed to the design, development, review, and general support of CLIPS, including: Jack Aldridge, Carla Armstrong, Paul Baffes, Ann Baker, Stephen Baudendistel, Les Berke, Tom Blinn, Marlon Boarnet, Dan Bochsler, Bob Brown, Barry Cameron, Tim Cleghorn, Major Paul Condit, Major Steve Cross, Andy Cunningham, Dan Danley, Mark Engelberg, Kirt Fields, Ken Freeman, Kevin Greiner, Ervin Grice, Sharon Hecht, Patti Herrick, Mark Hoffman, Grace Hua, Gordon Johnson, Phillip Johnston, Sam Juliano, Ed Lineberry, Bowen Loftin, Linda Martin, Daniel McCoy, Terry McGregor, Becky McGuire, Scott Meadows, C. J. Melebeck, Paul Mitchell, Steve Mueller, Cynthia Rathjen, Eric Raymond, Reza Razavipour, Marsha Renals, Monica Rua, Tim Saito, Gregg Swietek, Eric Taylor, James Villarreal, Lui Wang, Bob Way, Jim Wescott, Charlie Wheeler, and Wes White.