DEXA 90

A M. Tjoa and R. Wagner (eds.)

Database and Expert Systems Applications

Proceedings of the International Conference
in Vienna, Austria, 1990

Springer-Verlag Wien New York

Design and Implementation of Substantive Applications in Criminal Law: Beyond A Court Management Perspective

David E. Woodin, Esq.

Associate Law Assistant, Trial Part, Greene County Court
New York State Unified Court System
Greene County Courthouse, Catskill, New York 12414

This paper describes the experience of one New York court in extending its uses of microcomputers beyond the records management functions initially contemplated, to encompass substantive applications serving the judiciary in an advisory role. The applications described include a rule-based expert system in case disposition, and a system for the detection of ethical conflicts in the selection of assigned counsel. These substantive applications are seen to significantly extend the utility of the standard database functions typically encountered.

Microcomputers have been employed in the management of the court system of the State of New York for several years. As envisioned by the N.Y.S. Office of Court Administration (OCA), their primary functions have been to assist with indexing, scheduling, docketing, inquiry, and statistical reporting with respect to caseload processing.[1] As such, their users have been primarily non-judicial and clerical personnel. Impact upon judges has been indirect, through the reports generated in response to demand for case status information. The question has been raised whether there is not a more immediate role for the computer as a direct support to judicial decisionmaking. Several prototype systems have in fact been developed in other jurisdictions with respect to various judicial tasks.[2] The goal of the current work is to amplify the ability of existing database applications to support judicial functions, through the application of intelligent tools loosely coupled to existing database systems.[3] The benefits of this approach are modularity of design, and achievement of practical implementations. Its drawbacks include redundancy in information representation. The result, hopefully, will be the development of integrated systems able to directly support the judiciary as well as to perform ministerial recordkeeping tasks. Two examples are presented: a fully implemented system dealing with legal issues of case disposition, and a prototype system for the detection of ethical conflicts in the assignment of counsel in criminal cases.

In order to pursue the application of the computer as a tool to assist judges and law clerks in the substantive work of the courts, it is necessary first to consider the proper role of the computer in the judicial process. The process of "what judges do" when they decide a case in fact involves multiple processes. At present only a few of these processes are sufficiently well understood for an analytic model to be attempted. Roughly stated, they comprise those processes by which it is possible to solve "easy", or deductive cases.[4] Other processes involving intangible factors such as the exercise of moral judgment, may forever remain computationally intractable.[5]

Even restricting the analysis to so-called "easy" cases, there is considerable reason to hope for the successful application of substantive systems. A judge considering a case must be familiar with a wide range of rules and the conditions of their invocation. He must be alert to facts elicited from the parties which may trigger little used rules. He should consider the effect on the legal outcome of every relevant legal fact in the case being established or not established. He must know the effect of recent precedent on existing doctrines. The increasing number of statutes and regulations, the increasing factual complexity of litigation, and the burgeoning volume of case law all combine to make these formidable tasks indeed. It appears, therefore, that there is room for substantive systems in law which will solve "easy" cases and, by elimination, identify "hard" ones.[6] Because of the potential existence of intangible factors entering into every judicial decision, such systems should be seen always not as a substitute for the judge, but as a technical advisor to him or her. This premise underlies all phases of the present work.

The disposition of criminal cases in New York State typically involves tension between judicial or prosecutorial discretion and legislative regulation.[7] This tension is encountered at every stage of the proceedings, and is especially felt in the areas of sentencing[8], plea bargaining[9], consideration of lesser included offenses[10], initial charge[11] and indictment.[12]

The serious consequences[13] which may result from the failure to observe these limitations on discretion raise two challenges for the courts. The first is to recognize the circumstances in a given case which may invoke application of the various limiting statutes. The second is to accurately implement a statute whose application has been triggered by the facts of a given case. The need for powerful tools to assist judges in meeting these challenges is manifest. The solution proposed entails consultation of an expert system which is adept at recognizing the existence of the above issues, and suggesting routes to their resolution. Such a system should attempt to incorporate both the knowledge contained in the original written sources, and the experience of persons who have thoroughly studied the domain and who frequently deal with it. The system should be designed so as to be easily integrated into the existing database system of the court. Such a system dealing with the above issues has been constructed, and is described below.

The Expert System: An Overview

The expert system employed is an interactive MS-DOS PC-based substantive system, written in Turbo Prolog 2.0.[14] The program is compiled and is distributed commercially as a

stand-alone product under the product name *Gunga Clerk.*[15]

When consulted independently of the database system, a typical work session commences with selection of a particular offense by section number from a menu displaying all offenses loaded into memory. The title of the offense, its classification, and a summary of its elements are displayed for confirmation. Returning to the main menu one may elect to investigate sentencing options, plea bargains, or lesser included offenses with respect to the selected offense. Input is requested from the user during the course of a consultation through pop-up "Yes/No" menus. Selections are made by highlighting the chosen item and pressing "Enter." The response selected from the menu is incorporated into the output text. Output text may be echoed to the printer or to a disk file for later editing using a word processing program. The use of a disk file for storage of results is a key aspect of the program's integration into the overall database system, as will be seen.

The program prepares a list of all legal sentencing alternatives, together with detailed statutory authority for each potential sentence, by first generating every possible potential sentence, and then testing it against conditions for legality as expressed in the program's rules and accumulated facts input by the user. The program requests information concerning the defendant and the crime, as needed, to calculate the range of available sentences. The answers to questions posed by the program are incorporated into the report being generated, to provide a record of the consultation. Varying the responses to the questions posed, one observes the effect upon the available sentencing options.

With respect to plea bargains, the program first determines what, if any, statutory restrictions apply to the offense selected for analysis. Input from the user is requested as needed by the rule being executed. The program then displays the controlling statutory restriction, giving a detailed citation to its location, and explains why the restriction applies based on the information provided. Finally, it prepares a list of suggested offenses which satisfy the statutory plea bargaining conditions and which are potential lesser included offenses of the selected offense for purposes of plea. The user may select an offense from this list and immediately return to the sentencing module to learn the potential benefits of a plea of guilty to the lesser offense.

The program can generate two reports with respect to lesser included offenses. The first contains a list of offenses which are arguably potential lesser included offenses of the selected offense. A lesser offense may be added to this list under New York law when it is theoretically impossible, under any circumstances, to commit the selected offense without by the same conduct committing the lesser offense. The legal analysis is performed by comparing the elements of the two offenses, and by applying rules of legal inference between elements. The program collects offenses whose elements are all elements of the selected offense, or whose elements are legally implied by the elements of the selected offense. Where a conclusion depends upon such a chain of legal inferences, the chain of reasoning leading to the conclusion is explicitly displayed, along with supporting authority for each step drawn from statute and case law.

The second report contains a list of offenses which arguably are not lesser included offenses of the selected offense, regardless of their apparent similarity under the facts of a given case. An offense is added to this list when there is existing legal authority to the effect that the lesser offense is not a potential lesser included offense of the selected offense, because it fails the "impossibility" test mentioned above. This "non-included" list is obtained by extending the preceding method to include chains of reasoning containing links which have been held by case law to be invalid or false inferences. The list is then restricted to contain only offenses derived from the selected offense by a process of reasoning which includes one or more of these legally invalid inferences. The report produces the list of the offenses, along with the argument by which each was derived. The steps in the argument are explained and supported step by step as above, with the false or invalid links in the argument explicitly identified.

The final major feature of the program is a data-driven, forward-chaining process, in contrast to the diagnostic, backward-chaining process of the previous sections. Instead of selecting an offense by its Penal Law citation and analyzing it, the user may compare the facts of his case to an alphabetically sorted list of elements of offenses and highlight any group of these elements. The program will combine the elements selected, (and any other elements legally implied by those selected) building as many offenses as possible from the given set. This function assists in comparing the elements of the offense charged against the proof actually produced before the grand jury or at trial.

The Database System

The case records of the court are maintained in a relational database managed by a compiled Clipper[16] program. Separate files are maintained for case records, attorneys, diary information, counts of each indictment, and offenses. The database system is used for all of the case management tasks initially alluded to, including preparation of calendars and reports. The link from the database system to the expert system is accomplished by a simple modification to each program which allows one or more items of information, such as the citation of the offense under investigation and the type of analysis requested, to be passed as command line parameters. The code in the database program to call the expert system with parameters, and the extra code added to the system to allow it to receive, parse and process the parameters is relatively simple. The system need only recognize which of its data files must be loaded into memory, based on the citation of the offense being considered and the function being requested, and activate its various modules accordingly.

The return of information to the database is equally simple. The results of a consultation produced by a parameterized call to the expert system are automatically echoed to a standard log file, which upon return to the database program is read and appended to a text memo file attached to the current case record. These memo records in turn are utilized by the database system in preparing a summary report of each case upon demand for insertion in the court file, for review by the judge.

Selected Computational and Design Issues: A Grammar for the Generation of Sentencing Alternatives

A frequently documented application of Prolog is its use to implement parsers or generators of a given language. The structure of well-formed expressions in the target language is quite easily represented by a series of Prolog rules, which specify the ways in which the basic units of the language may be combined into complex phrases. Contemporary work illustrates the power of this technique applied to the generation of so-called "grammars of law."[17] Examination of the structure of New York sentencing law suggests a similar approach, the target "language" to be parsed being the taxonomy of authorized sentences. Article 60 of the New York Penal Law establishes a comprehensive sentencing scheme, organized hierarchically by type of sentence (see **Figure 1**). A simplified outline of the sentencing generator syntax is shown in **Figure 2**. Its associated Turbo Prolog code is shown in **Listing 1**.

PL 60.01. Authorized dispositions; generally.

1. Applicability. Except as otherwise specified, ... the court must impose a sentence prescribed by this section.

2. Revocable dispositions.
(a) The court may impose a revocable sentence as herein specified:
(i) the court, ... may sentence a person to a period of probation or to a period of conditional discharge ...; or
(ii) the court ... may sentence a person to a term of intermittent imprisonment....
(b) * * *
(c) In any case where the court imposes a sentence of probation, conditional discharge, or a sentence of intermittent imprisonment, it may also impose a fine authorized by article eighty.
(d) In any case where the court imposes a sentence of imprisonment not in excess of sixty days, for a misdemeanor or not in excess of six months for a felony or in the case of a sentence of intermittent imprisonment not in excess of four months, it may also impose a sentence of probation or conditional discharge ....

3. Other dispositions. When a person is not sentenced as specified in subdivision two,... the sentence of the court must be as follows:
(a) A term of imprisonment...; or
(b) A fine ...; or
(c) Both imprisonment and a fine; or
(d) Where authorized,... unconditional discharge....

Figure 1.

Adult Sentencing Parameters – BNF Syntax For Rewrite Rules

```
S ::= REV | IRREV
REV ::= SIMPREV | COMPREV
   COMPREV ::= fine + SIMPREV
   SIMPREV ::= PUREREV | SPLITREV
      PUREREV ::= int | REVLIB
      SPLITREV ::= def + REVLIB | int + REVLIB
         REVLIB ::= cd | p
IRREV ::= ud | fine | IMP | fine + IMP
   IMP ::= def | ind
```

Key to Terms

S-sentence. REV-revocable sentence. IRREV-irrevocable sentence. SIMPREV-simple revocable sentence. COMPREV-compound revocable sentence. PUREREV-pure revocable sentence. SPLITREV-split sentence. REVLIB-sentence of revocable liberty. IMP-sentence of imprisonment. fine-a fine. int-intermittent imprisonment. def-definite sentence of imprisonment. cd-conditional discharge. p-probation. ud-unconditional discharge. ind-indefinite term of imprisonment.

Figure 2.

According to New York law, a sentence (*S*) may be either revocable (*REV*) or irrevocable (*IRREV*). A revocable sentence may be simple (*SIMPREV*) or compound (*COMPREV*). A compound revocable sentence consists of a fine plus a simple revocable sentence (*fine + SIMPREV*), and so on, until one reaches the "atomic" terms of the sentence referring to fines (*fine*), probation (*p*), indeterminate imprisonment (*ind*), etc. This basic model generates in every case a list of all constructible sentences, without regard to their legality. The next step is to add predicates such as the *ok(X,Y)* predicate in **Listing 1** at appropriate points in the program to test and filter the output of the sentence generator against the legal restrictions, and to calculate various specific parameters of the sentence being generated, such as minimum and maximum periods of incarceration, probation, etc. The code in **Listing 1** depicts the general structure of the program, omitting clauses for *ok(X,Y)* and other filtering predicates.

```
sentence(S,R) :- rev(S,R).
sentence(S,R) :- irrev(S,R).

rev(S,R)    :- simprev(S,R).
rev(S,R)    :- comprev(S,R1),
               append(["PL 60.01(2)(c)"],R1,R).

comprev(S,R) :- simprev(S1,R1),
                concat(S1,"plus",S2),
                fine(S3,R2),concat(S2,S3,S),append(R1,R2,R).

simprev(S,R) :- purerev(S,R).
simprev(S,R) :- ok(splitrev,[]),splitrev(S,R1),
                append(["PL 60.01(2)(d)"],R1,R).

purerev(S,R) :- ok(intermittent,R1),
                intermittent(S,R2),append(R1,R2,R).
purerev(S,R) :- revlib(S,R).

splitrev(S,R) :- charge_is_a(felony),ok(definite,R0),
                 revlib(S1,R1),append(R0,R1,R),
     concat("Definite sentence up to 6 months plus ",S1,S).
splitrev(S,R) :- charge_is_a(misdemeanor),not(test(vtl1)),
                 revlib(S1,R),
     concat("Definite sentence up to 60 days plus ",S1,S).

revlib(S,R)  :- ok(cd,R1),cd(S,R2),append(R1,R2,R).
revlib(S,R)  :- ok(probation,R1),
                probation(S,R2),append(R1,R2,R).

irrev(S,R)   :- ok(ud,R1),ud(S,R2),append(R1,R2,R).
irrev(S,R)   :- ok(fine,R1),fine(S,R2),append(R1,R2,R).
irrev(S,R)   :- imprisonment(S,R1),
                append(["PL 60.01(3)(a)"],R1,R).
irrev(S,R)   :- imprisonment(S1,R1),fine(S3,R3),
                concat(S1," plus ",S2),concat(S2,S3,S),
                append(R1,["PL60.01(3)(c)"],R2),
                append(R2,R3,R).
```

Listing 1.

Detection of Lesser Included Offenses: A Problem in Searching Directed Graphs for Shortest Acyclic Paths

The problem of determining lesser and non-lesser included offenses involves the problem of finding the shortest acyclic path between two nodes in a directed graph. A directed graph may be defined as a set of nodes connected by links, each of which may be traversed in one direction only. A graph is represented visually as a collection of points, representing the nodes, connected by arrows, representing the links. (See Figure 3(A).) A graph may be represented in Prolog by a collection of statements of the form,

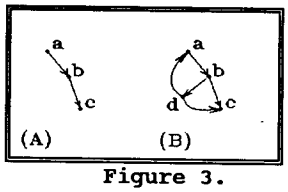

Figure 3.

imp(a,b). imp(b,c). imp(b,d). imp(d,c). imp(d,a).

where each such statement is to be interpreted as, "Node a implies node b", etc.

Such a structure is useful in the analysis of various legal domains, if the nodes are taken to represent atomic legal concepts. A directed link is drawn between two nodes whenever the legal relationship between their associated legal concepts is such that the presence of one necessarily implies the other. If a node, *a*, is considered to represent the legal element, "*dwelling*", the node *b* represents "*building*", and another node, *c*, represents "*premises*", the problems are (1) to recognize there is a connecting path between them, (*a-b-c*, representing *dwelling-->building-->premises*) (2) to construct or retrieve the path, (3) to select the shortest such path (rejecting paths such as *a-b-d-c*), (4) to avoid cyclic paths (*a-b-d-a-b-c, a-b-d-a-b-d-a-b-c, etc.*) (See Figure 3(B).)

```
/* Lesser Included Offenses */

lio(section(L),section(G),R) :-
  off(section(G),CG,EG),!,
  off(section(L),CL,EL),....,
  classorder(CL,CG),
  set_implies_set(EG,EL,PL),.....

/* Affirmative Set Predicates */

set_implies_set(_,[],[]).
set_implies_set
  (G,[H|T],[RH|RT]):-
  set_implies(G,H,RH),!,
  set_implies_set(G,T,RT).

set_implies(G,E,R):-
  findall(P,gpath(G,E,P),PL),
  minlist(PL,R).

gpath(G,E,P):-member(X,G),
        path(X,E,P).

path(A,Z,P):-path1(A,Z,[A],P).
path1(A,A,_,[l(A,A,yes,[])]).
path1
(A,Z,[A|P1],[l(A,X,yes,R)|P2]):-
  imp(yes,A,X,R),
  not(member(X,P1)),
  path1(X,Z,[X,A|P1],P2).

minlist([X],X):-!.
minlist([H|T],H):-
  minlist(T,M),
  listlen(H,HN),listlen(M,MN),
  HN<MN,!.
minlist([_|T],M):-minlist(T,M).
```

Listing 2.

The problems of recognizing and constructing a path are easily solved in Prolog. The additional condition of avoiding cycles in the graph adds only slight complications. A code fragment that constructs an acyclic path from any start node to any given goal node in a general graph connecting legal concepts is included in **Listing 2**, which is also an outline of the code used to compute lesser included offenses.

In the predicate *path(A,Z,P)*, *P* is a list of links between nodes each having the structure *l(Node1,Node2,Truthvalue,Rationale)*. Nodes such as *Node1, Node2* exist as arguments within the database of clauses *imp(Node1,Node2,Truthvalue,Rationale)*. The argument *Truthvalue* is here always assumed to be equal to the string "*yes*", indicating a positive implication between nodes. The extension to negative implications is given below. The argument *Rationale* is included to allow collection of explanatory text during path construction.

The predicate *gpath(G,E,P)* constructs through backtracking all paths from a given list or set of nodes, *G*, to a given node, *E*. A set of nodes, *G*, is said to "imply" a node, *E* along path *R* (*set_implies(G,E,R)*) if *R* exists and is the shortest of all paths from a node within *G* to *E*. The highest level predicate, *set_implies_set(G,L,R)*, succeeds if every element of *L* is "implied" by the set *G*, with *R* being the list of shortest paths from elements of *G* to each element of *L*. Offenses are stored in the structure (abbreviated here as) *off(section(Cite),Class,Elements)*. An offense cited as *section(L)* is a lesser included offense of an offense cited as *section(G)* if its classification *CL* is lesser in grade than that (*CG*) of *section(G)*, and if every member of the list *EL* of its elements is obtainable from the list *EG* of elements of *section(G)* according to the relation *set_implies_set(EG,EL,PL)*. An explanation for the result may be generated by examining the list of explanatory texts gathered in *PL*.

Nothing which has been said thus far deals with the general problem of finding the shortest path between two nodes. Algorithms for this purpose exist[**18**]; however, for this version of the program a naive brute-force method has been followed, namely that of simply computing all possible paths, and then selecting the shortest one. As the graph increases in complexity due to updates and upgrades to the program, it is likely that this method will have to be abandoned, to avoid combinatorial explosion in the number of possible paths which must be examined.

```
/* Non-Lesser Included Offenses */

n_lio(section(L),section(G),R):-
  off(section(G),CG,EG),!,
  off(section(L),CL,EL),
  classorder(CL,CG),
  set_n_implies_set(EG,EL,PL),....

/* Indifferent and Negative Set
Predicates */

set_n_implies_set(G,L,R):-
  set_i_implies_set(G,L,R),
  member(P,R),
  member(l(_,_,no,_),P).

set_i_implies_set(_,[],[]).
set_i_implies_set(G,[H|T],[RH|RT]):-
  set_i_implies(G,H,RH),!,
  set_i_implies_set(G,T,RT).

set_i_implies(G,E,R):-
  findall(P,gi_path(G,E,P),PL),
  minlist(PL,R).

gi_path(G,E,P):-member(X,G),
        i_path(X,E,P).

i_path(A,Z,P):-i_path1(A,Z,[A],P).
i_path1(A,A,_,[l(A,A,yes,[])]).
i_path1(A,Z,[A|P1],[l(A,X,T,R)|P2]):-
  imp(T,A,X,R),
  not(member(X,P1)),
  i_path1(X,Z,[X,A|P1],P2).
```

Listing 3.

The above concept may be easily extended, however, to include "negative" inferences, by utilizing the *Truthvalue* argument mentioned above. Each indicated link between nodes is now labeled according to whether it is true or false. (A link is considered "false" in this sense if there is affirmative legal authority to the effect that the implication between the given nodes does not follow.) The predicate *i_path(A,Z,P)* succeeds if there is a path *P* from node *A* to node

Z irrespective of the *Truthvalues* encountered along the way. *G_i_path(G,E,P)* finds such a path *P* from a member of the set of nodes *G* to the node *E*. *Set_i_implies(G,E,R)* succeeds if *R* exists and is the shortest such path. *Set_i_implies_set(G,L,R)* succeeds if every node in *L* is so connected to a node in *G* by a path in *R*. *Set_n_implies_set(G,L,R)* is now defined as succeeding if *set_i_implies_set(G,L,R)* succeeds, and there is a path *P* in *R* which contains a link having a *Truthvalue* of *"no"* (see **Listing 3**).

Selection of Assigned Counsel

We turn briefly to a problem involving ethics as opposed to substantive law, and consider the potential for systems associated with a database capable of advising on ethical questions.

An indigent defendant charged with a crime is entitled to be represented free of charge by an attorney assigned by the court. The usual procedure is to assign the public defender. Not infrequently, the public defender is precluded from undertaking the representation due to a conflict of interest, resulting, for example, from multiple defendants having adverse penal interests. In such cases the court must select an attorney from the community to undertake the defense. In a small county, most attorneys hold one or more public positions which may conflict with the performance of assigned criminal representation, either in all cases or under the circumstances of a particular case. The court must avoid assigning an attorney to the case who has an impermissible conflict of interest. In addition, even those attorneys not directly disqualified may be disqualified by association. The prevailing rule is that all partners or associates of an attorney are disqualified if the attorney is disqualified.**[19]** The court must accordingly consider, not only conflicting interests of the attorney it contemplates assigning, but those of his or her partners and associates as well. A Prolog program has been designed which assists in the analysis of this issue, and in the rotation of assignments among eligible attorneys in an evenhanded manner. Communication between the database system and this rule-based program is accomplished in the same manner as described above, the unique identifier of the attorney being passed as a parameter to and from the Prolog program. Portions of the Prolog code are presented in **Listing 4.**

Information pertaining to attorneys is stored in a structure (here abbreviated as) *attorney(Identifier,Attributelist)*. An attorney referenced by identifier *A* is considered a candidate for assignment in a case arising out of location *Loc* according to *possible(Loc,A)*, provided no disqualifying relation *dq(A,Loc,_)* is found. A general disqualification *general_dq(A,Exp)* may exist with explanation *Exp*, either because an element *X* of the attorney's attribute list *L* satisfies *holds_incompatible_office(L,Exp)*, or because this state of affairs exists with respect to another attorney *B* with respect to whom *associate_of(A,B)* is satisfied. Associates are identified as attorneys who each have the item *firm(F)* as a common element of their respective attribute lists. A disqualification *loc_dq(A,Loc,Exp)* due to the circumstances of a given case can occur, where the case arises out of or in some way involves a locality *Loc*, and the attribute list of attorney *A* indicates he is a municipal attorney *ma(Loc)* or a local part time judge *judge(Loc)* for that locality. As before, locality based disqualifications propagate to all members of an attorneys firm. Further extensions are easily imagined.

```
possible(Loc,A):-attorney(A,_),not(dq(A,Loc,_)).

dq(A,_,Exp)  :- general_dq(A,Exp).
dq(A,Loc,Exp):-location_dq(A,Loc,Exp).

general_dq(A,Exp):-attorney(A,L), holds_incompatible_office(L,Exp1),
                   concat(A,Exp1,Exp).
general_dq(A,Exp):-attorney(A,_), associate_of(A,B),
                    attorney(B,L),
                    holds_incompatible_office(L,Exp4),
  concat(A," is disqualified because associated with ",Exp1),
  concat(Exp1,B,Exp2), concat(Exp2,", who",Exp3),
  concat(Exp3,Exp4,Exp).

holds_incompatible_office(L,Exp):- member(X,L),
  incompatible_office(X,Exp1),
  concat(" is disqualified because ",Exp1,Exp).

incompatible_office(da, "a member of the District Attorney's office.").
incompatible_office(pd, "a member of the Public Defender's office.").
incompatible_office(ca, "a member of the County Attorney's office.").
incompatible_office(leg,"a member of the County Legislature.").
incompatible_office(judge("Greene County Supreme Court"),
                    "a Judge of the Supreme Court.").
incompatible_office(judge("Greene County Court"),"a Judge of County Court.").

location_dq(A,C,Exp):-location_dq2(A,C,Exp1),
  concat(A," is disqualified, being ",Exp2),
  concat(Exp2,Exp1,Exp3), concat(Exp3,", and the case involving ",Exp4),
  concat(Exp4,C,Exp5),    concat(Exp5,".",Exp).
location_dq(A,C,Exp):-associate_of(A,B),
                      location_dq2(B,C,Exp1),
  concat(A," is disqualified, being associated with ",Exp2),
  concat(Exp2,B,Exp3),    concat(Exp3,", who is ",Exp4),
  concat(Exp4,Exp1,Exp5), concat(Exp5,", and the case involves ",Exp6),
  concat(Exp6,C,Exp7),    concat(Exp7,".",Exp).

location_dq2(A,Loc,Exp):-attorney(A,L),
                         member(X,L), location_dq3(Loc,X,Exp).

location_dq3(Loc,judge(Loc),Exp):-concat("local judge for ",Loc,Exp).
location_dq3(Loc,  ma(Loc),Exp):-concat("municipal attorney for ",Loc,Exp).

attfirm(A,F):-attorney(A,L),member(firm(F),L).

associate_of(X,Y):-attfirm(X,F),
                   attfirm(Y,F),not(X=Y).
```

Listing 4.

Conclusion

Two examples of substantive rule-based extensions to conventional legal database systems have been shown. The applications were developed independently of the database system and of each other; communication between the applications and the database system is accomplished by passing required parameters via the command line or text files. The potential uses of substantive systems in the Courts are just beginning to be explored.

David E. Woodin, Esq. is a member of the bar of the State of New York. He is presently Associate Law Assistant, Trial Part, to the County Court of Greene County, New York, where he has served the New York State Unified Court System since 1982. He is also a freelance programmer and consultant, and sole proprietor of Due Process Software. He holds the degrees of J.D. from Albany Law School, M.A.T. in science and mathematics from Reed College, and B.A. in physics and music from Williams College.

References

[1]. Stout, Ronald M. and Seward, Ronald G. (1985) "Microcomputers: Information Managers in the Courts," 10 *The Justice System Journal* 97.

[2]. *See, e.g.*, Mulder, Richard V. and Gubby, Helen M., "Legal Decision Making By Computer: An Experiment With Sentencing", IV *Computer/Law Journal* 243 (1983). Pethe, Vishwas P., Rippey, Charles P. and Kale, L.V., "A Specialized Expert System For Judicial Decision Support", *Proceedings of the Second International Conference on Artificial Intelligence and Law*, p. 190. Simon, Eric, "ASSYST - Computer Support for Guideline Sentencing", *Proceedings of the Second International Conference on Artificial Intelligence and Law*, p. 195.

[3]. *See* Kerschberg, Larry, *Expert Database Systems, Proceedings From the First International Workshop*, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California (1986), Preface, p. iv.

[4]. See, Susskind, *Expert Systems in Law, A Jurisprudential Inquiry*, Clarendon Press, Oxford (1987); Gardner, *An Artificial Intelligence Approach to Legal Reasoning*, The M.I.T. Press, Cambridge (1987). Ciampi, "Artificial Intelligence and Legal Information Systems", in *Artificial Intelligence and Legal Information Systems, Volume I*, C. Ciampi (ed.), North-Holland Publishing Co., 1982, p. 49; Finan, "Lawgical: Jurisprudential and Logical Considerations", 15:4 *Akron Law Review* 675 (Spring, 1982); Meldman, "A Structural Model for Computer-Aided Legal Analysis", 6 *Rutgers Computers & Law* 27 (1971); Frank, *Courts on Trial*, Princeton, Princeton University Press (1949), pp. 206-208.

[5]. See, Tito, "Artificial Intelligence: Can Computers Understand Why Two Legal Cases Are Similar?", 7 *Computer/Law Journal* 409 (1987).

[6]. Susskind, *Expert Systems in Law, A Jurisprudential Inquiry*, Clarendon Press, Oxford (1987); Gardner, *An Artificial Intelligence Approach to Legal Reasoning*, The M.I.T. Press, Cambridge (1987).

[7]. See, e.g., *People v. Maderic*, 142 AD2d 892, 893 (3d Dept., 1989).

[8]. See New York State Criminal Procedure Law (CPL) 390.20, 390.30, 390.40, 400.10. New York State Penal Law (PL) Articles 60-85; *see esp.*, PL 60.01, 60.05, 60.11, 70.02-70.10.

[9]. CPL 220.10, 220.30.

[10]. *People v. Glover*, 57 NY2d 61; *People v. Green*, 56 NY2d 427.

[11]. CPL 100.40(1)(c), 200.50(7)(a).

[12]. CPL 170.35(1)(a), 210.20(1)(a), 210.25(1).

[13]. CPL 210.20(1)(a), 210.25(1)). *People v. Bartley*, 47 NY2d 965; *People v. Maderic*, 142 AD2d 892; *People v. Cook*, 93 AD2d 942; *People v. Hicks*, 79 AD2d 887; *Matter of Wadsworth v. Mogavero*, 71 AD2d 157. *People ex rel Gray v. Tekben*, 86 AD2d 176, *affd.* 57 NY2d 651, *see People v. Williams*, 95 AD2d 726. CPL 440.20(1), 440.40(1). CPL 470.15(2)(c), 470.45.

[14]. *™ Borland International*, 1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95066-0001.

[15]. *Gunga Clerk, version 1.2.* Copyright (C) 1988, 1989, *Due Process Software*, 340 Main Street, P.O. Box 433, Catskill, New York, 12414.

[16]. *™ Nantucket Corporation*, 12555 W. Jefferson Blvd., Suite 300, Los Angeles, CA 90066.

[17]. Koers, A.W., et al., *Knowledge Based Systems in Law*, Kluwer Law and Taxation Publishers, Deventer, The Netherlands, 1989, pp 58-68.

[18]. See Bratko, Ivan, *Prolog Programming for Artificial Intelligence*, Addison-Wesley (1986), pp. 262-273.

[19]. *A.B.A. Code of Professional Responsibility*, DR 5-105(D). "If a lawyer is required to decline employment or to withdraw from employment... no partner or associate of his or his firm may accept or continue such employment."