CCTIS: An Expert Transactions Processing System

Terrance Swift
Computer Science Department
SUNY at Stony Brook
tswift@cs.sunysb.edu

Calvin C. Henderson
Systems Development and Analysis
P.O. Box 849, Purcellville Va. 22132

Richard Holberger
Systems Development and Analysis

John Murphy
DHD Systems, Inc.
2222 Gallows Rd. Dunn Loring, Va. 22027

Edward Neham
Systems Development and Analysis

Abstract

CCTIS, the *Cargo Container Targeting Information System*, was developed for the U.S. Customs Service to help monitor and control goods imported by ship. As an expert system, CCTIS has a combination of features which make it of interest to the applied A.I. community. First of all, CCTIS interacts with a large database — but unlike most data-oriented expert systems CCTIS is used in a transactions-oriented environment and needs the speed of such a system. Secondly, there exists no single cognitive model for the domain of import control, and it is unlikely that such a model can be developed in the near future. To address this problem CCTIS includes the ability for users to weigh and parameterize rules. And thirdly, the information CCTIS uses is often derived from free text of low quality that must be corrected and analyzed through natural language analysis techniques. The system uses a logic-based approach to solving these problems, defining explicit algorithms to extract data from text, and logical rules to analyze transactions. Our experience shows that this approach can produce a robust system. CCTIS has been used every business day for over a year in the two largest ports in the country, and has aided in the seizure of a number of illicit goods. Design is underway to merge CCTIS with another Customs A.I. system and to deploy the resulting system nationally, processing every sea-based import into the U.S.[1]

Introduction

The principle means of importing goods into the United States is through containerized cargo. Over 9 million entries were filed for cargo imported into the country in 1993, through about 50 ports of entry. Accurate accounting of imports is important for three reasons. Firstly, cargo — whether it is cocaine or mongooses — may be illegal to import. Secondly, even shipments of legal material are subject to quotas and duties: in fact Customs is the second largest generator of federal revenue after the I.R.S. Finally, from a more general perspective, the number and kind of imports is valuable as economic data: vague or inconsistent information about imports lessens the quality of that data. While the U.S. Customs Service has over 1600 seaport inspectors, it obviously cannot inspect each cargo shipment. The purpose of CCTIS is both to aid inspectors in prioritizing what shipments to inspect, and to facilitate passage of low-risk imports into the country. It should be mentioned that CCTIS is not the only A.I. system developed for Customs: current work on CCTIS includes integration with these systems.

From the point of view of Customs, an importation is represented as a bundle of information, coming primarily from two different sources. Each carrier that transports cargo to the country must submit a *manifest* to Customs. This manifest contains a collection of *bills of lading* each of which:

- States the port of lading of the commodity
- Contains information about the *shipper*, *consignee* and *notify party* for all cargo.
- Textually describes the cargo, its destination, weight, etc.
- Provides a transcription of shipping labels on each container.

The manifest is usually delivered to Customs before the arrival of the voyage it describes. While it contains a few formatted fields, information in the manifest is mostly free text. Meanwhile, Customs is also getting information about the cargo from importers. A *filer*, usually a customs broker or agent, files an *entry release form* stating the nature of the cargo and that the consignee is ready to receive it. The entry covers most of the information in the manifest, but is compiled by different sources for different purposes. The data quality in entry forms is better than in manifests, although information from free text fields is still needed. When the importer pays any duties on the cargo it can be released, and then he or she also files an *entry summary form* which presumably corrects any mistakes in the entry release form and serves as the final, official document of import. In general, there is a many-to-many relationship between bills and entries, which, when matched together are called a *shipment.*

To sum up, the data about cargo comes to Customs from different sources, asynchronously, is often

[1] This paper reflects the opinions of the authors only, and does not represent policy of the U.S. Customs Service.

free text, and there is a lot of it. Clearly, while a range of computational techniques are needed to wade through this morass of data, A.I. is crucial to extracting and interpreting the information, and in determining whether the sources are consistent.

In order to help inspectors survive in such an environment, an expert system, CCTIS, which stands for the *Cargo Container Targeting Information System*, was developed for the U.S. Customs Service. CCTIS has a combination of features which make it of interest to the applied A.I. community. First of all, CCTIS interacts with the database of manifests and entries in various ports. But given the requirement that Customs expedite imports, CCTIS must fit into a transactions-oriented environment. Secondly, CCTIS needs to apply rules over a domain, import control, in which there exists no single cognitive model, and where it is unlikely that such a model can be developed in the near future. To solve this problem CCTIS includes the ability for users to weigh and parameterize rules. And thirdly, the information CCTIS uses is often derived from free text of low quality that must corrected and analyzed through natural language techniques. Despite these challenges, CCTIS has been used every business day for over a year in the two largest ports in the country, (where it processes approximately 30% of sea-based imports), and has aided in the seizure of a number of illicit goods.

The next sections present an overview of CCTIS, and detail its use of A.I. techniques, particularly in its natural language analysis and rule base. Later sections describe the history of the system: its development, deployment, and maintenance, paying particular attention to evaluating the A.I. techniques, and indicating perceived strengths and weaknesses of the underlying software: Prolog and Oracle. We conclude with future plans for the project.

A User's View of CCTIS

CCTIS is currently installed in Newark and Los Angeles having processed about 1.5 million containers in each port since its inception. Both of these ports process around 5-10,000 shipments a day, and each have around 10 inspectors who use CCTIS for several hours each day. It should be stressed that the present deployment is being expanded to become available to every sea port.

It should be noted that until very recently, inspectors reviewed imports by flipping through paper manifests, without the benefit of entry information. As a first step in its deployment, inspectors used a clerical prototype of CCTIS. The prototype had rudimentary rules and standardization and contained only bill information. Given their previous environment, the users were glad to have the prototype, although they quickly recognized its limitations. Just as importantly, the prototype became a vehicle for the users to communicate their needs to the development team.

One of the most important points learned is that users have different ways to think about a shipment. This has implications for the structure of the rule set, as discussed later but also for the user interface. Some users start their session with a screen of manifests, others with a screen of shipments. Accordingly, the interface allows a user to filter and sort the set of shipments he or she wishes to view in multiple ways. For example, the user might filter out those shipments which are most urgent to review – because they haven't been reviewed yet perhaps because they contain perishable commodities. Alternately, they might be reviewed on the basis of shipment weight, or because one of a set of rules fired for them. In any case, when the user is done reviewing a shipment, it is marked as reviewed in the data base, information which is made available immediately to other users at their next screen refresh.

Figure 1 presents a sanitized version of the screen CCTIS uses for matching bills with entries. Characteristics of the shipment, such as its vessel and voyage number, and pertinent dates are written at the top of the screen. Below this general information, bill information is presented on the left, and entry information on the right. The bottom of the screen contains a list of rules which fired for the shipment.

Data presented to the user is hypertextual. Selectable text is blue, non-selectable text is black. For instance, clicking on a commodity presents a user with a summary of the countries from which the commodity has been recently imported, the value of the imports, their total weight, and so on. This data is obtained from *profile tables* maintained by the system for each port, and which also contain information on entities (shipper, notify, and consignee parties) and their relationships. In all, each port has about 2 gigabytes worth of profile information.

CCTIS Architecture Overview

The architecture of CCTIS is best described through a high-level data flow description. Figure 2 shows a high-level overview of CCTIS. Data can be conceptualized as a stream of manifests and entries downloaded from a central mainframe into ASCII files on the local UNIX CCTIS server. (Represented in Figure 2 by the topmost queue). The *load organizer* module of CCTIS polls for these files and, if it finds one, performs three main tasks. First, it parses and analyzes any free text information into an internal format. (This will be discussed in more detail below). Next, because there is a many-to-many relation between bills and entries and because they arrive asynchronously, the load organizer searches a relational DBMS for the any bills or entries which match the input. Finally, certain components of the bills and entries are checked against their profile information. The rule set might use this information to determine that a shipment of bananas from Finland was be suspicious. The full set of standardized and collated data, along with relevant

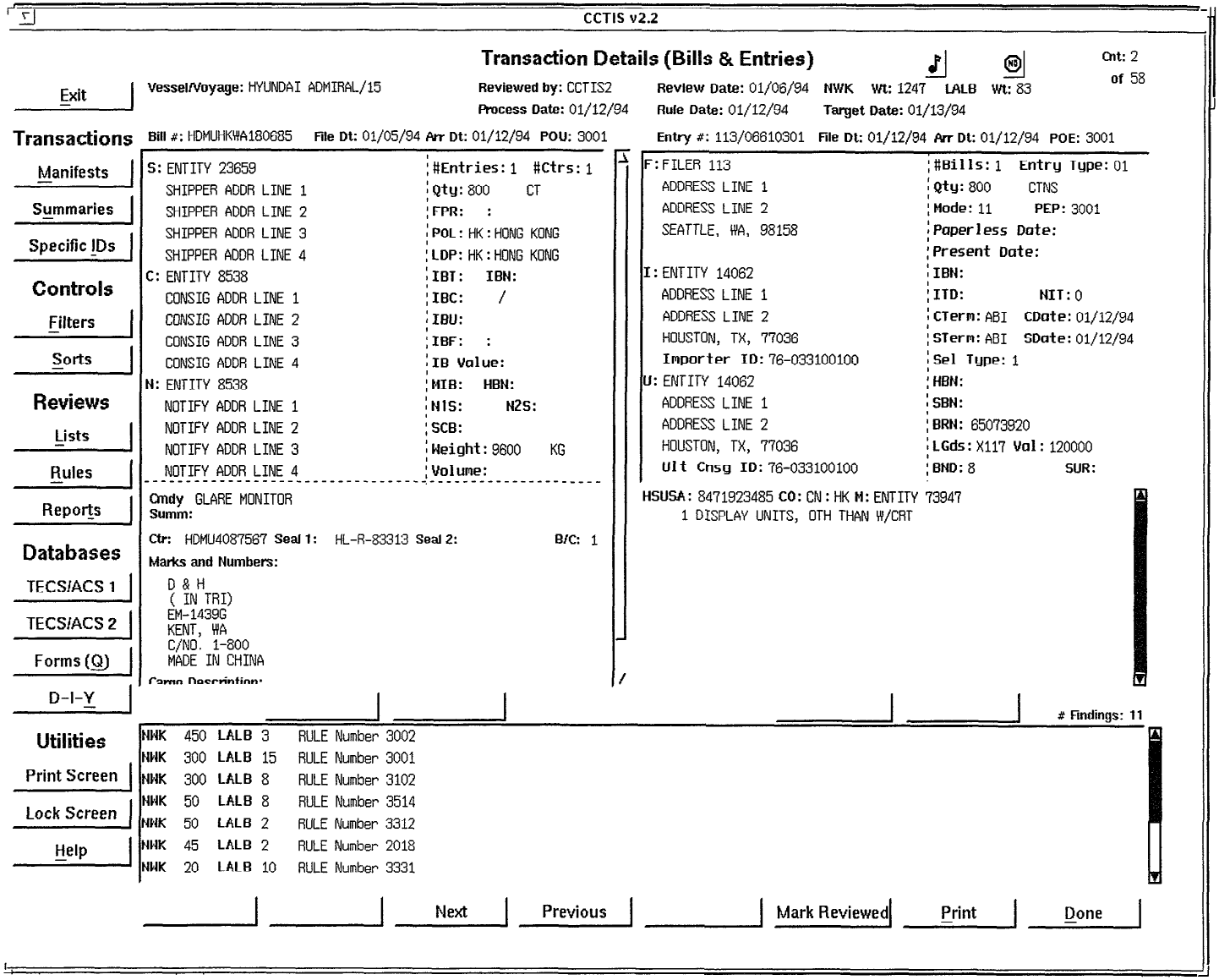

Figure 1: Screen for Matching Bills and Entries for a Shipment

information from the profile tables is written out to a *transaction file*. This file will be read by a *rule manager* which executes the expert system rule base against the shipment.

The rule managers apply portions of a set of about 300 expert system rules to each shipment. The applicability of each of the rules depends on attributes of the shipment, including whether previous rules have fired. Rules vary from simple checks of flags in the shipment files, to data-driven database searches, to complex checks of the consistency of the information in a shipment. Because of the lack of formalization of the domain of application, rules can be both parameterized and weighted. The parameterization is through *system parameters* and *match criteria* both of which can be thought of as user-defined logical predicates. The *weight* of a rule which can also be user-defined. The sum of the weights of all rules which fire for a shipment (the *weight* of a shipment) is used to help decide which shipments to investigate. A more detailed explanation and analysis of the CCTIS approach to rules is presented in a later section.

The rule firings for a shipment are written into the DBMS. When a user wishes to view shipments, he or she goes through the user interface, which accesses the database of shipments and rule firings built in the previous stages.

We have alluded to our actual implementation in our data flow description, and we complete the high-level description of our implementation here. CCTIS has been implemented and deployed on a SUN 670. The sets of download files can be thought of as its first queue, to be read by one of a group of load organizer processes. The set of transaction files, written by the load organizers and read by one of a group of rule managers can be thought of as another. Using formatted files and the database as an interface between modules handles the asynchronous nature of the input and provides for coarse multi-processing. Accordingly the number of load organizers and rule managers is adjustable, as are the times of their invocation. This multi-processing was motivated not only by the underlying SUN architecture, but also by the fact that transaction processing has a I/O component that is considerably reduced by the (somewhat gross) multi-threading in Figure 2. Oracle was chosen for the DBMS, the load organizer was written in C calling standardizers written in Quintus Prolog, the rule manager in Quintus Prolog. The user interface layout was developed in Motif UIL using ICS's Builder Xccessory, and the user interface callback modules in Motif and C.

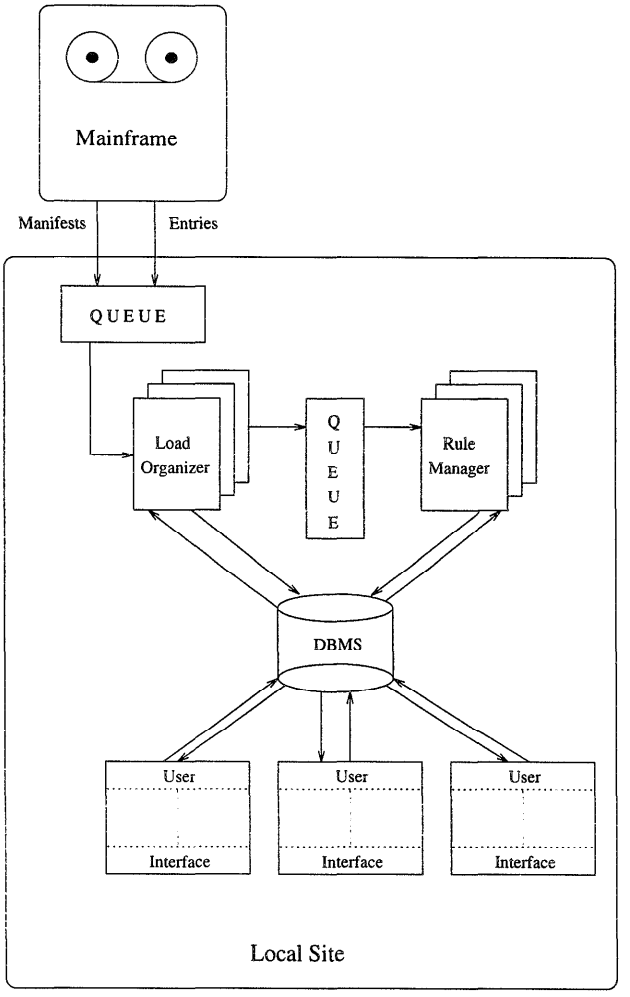

Figure 2: Top-level architecture of CCTIS

CCTIS Standardization

Several types of free text fields must be standardized and formatted before rules can use information in them. Of particular difficulty is standardizing names and addresses from bills of lading. The bills of lading purportedly contain names and addresses of enti-

ties involved in shipments. In practice they contain misspellings, abbreviations, both domestic and foreign addresses, multiple entities and addresses with their relationships, phone and fax numbers along with extraneous information not necessarily of use to the system and not necessarily in English. Because of these complications, commercial address standardizers were not deemed appropriate for the raw data. A typical example is provided: [2]

```
TO THE ORDER OF DATAWHACK DE
MEXICO S.A. DE C.V. CALZADA
VALLEJO 123, COL. IND. VALLEJO
```

Field separators are indicated by line breaks, which here have no bearing on information in the address. For this input the address standardizer produces

```
Entity: (base)
     Name: DATAWHACK DE MEXICO
     Title: SA
Address:
     Street: CALZ VALLEJO 123
     Town: VALLEJO
     Country: MX
org_type = 1
```

extracting the proper entity name and address, and inferring from the title, `S.A. DE C.V.` that it is an organization rather than an individual. It should be noted that because the organization name was made up, the standardizer had to parse out the name and address rather than doing a match against a database. In general, this facility is needed to handle small-time importers and badly misspelled names.

Parsing out names and addresses from free text is an exercise of interest in its combination of information retrieval techniques with natural language processing, rather than because it stretches the limits of natural language processing. As natural language analysis, processing bills of lading does not lead to questions of logical reference, say, or problems of belief. They do have certain problems of logical ambiguity, though. To take one example, in the text `ENT1 CARE OF ENT2 ADDRESS`, it is unclear to which entity the address belongs. It turns out that the appositive choice is not always the best, and determining the correct solution depends on knowledge such as whether the second entity is a shipping line [3].

After an initial tokenization, standardization takes place in three phases,

- A bottom-up parse corrects misspelled tokens and super tokens, e.g. 'SALT LAKE CITY' from the tokens 'SALT', 'LAKE', and 'CITY'.
- A top-down, frame-oriented parse which uses a DCG grammar to fill in slots of the frame.
- A post process resolves inconsistencies, and fills in missing information when possible.

[2]This example was based on actual data. Tokens were changed to ensure the privacy of the organization.

[3]The current standardizer assumes the address is appositive, however.

The structure of the name-address standardizer resembles the structure of many other natural language parsers, and we focus our discussion on correcting tokens in each of the phases. Names and addresses are notable for their prevalence of proper nouns, complicating the problem, and have prevented us from deriving an accurate estimate of the percentage of misspelled tokens. Our approaches center around minimum edit distance algorithms (Sankoff & Kruksal 1983), which define metrics between strings. These algorithms usually define as the distance between two strings the number of insertions, deletions or replacements needed to transform one string into another. The difficulty with such algorithms is their quadratic complexity for fixed string size, so that they are unsuitable for context-free searches (which must match against many possible tokens). Our approach to context-free searches is closely related however. For a list of crucial keywords, a set of tokens that have edit distance of 1 from a keyword can be generated and made into a table. For instance, if `CHICAGO` is declared to be a keyword, a table of translation clauses can be generated. This method substitutes space for time, and is sometimes called a reverse minimum edit distance algorithm (Kukich 1992). The tables vary from the extremely simple token substitutions like

```
translate('CHICAAGO','CHICAGO').
```

to the slightly more complex

```
remove_spaces('CHICAG','CHICAGO',
                        ['O'|Rest],Rest).
```

The latter clause can be interpreted as stating that if the token `'CHICAG'` is followed by the token `'O'`, the token `'CHICAGO'` should be produced, but the rest of the input stream should not change. In generating misspellings, each misspelling must be checked against a lexicon of keywords, cities, countries, and so on to avoid inadvertent transformations.

More difficult correction occurs when tokens have run together. In the following, `'HANJIN'` is a legitimate shipping company that is sometimes abbreviated `'HJ'` and both are taken to be prefixes which should be split out of tokens. It is not at all unusual to receive tokens like

```
HANJINBUSAN
HJBUSAN
HANJINBUSN
```

all representations of the vessel Hanjin Busan. In the third token, the abbreviation `'BUSN'` cannot be corrected by the above means unless the token is split apart. CCTIS uses what we believe is a novel programming technique to correct these tokens. From a list of prefixes, a trie is generated which moves from state to state as it reads the character list of each token. The trie itself can be generated as Prolog code that looks much like the tables above. An example provides a flavor of how the trie works. Assume the trie is in the state where it has read an `'H'`. The state of the trie is terminal (in this example) and can be represented by the clauses

```
remove_prefixH(65,[78,74,73,78|Suffix],
                              'HANJIN',Suffix).
remove_prefixH(74,Suffix,'HJ',Suffix).
```

In Quintus Prolog, the first argument, the character representation of 'A' or 'J' is used for index. If the current character is 'A', the next states of the trie, where 'N','J','I', and 'N' are recognized can be folded into this state since it is deterministic at this state. If the lookahead fails (as it would for the token 'HANOVER') no transformation is made. The trie generator has proven robust even though it is only about 100 lines long. We believe examples like this illustrate the power of Prolog for the general programming techniques that always accompany A.I. applications [4].

Until now, context-sensitive transformations have not been mentioned. They follow the same general pattern as context free transformations, but here a straight minimum edit distance algorithm can be used without excessive cost. Bills often have text like

```
SAME AS CONSINGE
```

in the notify slot. Here *consignee* is badly misspelled. In this case, after the (DCG or bottom-up) parser has recognized SAME AS, it can call an edit distance predicate against a few selected tokens such as `'CONSIGNEE'` and `'LAST'`.

Minimum edit distance algorithms are also used in the post processing phase since this phase has a great deal of contextual information to limit the search. For instance, if the city field is `'EDISNOE'` and the zip code is 08817, the standardizer would use a minimum edit distance algorithm of `'EDISNOE'` against the proper city for the zip to allow the transformation into the correct city, `'EDISON'`. If the city determined from the parse does not match the proper city for the zip code, and its edit distance is greater than the specified maximum, the address is inconsistent. At this stage, no determination is made about whether it is the city or the zip (or both) which is incorrect.

CCTIS Expert System Rules

The domain of CCTIS is import control. Unlike other domains for which expert systems have been built, such as chemistry, medicine or engineering, the import control cannot be systematically studied: there are no textbooks on smuggling. We claim that one result of this lack of codified knowledge is that the opinions of experts differ even more than they would in scientific fields. Another, perhaps deeper difference is that predictions of illicit imports are *reflexive* (Buck 1962) in the sense that successful predictions of illegal behavior will cause a change in the behavior under study. These considerations imply that an expert system for import control must have rules which are both easily *configurable* and *changeable*. This section discusses CCTIS's approach to rules along with its advantages and disadvantages. For obvious reasons, the discussion will need to be at a high level, since the rules themselves are deemed sensitive.

Allowing users to write their own rules would provide the ultimate in a configurable system, but is impractical from both a technical and a managerial standpoint. Rather, CCTIS allows users to customize rules through *system parameters*. System parameters reside in the DBMS, and are seen by Prolog as clauses of the predicate `sys_parm/2`. They can be changed either by the system administrator, in the case of parameters which are shared throughout a site, or by an individual user. Usually, parameters which express more general knowledge are decided by committee and shared throughout a site. More particular information, may be represented by match criteria, which allow a user to create a rule by specifying a regular expression on a field of a shipment or on a combination of fields. Match criteria can be added by an individual user, and may be shared or, if speculative, may not be. The set of three hundred rules use about 30 system parameters, and allow about 20 different types of match criteria.

A second means of changing the rule set is by weighting rules. Rules can be classified as intermediate *lemmas* which describe benign characteristics of a shipment, (such as the probable country of origin of cargo), and *weighted* rules. As has been mentioned, incoming shipments pass through a rule base and accumulate a weight, based on the weights of individual rules. The rules can have their weights modified by individuals to reflect the rules usefulness. Rules which are parameterized through match criteria may be weighted after parameterization.

Because a weighted rule can have a weight of 0, the distinction between weighted rules and lemmas is not strict. Formally, the weights of the rules, if properly scaled, can be seen as a likelihood ratio [5]:

$$\frac{P(SuspiciousShipment|Rule)}{P(SuspiciousShipment|\tilde{}Rule)}$$

If the assumption is made that the rules are independent of each other, then the likelihood ratios are multiplicative. In this case, the likelihood ratio of a suspicious shipment given a set of rules is the product of the likelihood of the rules taken separately, or if logs are taken the joint likelihood is the sum of the separate likelihoods. If logical combinations of weighted rules are to be expressed the constituent rules must have weight 0 for this interpretation to hold. For instance if A and (B or C) have to hold in order for a shipments weight to be substantially increased, the

[4] Recent experiments in XSB(Sagonas, Swift, & Warren 1994) compilation illustrate the efficiency of techniques like those mentioned. Using similar tries as an indexing transformation can give significant speedups over naive abstract machine code for many Prolog predicates.

[5] We are indebted to P. Szolovits for originally suggesting this interpretation.

weights of A, B, and C must all be set to 0. While the independence assumption for CCTIS is not violated by the logical form of the rules — no clause of any rule subsumes a clause of another — it is possible that certain rules could be statistically correlated with one another.

The CCTIS team has found that users have been generally satisfied by this model. Rules are changed and reparameterized in response to intelligence information and allow the different districts of Customs to maintain their own corporate cultures. Reflecting this organizational characteristic, weights often differ radically between field sites. In Figure 1, which is a sanitized version of an actual shipment, Newark weightings assign the shipment a weight of 1247, while Los Angeles/Long Beach weightings give 83. Discussions of the differing weights and parameters would provide an excellent vehicle for transferring knowledge among field sites, but this has not yet proven possible.

Over the course of the deployment, we have noticed three trends in the users' response to weighted rules. First, the range of weights for rules widened as users discovered that certain rules fired relatively frequently (or because the data was not standardized properly) and consequently were weighted less. Likewise, as certain rules aided in seizures they were weighted more heavily. Next, it turned out that certain shipments would cause particular rules to fire repeatedly with different values, and many requests were made to aggregate the results of these rules. Finally, it was determined that while certain rules deserved low weighting in themselves, they became more important in conjunction with other rules, and combination rules were developed.

A disadvantage of this approach is that, because of the close involvement of the user in determining the weights of the rules, it is difficult to introduce uncertainty measures among the intermediate concepts in such a way that they affect the weights of the rules in a sound manner and are also understandable by the user.

Evaluation of CCTIS

Table 1 presents a rough estimate of the amount of time needed to specify and code various parts of CCTIS along with the number of lines of code used in the current version. While some of these modules, such as the installation and administration scripts, do not have a A.I. component, we provide them as a basis of comparison. CCTIS was developed by a small group of experienced programmers and designers. Together, about 1.5 person years were spent in design, about 3.75 in coding and testing, and a little over a person year in maintenance and documentation. The final product has about 70,000 lines of code, not including generated tables for the standardizers or UIL files generated by Builder Xccessory. Not surprisingly, design of the rule manager and user interface comprise most of the specification and knowledge acquisition time.

User Training and Support, including documentation, has taken a great deal of time, reflecting in part efforts to introduce an A.I. system on a UNIX platform into a corporate culture used only to mainframes. As an instance, non-technical users do not differentiate between various aspects of an application system, especially when they are first introduced to it, and the times in table 1 reflect this. However, much of the effort towards user acceptance, such as writing user manuals, will not have to be repeated for new ports.

One avoidable maintenance problem for CCTIS is that much of the processing occurs in the local site (as depicted in Figure 2), and is harder to maintain than a centralized site. To lessen deployment and maintenance costs, we are moving toward a centralized architecture for CCTIS and another Customs A.I. system. In this architecture, the load organizer and rule manager tasks would be run at headquarters, while the user interfaces would run off local snapshots of the CCTIS database. Of course, it is critical to continue to provide users with the control they currently enjoy in, for instance, parameterizing and weighting rules.

CCTIS has aided in a number of seizures, but we do not have hard statistics on the likelihood of a seizure using CCTIS as opposed to the likelihood not using it. Because of the difficulty in answering that question, it is doubtful that enough resources will be spent to answer it. Indeed, only a small fraction of inspected containers are seized, whether or not an expert system is used. Furthermore, experimenting in our domain, say by intentionally trying to smuggle goods into the country, is impractical. Because predictions are reflexive, it also begs a cognitive question: the pseudo-smuggler may use his knowledge of the system to beat it. On the other hand, users are now able to mark as reviewed 60-70% of shipments a day, a far higher number than could be reviewed using paper manifests. This number is especially impressive given that unreviewed shipments are likely to be those weighted the least suspicious by the expert system.

Evaluation of CCTIS tools

As mentioned above, the A.I. components of CCTIS are written in Prolog, and the knowledge base is distributed between Prolog and Oracle. On one level, these products were chosen because of the developers familiarity with them, but there are deeper reasons as well. It is well-known that the logic programming paradigm is highly suitable for natural language analysis, both through DCG's and because Prolog's relational model allows a transparent access to knowledge bases. (No doubt, the functional paradigm would offer its own strengths).

Prolog's strengths for writing rules are perhaps less appreciated. Prolog clauses have an if-then formulation and simple logical syntax which make them appear suitable for expert systems. It can be argued though that a fundamental impedement to Prolog use

Module	Speficication	Coding	Lines of Code
Load Organizer - non-standardizers	3 p/m	9 p/m	12,000
Load Organizer - standardizers	1 p/m	3 p/m	13,000
Rule Manager	8 p/m	6 p/m	9,000
User interface DB Access Code	1 p/m	6 p/m	12,000
User interface Display Code	6 p/m	16 p/m	22,000
Instalation / Administration Code	1 p/m	6 p/m	5,000
Total	20 p/m	45 p/m	73,000
User Training	8 p/m		
User Support	6 p/m		

Table 1: Development Effort of CCTIS

for expert systems is in its use of backward-chaining SLD. Prolog can go into an infinite loop when evaluating recursive 'feedback' rules, or at the very least perform redundant subcomputations prevented in a bottom-up environment. As a result, Prolog alone is not yet suitable for naive users or specifiers. Recent experiments which add tabling to Prolog (see, for instance (Sagonas, Swift, & Warren 1994) which uses SLG evaluation), alleviate this problem, but may not have the robustness of commercial Prologs.

However, when programmers in charge of a rule set are not so naive, Prolog is an excellent choice since it can provide facilities for coding atomic rules, their control, their aggregation if necessary, and the glue to interface with other modules. Expert system shells usually provide these facilities through an escape to some sort of procedural language, whereas in Prolog a single language can be used.

Other improvements to Prolog that would have payoffs for systems like CCTIS are better facilities for determinacy and better access to knowledge bases. Determinacy detection is important because realistic Prolog programs nearly always include cuts or conditionals to control evaluation. These constructs make the code less declarative, and if mishandled, can make the code slower as well. Automatically detecting determinacy through a compiler is undecidable in general: incorporation of analysis techniques to detect it in certain cases is an open research topic (Dawson *et al.* 1993). At the same time compilation techniques for deterministic logic programming languages such as FGHC are well known (Ueda 1987). The addition of optional guards to Prolog would allow the programmer to obtain the speeds of deterministic execution when possible, and the flexibility of non-determinism when required.

Another issue of importance to expert systems in general is how to access and update a knowledge base. Prolog is often efficient at accessing small knowledge bases of roughly the size of main memory [6] — as long as the data is in memory and properly indexed. A simple call to an external database takes about 10,000 times longer than a call to another Prolog predicate. When it comes to updating a knowledge base, however, most Prologs fail to provide even the most rudimentary functionality. In Quintus, code can be either static or dynamic. Static code can be saved into or loaded from object files, at roughly the bandwidth of reading from a file system, but the static code cannot be updated. Dynamic code can be updated, but it cannot be loaded through object files, and instead must be read and compiled from ASCII files. Creating object files for dynamic code would allow a knowledge base suitably updatable for a lexicon with the access speeds required for one.

Future Plans

Given its ambitious functionality and the relatively small amount of programming time that has gone into it, it is not surprising major elements and tasks remain undone. Some of this work will take place when CCTIS is merged with the *Targeting Information Management System (TIMS)*, a separate A.I. project which has also been successful in the field. While sharing certain of the CCTIS functionality, TIMS is a complementary system in many respects, especially in its use of probabilistic neural nets in its rules. We believe the neural approach of TIMS and the logic-based approach of CCTIS will balance each other and their juxtaposition will lead to better validation of each approach [7]. The resulting system, called the *Automated Targeting System (ATS)* will be national in scope and rectify some missing elements of CCTIS. We discuss some of these from an A.I. (and CCTIS's) point of view.

One major extension is the use of data and knowledge base information not available heretofore. Aspects of the ATS/CCTIS domain, such as locations and names are shallow but data-intensive. We are currently evaluating commercial address standardizers

[6] The CCTIS developers were involved in an development effort which developed a Prolog expert database system whose running size was 1/2 gigabyte. The system, which was unrelated to CCTIS, lasted for three years in the field.

[7] It should be mentioned that TIMS offers some logic-based rules not covered by CCTIS, as well

which can be used as a post process stage to the CCTIS/ATS standardizer and which contain information that will allow validation of street addresses, apartment numbers, certain firm names, and so on. We are also trying to obtain geographic data which will be useful in visualization and in rules themselves. Furthermore, we will be refining our parsing techniques on fields like cargo description and on the shipping label transcriptions.

Not less important is extending the rule validation process beyond its present state. As mentioned above, rule validation is difficult in our domain. The following techniques may be of use however.

- For seizures that are made without the use of an expert system, run the shipment through the rule set in each port to determine which factors are effective and which are not. This technique can also help identify rule sets in particular sites which may be ineffective.
- Build a correlation matrix for the different rules. When rules are strongly correlated their weights may effectively be higher than anticipated, and the weights may be lowered or the ruled disabled.

Acknowledgements

The authors would foremost like to thank our users, whose ideas have improved CCTIS in every aspect and Tim Hawes and Tony Maresca who have supported our efforts in developing the system. David Alberts was instrumental in getting the project started, Carl Krebs wrote and designed much of the original rule manager, and Ed Jenkins much of the user interface code. On a technical level, this paper has benefitted from comments by W. Dougherity, T. Ewing, P. Szolovits, and J. Hill. We also thank O.N.D.C.P. which has partially funded this project.

References

Buck, R. 1962. Reflexive predictions.

Dawson, S.; Ramakrishnan, C.; Ramakrishnan, I.; and Sekar, R. 1993. Extracting determinacy in logic programs. In *Proc. of the Int'l Conf. on Logic Programming.*

Kukich, K. 1992. Techniques for automattically correcting words in text. *ACM Computing Surveys* 377–441.

Sagonas, K.; Swift, T.; and Warren, D. 1994. XSB as an efficient deductive database engine. In *Proc. of SIGMOD 1994 Conf.* ACM.

Sankoff, D., and Kruksal, J. 1983. *Time Wraps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley.

Ueda, K. 1987. Guarded horn clauses. In *Concurrent Prolog: Collected Papers*, 356–375.