The Credit Act Advisory System (CAAS): Conversion From an Expert System Prototype to a C++ Commercial System.

George Vossos* John Zeleznikow* Allan Moore+ Dan Hunter~

Abstract

CAAS is a rule-based expert system which provides advice on the Victorian Credit Act 1984. It is currently in commercial use, and has been developed in conjunction with a law firm. It uses an object-oriented hybrid reasoning approach. The system was initially prototyped using the expert system shell NExpert Object, and was then converted into the C++ language. In this paper we describe the advantages that this methodology has, for both commercial and research development.

Introduction

CAAS is an example of a legal knowledge based system which aids in the process of statutory interpretation. The system reasons using a deductive[1] reasoning approach, referencing hypertext and databases only when needed. As such, the system does not perform any type of case-based reasoning with precedent cases. CAAS was never intended to replicate the expertise of a solicitor, but only to direct a lawyer unfamiliar with the Victorian Credit Act[2] to obvious breaches of the Act.

The system's goal is classifying loan transactions as either regulated, non-regulated or exempt under the Act. Once a classification has been determined, the user can proceed to interrogate the system as to reasons for its decision. Further, she has the option of viewing any statutory requirements associated with the particular classification. Advice provided by the system is based on facts entered by the user in response to questions posed by the system. Practitioners can experiment with hypotheticals. CAAS was designed using object-oriented design principles that enhance performance and permit the system to:

- advise whether certain credit contracts fall within the ambit of the Victorian Credit Act and related legislation, and if so, list legal obligations with which the creditor may need to comply;
- determine whether certain transactions are in breach of the Credit Act and explain why;
- enable users to easily input the characteristics of a problem case into the system in a non-technical manner, using menus and forms;
- permit client data to be saved, retrieved and amended;
- allow the user to browse through the sources of the law using hypertext techniques;
- allow users to pose hypothetical questions;
- produce reports capturing and explaining the reasoning of a decision; and
- allow users to check penalties and other statutory requirements set down by the Act.

Problem Description

At Allan Moore & Co. Pty ('the firm'), solicitors perform the task of diagnosing Credit Act related legal problems. Solicitors receive correspondence from clients and must act quickly to resolve the problems and provide concise commercially-oriented advice. The current manual handling of the problem-resolution process suffers from the following disadvantages:

- In most legal firms practitioners develop expertise in specific areas of the law. If the relevant expert is unavailable, then so is the expertise. If the expert leaves the firm, much of the expertise is lost.
- Solicitors work on the principle of maximising 'billable time'. Experienced staff members are unlikely to sacrifice time to take on the additional 'non-billable' task of training replacement staff members.

Solicitors at the firm carry out three basic functions for credit law queries: administration, diagnosis and referral. Whilst, administration and referral were determined to be functions of conventional technology, diagnosis required the application of Artificial Intelligence technology.

A software solution tightly integrating these functions was designed and deployed. Initially, a prototyping approach was taken to development, allowing intended users to participate in the design of the system while at the same

time pleasing management by developing an initial model of the system which they were able to monitor. The prototype was designed using Neuron Data's NExpert Object Development Environment: an interactive interpretive environment that makes use of graphical editors and browsers. Once the prototype had the approval of management, it was converted to a generic C++ program running under Microsoft Windows 3.1. During this conversion, the technical design was only slightly modified to fit the message passing paradigm of C++.

Overview of the Prototype CAAS Expert System

The first step in CAAS 's development involved capturing the firm's expertise in credit law and then translating this into a decision support system. In order to facilitate modelling, the knowledge engineers decided to opt for an expert system (ES) shell that

- supported a rich set of knowledge representation schemas;
- provided an application interface library (API) into and out of the main operating kernel;
- provided access to some of the more common databases; and
- supported the rapid prototyping development methodology.

After careful deliberation with the domain experts, it was decided that a hybrid object/oriented-rule-based approach best represents the heuristic knowledge needed to reason in this domain. An advantage of this approach was that the knowledge-base could be designed to fully exploit the power of the object-oriented paradigm[3] while at the same time taking advantage of the inference engine's exploratory power. Adopting this methodology greatly simplified the design and implementation of our knowledge-base. It was possible to develop the system with minimal programming which convince the firm of its value. An associated long term benefit was that the resultant knowledge-base was easier to port and maintain, as the various knowledge sources (classes, objects and rules) were clearly identifiable.

We adopted a sidewards chaining inference engine strategy: combining both forward and backward chaining of production rules. The system pursues a consultation by initially asking top level questions and then selects an appropriate hypothesis to pursue given the facts volunteered by the user, in addition to any new deduced assertions. The assertions are deduced by forward chaining. From this point onwards the system resorts to backward-chaining to frame questions.

Once the knowledge base correctly represented the decision making process of the experts, it was integrated with a graphical user interface (GUI), hypertext and a database. HyperCard was chosen as the ideal platform to perform this integration due to its i) support for rapid prototyping, ii) adherence to object-oriented principles, iii) full hypertext support and iv) the ability to call out to the host language and invoke binary code. Through the binary code interface we were able to access the knowledge-base.

Conversion

When the behaviour of the prototype was approved, it was ported to a PC based environment to be compiled into a binary format using a C++ compiler and Windows 3.1 We chose C++ because it is truly object oriented, runs on IBM-PCs and is commercially available. Though the use of C++ dictated the way we approached the design of the expert system (in the use of petri-nets, as described below) it did not change the fundamental nature of the expert system.

The porting process involved i) simulating the GUI and hypertext elements of the prototype under Windows 3.1; and ii) porting the knowledge-processing of the ES into C++. Converting HyperCard GUI and hypertext elements was simplified by combining tools such as the Borland C++ Resource Editor and Borland C++ OWL programming class library. To simulate the expert system module, we specified the expert system in the form of a petri-net[4] and translated this specification into C++. Class slots were used to represent system states, while their associated values were used to represent transitions from one state to the next. Once in this form, the petri-net was converted into a C++ program with meta-slot control simulated by message passing. This port to a 3GL environment was facilitated by ensuring that the expert system prototype adhered to the meta-slot control strategy. This type of modelling displaced control from the inference engine and trapping and handling control in the system was given to slot 'daemons'. In addition, specifying the expert system module as a petri-net meant that the developers could graphically validate the system. Significantly, this architecture and methodology is applicable in building future commercial legal expert systems.

Figure 1: Extracts of code used in CAAS, which helps illustrate the breakdown of the design into two main C++ classes, the Slot and the KB classes.

```
typedef char*     Field;
enum BOOL         {FALSE,TRUE,UNKNOWN,HELP};

class Slot
{
friend class KB;
        Field           Name;
        VType   Value;          //union      type
containing INT or STR
        Field           Why;
        Field           Comment;
        Field           PromptLine;
public:
                //constructors
        Slot(Field, SlotType, AVal, Field, Field, Field);
        Slot(const Slot&);

                //destructor
        ~Slot();
};

class KB
{
                        //hypothesis  slots
```

```
Slot*       loanInit;
Slot*       loanInit2;
Slot*       ldefcred;
            ......
                    //meta-slot daemons
Slot*       n_sdate85;
Slot*       n_sinwriting;
Slot*       n_crvic;
            ....
                    //attribute slots
Slot*       SDate85;
Slot*       SInWriting
Slot*       CRVic;
            ......

public:
                    //constructor
KB();
                    //destructor
~KB();
                    //generic 'get' method that
determines/allocates values to slots

BOOL   get(Slot**);
                    //knowledge methods: these
correspond to top level hypothesis)
BOOL   LoanInit();
BOOL   LoanInit2();
BOOL   LDefCred();
            .......
};
```

Figure 2: Translation into C++ of the LoanInit rule.

```
BOOL KB::LoanInit()
{
    if(     get(&SDate85)          AND
            get(&SInWriting)               AND
            get(&CRVic)
      )
      {
            loaninit->Value.actualVal.iVal = TRUE;
            return LoanInit2();
      }
```

Development Considerations - Architecture

The value of CAAS lies in its knowledge representation and control strategy. Knowledge is captured in classes, objects and rules specific to the domain. Development proceeded by acquiring necessary knowledge from domain experts, programming the knowledge into CAAS, demonstrating the diagnostic behaviour to experts, then refining behaviour as needed. CAAS uses the knowledge base to engage the user in a question-and-answer dialogue. This approach is natural for the solicitor, because the client is often on the telephone, and the solicitor must determine the exact nature of the problem by querying the client.

Two alternative Artificial Intelligence technologies were candidates for developing CAAS, but neither met the firm's needs. Case based reasoning was assessed as too slow for the requirements of this project and its diagnostic behaviour too difficult to control. Neural network technology was ruled out because it i) requires specialised software and hardware platforms and ii) has no way of justifying conclusions reached. The advantage of the hybrid rule-based/ object-oriented approach is simplicity—the knowledge base is in a format that facilitates maintainability.

Development Considerations - Engineering Methodology

Our experience has revealed that prototyping is a useful technique for developing knowledge based systems. Prototyping reduces the risks of producing a system which does not match user requirements, is too expensive or is unreliable. Not all systems however lend themselves equally well to prototyping. The CAAS knowledge base application was a particularly good candidate since

- the requirements were ill-defined; and
- the expert system was interactive, relying extensively on user dialogues

Developing CAAS involved many steps. The most important of these were—

Specification - Specifying CAAS could not be as detailed as for conventional systems largely because the knowledge that would emerge could not be known until well into the process. CAAS specifications were therefore left as a list of aims. These aims were modified during development as technical problems were resolved.

Feasibility - The feasibility stage of the development process involved analysing the resources required, the size and difficulty of the task and the resources and time necessary for the successful completion of the project. Twelve months was allocated for the completion of the whole system.

Knowledge Acquisition - We decided to adopt the engineer-driven approach. Here, the knowledge engineer learns as much as possible about the domain from the expert and then proceeds to translate the newly acquired knowledge into a representation that is suitable. The knowledge engineer began with the statute and was able to develop a first draft representation of the knowledge. Consultation with the legal specialists consequently aimed to refine the knowledge base.

Non Functional Requirements

User Interface & Human Factors - One of the requirements that was implicit in developing CAAS was the need for a sophisticated user interface. Intended users of the CAAS system were not using a computer system to access dead data but rather, manipulating knowledge. This manipulation needed to be performed intelligently if users were to be properly supported in the tasks that they conduct.

System Integration - To be successful, a tight integration of the modules was fundamental. It was insufficient to have separate applications handling each of the needed functions; instead, the movement between expert system, database, and hypertext modules had to be seamless and natural. The CAAS system should not be

viewed just as an AI application, but rather a solution where AI has a key role in concert with other technologies.

System Modification - Maintaining or updating the knowledge-base will always involve a knowledge engineer. It is unfeasible to allow domain-experts to change the contents of the knowledge-base, as the code is in binary form. Any changes made to the system will require changes to the source code which in turn involves having to re-compile and re-link the program code. The task of maintaining a knowledge base of objects and rules is considerably more difficult than maintaining a database of facts.

Integrating Commercial and Research Programmes

Initially, CAAS was developed on a strictly commercial basis. Allan Moore & Co wished to gain a marketing edge by developing a powerful, user-friendly expert system. CAAS has met these criteria, and has done so at a relatively cheap cost.

The developers of CAAS have been performing research in the IKBALS (Intelligent Knowledge Based Legal Systems) project. IKBALS has attempted to integrate rule-based reasoning, case-based reasoning and intelligent information retrieval using the principles of distributed artificial intelligence and intelligent and co-operating information systems. The porting of CAAS to the C++ language has suggested to the IKBALS project techniques for developing such intelligent co-operating legal information systems (ILCIS), and the IKBALS project has developed a methodology for designing such ILCISs.[5]

Conclusion

In constructing CAAS, we have developed a framework for building commercial legal expert systems using C++, rather than relying on more expensive expert system shells. The system is being commercially marketed to organisations which provide credit. The generic nature of the system facilitates interfacing and integrating to existing software such as document modelling and precedent management. As a follow up, the firm has asked us to develop legal expert systems in the areas of bankruptcy, debt recovery, privacy and civil procedure. This work is currently in progress.

---

* Database Research Laboratory, Applied Computing Research Institute, LaTrobe University, Bundoora Victoria 3083, Australia, Tel: +61-3-563-6552

+ Allan Moore & Co. Pty, Solicitors, 11 Bank Place, Melbourne, Victoria, Australia, 3000, Tel: +61-3-602-2411, Fax: +61-3-602-1505

~ Law School, University of Melbourne, Parkville Victoria, 3052, Tel: +61-3-510-7655

1 The production rules forming the knowledge base are heuristic rules supplied by experts from Allan Moore & Co. As such, the system does not directly model or reason with statutes or precedents per se

2 The Credit Act, 1984 (Vic) and the Credit (Administration) Act, 1984 (Vic). For further information on the Act see: Cavanagh, S. W. and Barnes, S. , *Consumer Credit Law in Australia - Commentaries on the New Credit Legislation*, Butterworths, 1988; Duggan A. J., Begg S. W., Lanyon E. V., *Regulated Credit- the Credit and Security Aspects*, The Law Book Company Limited, 1989; Levine, J. R. , *Victorian Consumer Credit Legislation with Annotations*, CCH Australia Limited, 1984.

3 Object hierarchies inheriting both slots and methods were used quite extensively.

4 A Petri net is a directed graph with two kinds of nodes, places and transitions, interconnected by arcs - in such a way that each arc connects two different kinds of nodes (ie. a place and a transition). Such a graph is called a bipartite directed graph. For more on Petri Nets, see Rosenberg, G. (ed.), *Advances in Petri Nets 1990*, Lecture Notes in Computer Science, Volume 483, Springer Verlag, Berlin 1990.

5 For further information on this project see, Zeleznikow, J., Vossos, G. and Hunter, D., 'The IKBALS Project: Multi-Modal Reasoning in Legal Knowledge Based Systems,' submitted to *Artificial Intelligence and Law Journal*.