CLIPS IMPLEMENTATION OF THE AUTOMATED DOCUMENT DISTRIBUTION SYSTEM

Shamboul Mohamed and Ralph Martinez

Computer Engineering Research Laboratory,
Electrical and Computer Engineering Department,
The University of Arizona
Tucson, AZ 85721

ABSTRACT

The processing and distribution of documents in a large corporation requires a set of routing procedures for each type of document. Some of these documents may require managerial review and signature release authority to leave the organization. The document must be routed through the different levels of the organization according to the document procedures. The availability of the signers and reviewers becomes a delay factor in the routing of the document. This paper describes an approach to a solution to this problem using artificial intelligence and expert system concepts coupled with distributed computer networking to distribute the documents. A prototype system has been demonstrated in the Computer Engineering Research Laboratory, University of Arizona.

1.0 INTRODUCTION

Document distribution in an office information system for a large corporate environment has taken many forms in the past. A corporation may have several hundred types of official and unofficial documents which must be distributed at all organizational levels. Official documents may contain internal sensitive material or material for distribution outside the corporation. Unofficial documents may include routine memorandums and electronic mail between employees. The official documents usually will require technical and managerial review before they are released to their destination. Thus, corporation managers at various levels of the corporation must provide review and signature release authority to the document. There are set procedures and protocols which govern the distribution of the documents. The distribution protocols are integrally related to the organizational structure of the corporation. A document distribution system must comply both with the distribution protocols and the organizational structure. This paper addresses the automation of document distribution using artificial intelligence and expert system concepts coupled with computer networking techniques.

Document distribution systems have been researched and developed in several organizations. Two recent examples are summarized here. The first example comes from IBM. The Document Interchange Architecture (DIA) is composed of workstations and document distribution nodes (DDNs) [Shick, 1982]. The workstations are processors that serve either application processes or serve as source or recipient nodes. The DDNs are systems which store and forward the documents between the workstations. The DIA defines a document interchange unit as the basic message communications structure between DIA entities. The structure consists of a prefix header, a command sequence, data and document units, and a suffix header. The command sequence defines the functions to be performed when DIUs are exchanged between DIA applications. The DIA supports several DDNs each with its set of workstations. The DIA is an asynchronous information distribution system. The DIA does not contain provisions for signature release authority.

Another example is the Message Transport Architecture (MTA) developed by AT&T [AT&T, 1986]. MTA is designed to provide application and presentation layer support for the exchange of information across unified message services. It is composed of two distinct architectures, namely:

1. Message Service Architecture (MSA)
2. Content Description Architecture (CDA)

MSA is a hierarchical structure, which comprises universal message transport header, universal message report header, and message transport services. CDA is used to describe and manipulate the contents of the message. It consists of the Universal content description header, content description services, described contents. MTA is basically a centralized system which does not deal with signature release authority.

These two examples represent systems which use distributed processing to route the documents to their destinations. Neither system handles the signature release authority requirement used by most corporations [Gardner, 1987].

The Automated Document Distribution System (ADDS) has a well defined hierarchical structured architecture. Figure 1 depicts the document interchange hierarchical structure. It has more information on the header about stops history and number of stops to be visited. This is due to the fact that (ADDS) does more processing to its message or document, such as:

a. Obtaining signature release authority
b. Checking availability of reviewer or signer

c. Setting expiration times for reviewers or signers' absence.
d. Providing co-signer's name and address.

In ADDS, a document is originated as an "electronic file" on a user workstation (US), called the Writer. The document is processed by an inference engine in the US which also appends the list of Signers and Reviewers. The document is then sent to a Knowledge Base Server (KBS) which adds additional information regarding the distribution of the document. Each document contains headers for the communications network in the organization, distribution control header, and the document text body. The KBS stores the document according to the user profiles in the organizations. Activity of reviewing and signing the documents is originated at the user US. The document is retrieved from the KBS, reviewed by the user, signed and returned to the KBS for intermediate storage. When the KBS has determined that the document has all the required signatures (Signwords), the document is sent to the final destination. In ADDS the knowledge is distributed, each workstation has a local knowledge base in addition to the system main knowledge base server.

2.0 AI IMPLEMENTATION OF ADDS

In the first phase of the project we have used conventional programming approach (using the C-language) to implement some of the AI concepts used. In the second project phase, which is the subject of this paper, we are using AI tools and expert systems approach to implement the design. We have decided on implementing an MS-DOS based interactive expert system shell on each user workstation, and a UNIX-based real-time or callable expert system or an AI-tool on the knowledge base server. A prototype ADDS environment is shown in Figure 2. The overall architecture for the ADDS system is depicted in Figure 3.

Applying the evaluation criteria developed for this purpose, but putting more weight to the cost factor, we have settled on using CLIPS. CLIPS is an expert system language having LISP-like syntax which uses parentheses as delimiters. Although CLIPS is not written in LISP, the style of LISP has influenced the development of CLIPS. CLIPS stands for the C Language Integrated Production System (CLIPS) [CLIPS, 1987].

We have embedded CLIPS as an interactive expert system in each workstation. Our model includes ten workstations, two workstations for each organizational level, for the assumed five level organization model used the purpose of this project. The overall architecture of the ADDS distribution system is configured as shown in Figure 4. For the Knowledge Base Server, we are using CLIPS as a callable expert system embedded in a main C program. In this paper we describe the reasons for choosing CLIPS, its basic features, and the implementation of ADDS' workstations and knowledge base server using the CLIPS expert system language.

3.0 USING OF CLIPS EXPERT SYSTEM LANGUAGE

3.1 REASONS FOR CHOOSING CLIPS

There are two sets of reasons for choosing CLIPS for this particular application: distinguishing characteristics of CLIPS as an expert system language, and the general advantages of CLIPS as an expert system.

3.1.1 DISTINGUISHING CHARACTERISTICS OF CLIPS

CLIPS can run on both MS-DOS and UNIX operating systems. Thus it is suited for the PC-based workstations and the Unix-based knowledge base server. It is an expert system tool written in and fully integrated with the C language : the 'C' Language Integrated Production System (CLIPS). It was specifically designed to provide high portability, low cost, and easy integration with external systems. CLIPS provides reasonable performance on a wide variety of computers. It also includes tools for debugging an application.

Some problems have been encountered in delivering LISP-based or PROLOG-based expert systems, namely: low availability of LISP or PROLOG on conventional computers; high cost of state-of-the-art LISP and PROLOG tools and hardware; and poor integration of LISP and PROLOG with other languages , making embedded applications difficult. CLIPS has been developed to solve these problems. CLIPS is developed in C and is most easily combined with user functions written in C. However, other languages can be used for user-defined functions, and CLIPS may even be embedded within a program written in another language.

CLIPS is data driven program , where the facts are the data that stimulate execution. This is one example of how CLIPS is different from procedural languages such as Pascal, Ada, BASIC, FORTRAN and C. In procedural languages, execution can proceed without facts. That is, the statements are sufficient in those languages to cause execution. However, in CLIPS, facts are required to cause the execution of rules and so play an essential role.

CLIPS is a rule based language based on the Rete algorithm. The Rete algorithm was specifically designed to provide very efficient pattern matching. CLIPS has attempted to implement this algorithm in a manner that combines efficient performance with powerful features. When used properly, CLIPS can provide very reasonable performance even on microcomputers.

The major difference between a rule-based system like CLIPS and procedural languages such as Ada, Pascal or C is that the rules of CLIPS can be activated in parallel while the statements of other languages are sequential in nature. That is, multiple rules can be put on the agenda. In procedural languages like Ada, Pascal and C, execution normally proceeds line by line in sequence. In CLIPS, any of the rules can be potentially executed if the patterns are matched.
by facts and the rule is activated.

In CLIPS, the interpreter automatically matches rules against facts, even before your program starts executing. As soon as a fact is asserted, CLIPS starts checking for matches against rules. In other languages such as FORTRAN, Pascal, Ada and C, nothing happens until you issue a run command. But in CLIPS and LISP, the interpreter is always running. CLIPS also provides a more direct way of control through salience, which is a feature used to set the priority of rules.

3.2 CLIPS Features and Overview

The C Language Integrated Production System (CLIPS) was developed by the Artificial Intelligence Section (AIS) of the Mission Planning and Analysis Division (MPAD) at NASA/Johnson Space Center (JSC). CLIPS is a type of computer language designed for writing applications called Expert Systems. The basic elements of CLIPS are:

1. fact-list: global memory for data
2. knowledge-base: contains all the rules
3. inference engine: controls overall execution

A program written in CLIPS consists of rules and facts. The inference engine decides which rules should be executed. The primary representation methodology is a forward chaining rule language based on the Rete algorithm. CLIPS provides reasonable performance on a wide variety of computers. It also includes tools for debugging an application. CLIPS is in use at most NASA centers, numerous universities, and many places in private industry.

RULES

The primary method of representing knowledge in CLIPS is a rule. A rule is a collection of conditions and the actions to be taken if the conditions are met. The developer of an expert system defines the rules which describe how to solve a problem. The entire set of rules in an expert system is called a knowledge base. CLIPS provides the mechanism (the inference engine) which attempts to match the rules to the current state of the system and applies the actions. The current state is represented by a list of facts.

FACTS

Facts are the basic form of data in a CLIPS system. Each fact represents a piece of information which has been placed in the current list of facts, called the fact-list. Rules execute (or fire) based on the existence or non-existence of facts. A fact is constructed of several fields separated by spaces. Any number of fields may be stored in a fact, and the number of facts in the fact-list is limited only by the amount of memory in the computer. Facts may be asserted into the fact-list prior to starting execution and may be added (asserted) or removed (retracted) as the action of a rule firing. If a fact is asserted into the fact-list that exactly matches an already existent fact, the new fact will be ignored.

Each field within a fact can be one of three things: a number, a word, or a string. The first fact in the fact-list is asserted automatically by the system during a reset. This fact (initial-fact) can be treated like any other fact and may be matched or retracted.

INTEGRATION WITH OTHER LANGUAGES

When using an expert system, two kinds of integration are important: embedding CLIPS in other systems, and calling external functions from the RHS or LHS of a CLIPS rule. CLIPS was designed to allow both kinds of integration. Using CLIPS as an embedded application allows the easy integration of CLIPS with existing systems. This is useful in cases where the expert system is a small part of a larger task or needs to share data with other functions. In these situations, CLIPS can be called as a subroutine and information may be passed both to and from CLIPS. Embedded applications are discussed in the Advanced Programming Guide.
to an IF THEN statement in a procedural language such as Ada or Pascal. An IF THEN rule can be expressed in a pseudocode statement as follows.

IF certain conditions are true
THEN execute the following actions

As an example of pseudocode format let’s see how it translates to CLIPS syntax

IF document is formal
THEN signature is needed

The CLIPS format will be

(defrule formal-document
(document-is formal)
=>
(assert (signature-is needed))

The spacing of rules on multiple lines and indentation is done to make it easier to read. The entire rule must be surrounded by parentheses. Each of the patterns and actions of the rule must be surrounded by parentheses. A rule may have multiple patterns and actions. The parentheses surrounding patterns and actions must be properly balanced if they are nested. In the formal-document rule, there is one pattern and one action. If all the patterns of a rule match facts, the rule is activated and put on the agenda. The agenda is a collection of activated rules. There may be zero or more rules in the agenda.

The last part of a rule is the list of actions that will be executed when the rule fires. In our example, the one action is to assert the fact (signature-is needed). The term fires means that CLIPS has selected a certain rule for execution from the agenda. A program will cease execution when there are no rules on the agenda. When there are multiple rules on the agenda, CLIPS automatically determines which is the appropriate rule to fire. CLIPS orders the rules on the agenda in terms of increasing priority and fires the rule with the highest priority, called salience.

The part of the rule before the arrow is called the left-hand side (LHS) and the part after the arrow is called the right-hand side (RHS). The design of CLIPS is such that rules only see facts that have been entered after the rules.

CLIPS was programmed with a characteristic of a nerve cell called a neuron. After a neuron transmits a nerve impulse (fires), no amount of stimulation will make it fire for some time. This phenomenon is called refraction and is very important in expert systems. Without refraction, expert systems would be always caught in trivial loops. Only rules that have been fired experience refraction, not rules on the agenda.

5.0 CLIPS IMPLEMENTATION OF ADDS

In our application the knowledge is distributed between the workstations local knowledge bases and the knowledge base of the knowledge base server. Each workstation knowledge base is composed of a facts-list and a rule base. The facts-list represents the workstation data base. It includes workstation description information, designated stand-in signers list, access information for communicating with the KBS, and it may contain the user's absence schedule. The rule base contains the rules needed for intelligent processing of the document.

The knowledge base of the KBS consists of a database of information about the organization workstations that are connected to that KBS, a set of facts that act as a legend to the organizational data bases, plus the sets of rules that comprise the rule base. The rule base of the KBS is designed in such a modular way as to perform the different functions of the KBS intelligent processing.

The knowledge bases of ten users' workstations plus the knowledge base of the KBS have been coded using CLIPS expert system language. The ten workstations are to represent our model of the five level structured organization. Two workstations are implemented at each level, so that each user can have a cosigner at the same level as his. At level 1 we have WS-10 and WS-11, similarly, WS-20 and WS-21, WS-30 and WS-31, WS-40 and WS-41, and WS-50 and WS-51 at levels 2,3,4,5 respectively.

The communications subnet work used in this implementation is the same, Sytek LocalNet 20 broadband LAN. CLIPS has been modified to pass variables to and from the Communications Software Modules, which are written in the C language, so as to make document file transfer between the workstations and the KBS possible.

5.1 USER'S WORKSTATION IMPLEMENTATION

CLIPS has been embedded in each of our ten workstations model as an interactive expert system. It has also been integrated with other modules of the ADDS project. Some user-defined functions are needed to embed the automatic updating of the document, enacted by CLIPS; into the document structure. CLIPS was modified to give the ADDS prompt, to make it more user-friendly for workstation user. Due to all these modifications, CLIPS has to be recompiled to effect these changes. We have coded the knowledge bases of ten workstations using CLIPS expert system language, with two workstations at each level of our five level organization structure model. Figure 5 shows a typical AI-based workstation.

To demonstrate the application of CLIPS expert system language for performing the WS processing functions and establishing its local data base let us consider the complete knowledge base, both the rule base and the facts-list of a user's workstation. Let us assume that the workstation's user is a Branch head at organization A, level 4, who is using a workstation that is connected to the ADDS system. A document generated by the Branch head needs three designated signatories, Division head,
Department head, and the President to be released. Thus the WS local facts-list, in CLIPS implementation, will include this designated stand-in signers list, workstation description information, and mail queues path names to interface with the KBS. The rule base will include the rules needed to establish the relations governing the document processing procedures.

One way of coding the knowledge base for WS-40, whose user is a Branch head will be as follows:

```
;************* WORKSTATION-40 KNOWLEDGE BASE *************
;RULE BASE AND FACTS LIST USING CLIPS EXPERT SYSTEM
;LANGUAGE. CLIPS IS THE C LANGUAGE INTEGRATED PRODUCTION
;SYSTEM, WHICH HAS A LISP-LIKE SYNTAX THAT USES
;PARENTHESES AS DELIMITERS.

(deffacts workstation-40
  (user-name Wilcox)
  (level 4)
  (ws-position Br-Hd)
  (Div-Hd Tao)
  (Dept-Hd Archi)
  (cosigner Aaron)
  (workstation-40 address 400)
  (KBS-mailqueue "\\usr\\kbs\\mail\\queue")
  (user-mailqueue "\\usr\\kbs-users\\ws-40\\mail\\queue")
)

(defrule make-initial-fact
  (initial-fact)
  (system "cls")
  (assert (workstation-40 system up user available use adds))
)

(defrule purpose
  ?workstation-40 <- (workstation-40 system up user available use adds)
  (system "cls")
  (assert (user is writer))
  (assert (document - (read))))
```

The first part of this knowledge base consists of a Deffacts construct which contains the facts-list that acts like a data base or information storage. It contains WS-40 description information, such as the user-name of the WS’s user, his level in the organization, and his job position (or title). It also includes the designatory signers-list, the WS user's cosigner name, the WS address, and mail queues path names for interface with the KBS.

The second part is the rule base which contains a set of (5) rules out of the (22) rules that are needed for the WS part of the document processing. The first rule is a start-up rule which contains the initial-fact as the only pattern or condition. The initial-fact will always be asserted after a reset command, which acts as a start-up command for the ADDS system. The rule base provides a menu-driven interactive user interface which allows the user to pose questions, answer questions, choose from provided options, query the system reasoning, and assist in obtaining solutions. Rule 2 gives the system main menu with five options to choose from, cd for create document, rm for receive mail, sa for set absence schedule, sd for send document, and os for go to the operating system. The rest of the rules which are self explanatory provide the necessary document processing through interacting with the WS user.

```
5.2 KNOWLEDGE BASE SERVER IMPLEMENTATION

CLIPS has been embedded as a callale expert system in a main C program, so that CLIPS solutions for the document processing problems can be done under program control, as there will be no human in the loop to interact with the system. Thus it will act as a real-time expert system to perform document processing in real-time, accepting real-time data inputs generated by the WSs as documents sent through the communications interface, and providing real-time data outputs as
```
processed documents deposited at WSs users' mail subdirectories for workstations users retrieval. The KBS architecture is shown in Figure 6.

The KBS mail directory, the WSs users' mail subdirectories, temporary document storage files system, and permanent document archives will be the same, except that they will encounter minor adjustments to pass variables to and from CLIPS for compatible interface.

The knowledge base of KBS includes ten facts-lists: one for each WS connected to the KBS in our ten WSs model, and a user-key facts-list which acts as a legend by indexing organizational data bases (or workstations facts-lists) by user names for easy reference by CLIPS. It also includes a modular structured rule base, needed for performing the modular functions of the KBS.

The workstations' facts-lists are the organizational data bases in this implementation. Each WS facts-list contains workstation description information, such as the WS's user name, his signature, his level and position in the organization, his cosigner name, the workstation address, the WS's user mail queue path name, and the WS's user document delivery expiration time. The user's signatures are only kept at the KBS facts-lists for protection purposes.

The rule base is structured and modular to fulfill the modular nature of the KBS processing functions. The rules included will formulate the solutions to document processing and distribution functions carried out by the mail distributor and the data base manager. These functions include:

1. Checking the KBS mail queue for incoming documents, retrieve the document, if any, and deliver it to the intended recipient user mail subdirectory. Update the KBS and user mail queues accordingly.

2. Interface with the user-key legend to load the intended WS's facts-list to get the WS's address, its user mail queue path name, and his document delivery expiration time.

3. Set the expiration timer by the time the document is deposited to the WS's user mail subdirectory.

4. Storing of the document to temporary and permanent archives.

All these functions have been catered for in a modular fashion.

The main C program is designed in such a way that when the document is received by the KBS, the header unit is stripped out from the document. The status-field and list-of-stops are stored in a separate file named status.add. Each stop of the control header stops, information are stored in a separate file named after the stop as stop#.add. These files will be transformed to facts-lists so as to be loaded into CLIPS for document processing. According to our model of the 5 level organization structure, if a document is intended for two readers, then the number of stops needed will be seven, for the worst case where the document writer is in the lowest level, 5, and hence he needs 4 designatory signers.

The details of the KBS CLIPS code can be found in Mohamed's dissertation [Mohamed, 1988].

6.0 SUMMARY

An Automated Document Distribution System (ADDSS) that satisfies a large corporation requirements for document processing and distribution, and which also handles the signature release authority requirement has been prototyped. The design was based on artificial intelligence concepts and distributed computer networking. The design project has been implemented using two approaches, a conventional programming approach using the C language and an artificial intelligence approach using CLIPS expert system language.

The basic assumptions that when a workstation user is not available his workstation will be set OFF, and that all communications have to be initiated by workstations only, impose some constraints on the system performance. They hinder the speedup of document delivery process. All documents retrievals have to be initiated from workstation side only. These assumptions are necessitated by the state-of-the-art commonly used PCs at users' workstations. If a multitasking environment is available at the workstation side, the KBS can initiate sending documents to the workstations, where the user can be interrupted for an incoming document, or the document can be deposited on the workstation side while the user is busy with another process thus speeding up the document delivery process.

The scope of this project is to consider the case of a LAN with one knowledge base server (KBS). To extend the project to a set of compatible LANs or incompatible LANs using Internet Gateways, belonging to different organizations as might the case in normal situations, some additional features and considerations have to be addressed. This will be the next phase of the project. The knowledge base server will have to perform safe storage and forwarding of the documents i.e., the KBS will include a network node functions plus the document processing and distribution functions. Knowledge base servers have to communicate with each other through the distribution system in real-time, and perform document processing and distribution in real-time too, thus real-time expert systems implementation at least at the KBS side will become a necessity.

The restriction that all communications have to be initiated from workstations side only has to relaxed especially, with the advent of multitasking environments being common features on PCs, so that a workstation can receive a document sent from the KBS while the user is working on another task. The user can be interrupted or alerted for new incoming documents while working on a different task from a background process.

Some form of parallel processing can be implemented so as to allow for document depositing
and retrieval to be done simultaneously at any of the KBS mail queues to improve system performance and speedup document delivery process.

A more sophisticated scheme that will consider the document value and level of importance as a factor in determining the document delivery expiration time may be implemented in the future by using an expert system that can perform natural language understanding features and have some learning capabilities.

REFERENCES


**FIGURE 4.** ADDS DISTRIBUTION SYSTEM

**FIGURE 5.** AI-BASED WORKSTATION ARCHITECTURE

**FIGURE 6.** KNOWLEDGE BASE SERVER (KBS) ARCHITECTURE

**WS FEATURES**
- HIGH SPEED CPU
- 640 KB RAM
- COLOR MONITOR
- RS-232 NETWORK INTERFACE
- ENHANCED KEYBOARD