An Issue-Oriented Approach to Judicial Document Assembly[1]

L. Karl Branting
Department of Computer Science
University of Wyoming
Laramie, Wyoming 82071-3682
karl@eolus.uwyo.edu

1 Introduction

One of the most pervasive problems confronting the judicial system in the United States during recent decades has been the delay and congestion of court dockets resulting from the explosive growth in case filings [Sne89]. The volume of cases filed in U.S. state courts reached nearly 100 million in 1989 [RO91]. Computer programs to enable judges to use their time and expertise more efficiently can therefore potentially make a significant contribution to the judiciary.

One task that lends itself well to automation is assembling judicial opinions, decisions, and orders. Automating the mechanics of constructing judicial documents can free judges to direct more of their attention to the central judicial functions of evaluating the credibility of evidence and interpreting statutes and precedents.

2 Two Approaches to Automated Judicial Document Assembly

Two approaches to judicial document assembly can be distinguished. Conventional document assembly programs use a *document-oriented* approach. The focus of these systems is typically on automating the selection of document components, such as paragraphs and clauses, and instantiating user-provided values for variables embedded in those components [Lau92]. The problem-solving method of these systems can be characterized as *hierarchical refinement:* Such systems typically contain a separate template for each document type together with a set of rules for instantiating the template. Elements required to instantiate a template may themselves be templates, *e.g.,* "sub-documents" in CAPS and "submodels" in Scrivenir [Lau92].

However, judicial decisions and orders have an important characteristic that other legal documents in general lack: they express a justification for a legal conclusion in terms of a set of legal rules and findings of fact from which the legal conclusion follows. *Issue-oriented* document assembly is an alternative approach better suited for such documents. The issue-oriented approach makes use of an explicit representation of the legal rules under which a decision of a particular type can be rendered. Text is associated with assignments of truth values to the legal predicates occurring in the rules. The system uses the rules to build a justification reflecting a judge's rulings on each issue relevant to the ultimate decision. A document is generated by assembling the text associated with the truth value assignment of each legal predicate occurring in the justification.

There are four advantages of the issue-oriented approach to judicial document assembly. The first is that issue-oriented systems are more flexible because they permit documents reflecting various combinations of issues to be generated dynamically. Document-oriented systems, by contrast, typically

[1] Support for this research was provided by a grant from the National Center for Automated Information Retrieval. Extensions to XLISP used in the implementation of LawClerk, including the entire graphical interface, and portions of LawClerk's code were written by Patrick Broos.

require combinations of issues to be anticipated, because these systems generally lack a model of the operative legal rules and therefore cannot model the dependencies among legal predicates that determine valid combinations of issues.

A second advantage is that explicit representation of legal rules permits an issue-oriented judicial document assembly program to function as a decision support system. Simulating the judge's decision process permits such a system to insure that (1) the judge rules on all and only the issues relevant to the case and (2) the judge's decision refers to all relevant evidence.

Third, declarative representation of legal rules permits the system to explain, in terms of the overall goal of resolving conflicting claims, both why it is eliciting particular information from a judge and how it reached a given conclusion.

Finally, issue-oriented systems are easier to maintain, because accommodating a change in the law requires changing only the affected legal rules and their associated text rather than an entire decision template.[2]

3 Control Strategies for Issue-Oriented Judicial Document Assembly

The issue-oriented approach to judicial document assembly uses an explicit representation of legal rules to construct a justification consistent with the judge's ruling, which is then used to create a judicial document. The central design decisions in such a system therefore concern (1) how the justification is constructed and (2) how the document is assembled from the text associated with the justification.

3.1 Constructing Justifications

Constructing a justification for a legal decision from legal rules is similar to the task performed by rule-based expert systems. It differs primarily in that the purpose of constructing a justification is to model the judge's decision rather than making an independent judgment. The justification-constructing portion of an issue-oriented judicial document assembly system (hereinafter the *justifier*) must therefore elicit the judge's ruling on each issue relevant to the ultimate decision.

Perhaps the simplest approach to constructing a justification is to use the same depth-first, backchaining, satisficing search strategy used in the PROLOG programming language, querying the user when a goal is reached for which there are no applicable legal rules. By requiring the judge to rule on the open-textured predicates that appear as subgoals, this strategy (1) insures that the judge rules on every issue necessary to support the ultimate decision in the case, while at the same time (2) deferring to the judge on every issue involving evaluation of credibility or interpretation of legal authorities.

However, there are two situations in which this strategy requires modification. First, an important form of decision support that an issue-oriented judicial document assembly system (JDAS) can provide is insuring that the judge's decision refers to all relevant evidence. This requires that the system have some model of the connection between evidence and legal predicates. In LawClerk, described below, this connection is expressed by *prima facie* rules. When the judge is queried about a legal predicate, LawClerk informs the judge what evidence in the case record tends to establish and what evidence tends to negate the predicate. Finding all evidence that either supports or negates a predicate requires an exhaustive search strategy using *prima-facie* rules.

Second, the use of a depth-first search strategy for legal rules assumes a model of judicial parsimony in which a legal issue should be resolved on the basis of exactly one legal rule. Once a rule has been found that satisfies a predicate, it is unnecessary to consider whether alternative rules might lead to the same result. However, in many trial courts and administrative agencies it is not unusual for decisions to be rendered on multiple grounds. Modeling a decision rendered on multiple grounds requires an exhaustive, rather than a satisficing, search. Thus, the most appropriate control strategy for the justifier may depend on the conventions of the judicial body for which the JDAS is intended.

[2] It should be noted that CAPS and Scrivenir provide command languages capable of expressing complex legal rules and have some explanation capability, and Scrivenir uses a declarative representation of legal rules [Lau92]. Thus, notwithstanding that most CAPS and Scrivenir applications have followed a document-oriented approach, issue-oriented systems could presumably be implemented within these environments.

3.2 Mapping Justifications to Documents

A second issue in issue-oriented JDAS's concerns the process of creating a judicial document from a model of the judge's justification for the decision. This process may be either incremental, with the text associated with an issue added to the document as the issue is decided, or nonincremental, with the construction of the document beginning only after the justification has been completed.

The simplest incremental approach is to append to the document the text associated with each ruling by the judge as the ruling is made. The disadvantage of this *single destination* approach is that a ruling on a single issue may have effects in several different parts of an opinion, *e.g.*, both in the Conclusions of Law and in the Decision section. Alternatively, a *multiple destination* approach uses an opinion template representing the sections of an opinion into which multiple text items associated with a single legal issue can be written. As the judge rules on each issue, text associated with the ruling is appended to each affected section of the opinion.

However, the effectiveness of both incremental approaches depends on two properties of the judicial documents being produced. The first is *issue locality*, the property that text associated with the resolution of one legal issue is independent of the text associated with any issue addressed later in the decision. The second is that the order in which legal issues are addressed by the justifier is the same as the order in which issues should appear in each section of the document.

If the target documents lack issue locality or if the justifier doesn't address legal issues in the order in which they should appear in the document, a nonincremental approach permitting use of complex mappings from justifications to documents is necessary. One approach to creating documents from justification structures is described in [Bra91a].

4 Related Research

A prototype issue-oriented judicial document assembly program, JEDA, was developed by Vishwas Pethe, Judge Charles P. Rippey, and L. V. Kale [PRK89]. JEDA uses PROLOG's depth-first, satisficing search strategy and performs incremental mapping from justifications to text. JEDA uses mixed-initiative data entry: uncontested facts (*e.g.,* names the parties, docket number) and allegations concerning a case (*e.g.*, medical reports, test results) are entered by filling in a series of entry forms. During a consultation, the justifier solicits rulings from the judge concerning legal predicates occurring as subgoals.

Judge Rippey, an Administrative Law Judge at the U.S. Department of Labor who has used JEDA since 1987, reports that JEDA reduces the time necessary to write decisions on claims for benefits under the federal Black Lung Benefits Act by a factor of at least two [PRK89]. Unfortunately, JEDA has not been accepted by other ALJ's on Judge Rippey's division [Rip91].

A key limitation of JEDA is its inability to accommodate changes in the law or to apply to domains other than Black Lung Benefits Act cases. JEDA's knowledge of statutory rules, precedents, and boilerplate text are hardcoded and cannot be changed without rewriting the program itself. Moreover, JEDA's case format, record types, rules, and boilerplate language are all tailored exclusively for cases arising under the Black Lung Benefits Act. As a result, JEDA cannot be applied to cases of any other type without extensively rewriting program code.

5 LawClerk: An Issue-Oriented JDAS Shell

LawClerk is an issue-oriented JDAS shell under development at the University of Wyoming designed to address the limitations of JEDA. LawClerk uses a declarative representation of rules, issue templates, and entry forms that permits it apply to a variety of different domains and to accommodate easily changes in the law or in the idiom of a particular judge or document type.

The initial implementation of LawClerk (version 1.0) uses depth-first, satisficing search with legal rules and exhaustive search with *prima-facie* rules and multiple-destination incremental mapping from truth-value assignments to document sections. Data entry is mixed-initiative: information other than rulings is entered by filling in entry forms, and rulings are obtained interactively. LawClerk version 1.0 is written in XLISP, a compact public-domain LISP dialect implemented in C. XLISP's interface was extended using Borland's

C++ TurboVision library to support pop-up menus, dialog boxes, editing windows, and mouse-sensitive push-buttons.

The Food Stamp Fraud Consultant

The behavior of LawClerk 1.0 can be illustrated by the Food Stamp Fraud Consultant, a prototype JDAS for Food Stamp Fraud cases implemented using LawClerk 1.0 and developed in conjunction with the Colorado Division of Administrative Hearings.

5.1 Food Stamp Fraud

Under the Code of Colorado Regulations 12CR8 §B-4425, Colorado food stamp offices are responsible for investigating cases of alleged fraud in obtaining food-stamp benefits and for initiating administrative hearings on such cases. A food stamp recipient who is shown to have committed an "intentional program violation," *i.e.,* made a "false or misleading statement, or misrepresented, concealed or withheld facts" may be disqualified from receiving additional food stamp benefits for 6 months, 12 months, or permanently depending on whether the recipient has had 0, 1, or 2 previous disqualifications.

5.2 Entry of Case Information

Interaction with the Food Stamp Fraud Consultant is in two phases. In the first phase, information other than legal conclusions is entered by filling in values for fields in entry forms of the following types:

- Case caption
- Food stamp application
- Allegations
- Admissions
- Denials
- Hearing notice
- Hearing

Only the case caption and hearing forms are required to be filled out. A set of completed entry forms for a given case constitutes the *case record.*

Because it does not involve legal conclusions, this information can be entered by clerical personnel. All attributes of the entry forms, including the number and type of forms and the number, placement, datatype, and label of every field in each form, are represented declaratively and can therefore be easily modified with a form editor.

5.3 Consultation with Judge

The second phase of interaction with the Food Stamp Fraud Consultant is the consultation with the judge, during which the program prompts the judge to rule on each issue, *i.e.,* assign a truth value to each legal predicate, relevant to the outcome of the case. A legal predicate is deemed relevant if it appears as an antecedent of a rule that applies to the ultimate issue in the case (*e.g.*, disqualification from food stamp benefits) or if it is an antecedent of a rule applying to a relevant predicate. When the Food Stamp Fraud Consultant is told or infers that a predicate is true or false, it retrieves a text template associated with a truth-value assignment for that predicate, instantiates it, and presents it to the user for editing. When the text satisfies the user, it is added to the appropriate section of the opinion under construction. After the Food Stamp Fraud Consultant has determined the truth value of all relevant facts, it presents a draft of the final opinion to the user for a final edit.

The consultation begins with the user selecting from a menu the ultimate issue in the case.[3] Associated with the ultimate issue is a set of initial *actions*, *i.e.,* template/case-section pairs. The initial actions contain templates for the portions of each section of the opinion that are independent of the outcome of the case. For example, one initial action is to instantiate a template stating whether the parties were represented by counsel and writing the instantiated template to the prologue of the opinion.

LawClerk then backchains through its legal rules for disqualification. The top-level rule for disqualification is as follows:

```
#s(legal-rule
    consequent (disqualification
```

[3] Currently, only one ultimate issue is implemented in the Food Stamp Fraud Advisor, (disqualification ?person ?period ?amount), *i.e.,* ?person is disqualified from receiving food stamps for ?period and is liable for ?amount in overissuance.

```
                         ?person
                         ?period
                         ?amount)
    antecedents ((intentional-program-violation
                         ?person)
                 (disqualification-period
                         ?person
                         ?period)
                 (fs-overissuance
                         ?person
                         ?fso
                         ?amount))
    citation "Section B-4425.32")
```

This rule expresses the requirement that to establish disqualification from food-stamp benefits, one must establish that an intentional program violation was committed and determine the applicable disqualification period and the value of the food-stamp overissuance. Under this rule, the top-level goal of establishing (disqualification Jones ?period ?amount) gives rise to the subgoal of establishing that Jones committed an intentional program violation. The following rule expresses the requirements for establishing this subgoal:

```
#s(legal-rule
    consequent (intentional-program-violation
                         ?person)
    antecedents ((applied-for-benefits
                         ?person)
                 (certified-to-receive-benefits
                         ?person)
                 (misrep-or-omit
                         ?statement
                         ?person
                         ?misrep-type)
                 (intentional
                         ?statement
                         ?person
                         ?misrep-type)
                 (received-penalty-notice
                         ?person)
    citation "Section B-4425.1.1")
```

Under this rule, establishing an intentional program violation requires establishing that a person applied for and was certified to receive food stamp benefits, that the person intentionally misrepresented or omitted information, and that the food stamp application contained a notice informing the person of the penalties of misrepresentations or omissions.

When LawClerk reaches a goal to which no legal rules apply, it searchs for *prima facie* rules having either the goal or its negation as their consequent. *Prima facie* rules express an evidentiary relationship between case facts and the legal predicates they tend to establish or negate. For example, the following *prima facie* rules express the tendency of allegations of unreported employment to confirm, and denials of unreported employment to negate, the existence of unreported employment:

```
#s(prima-facie-rule
      antecedents
          ((access '(allegations
                        unreported-employment
                        employer)
          *current-case-record*))
      consequent (unreported-employment
                        ?person))
#s(prima-facie-rule
      antecedents
          ((access '(response
                        denial
                        unreported-employment)
          *current-case-record*))
      consequent (not (unreported-employment
                        ?person)))
```

After determining the applicable *prima facie* rules, LawClerk asks the user to rule on the issue, presenting the supporting or negating evidence identified through the *prima facie* rules, *e.g.,*:

> Is it the case that Jones had unreported employment income?
>
> Supporting evidence: The Denver Food Stamp Fraud Office alleged that Jones had unreported employment income.
>
> Negating evidence: Jones denied having unreported employment income.

This use of *prima facie* rules insures that the judge will be aware of any evidence in the case record relevant to the current legal issue.

When the judge is asked to rule on the truth of a legal predicate (as opposed to supplying a date, amount of money, or other value) the user is presented with the choice of ruling "Yes," "No," or asking "Why?" If the "Why?" option is chosen, LawClerk responds by identifying the immediately superior goal. For example, if the user is asked whether Jones applied for food stamp benefits and the user asks "Why?" LawClerk prints the following explanation:

> [I.e., Why I am I trying to determine whether Jones applied for food stamp benefits?] I'm trying to determine whether Jones committed an intentional program violation. This conclusion would follow under Section B-4425.1.1 from the following:
>
> Jones applied for food stamp benefits.
>
> Jones made a false or misleading statement, or misrepresented, concealed, or withheld facts.
>
> ...

Repeated invocations of "Why?" will access successively higher goals until the top-level goal is reached.

When the judge rules on an issue, the actions corresponding to the truth value assignment of the predicate are retrieved, and the template in each action is instantiated, presented to the user for editing, and written to the appropriate section. Templates in LawClerk 1.0 are lists containing strings, references to the case record, or Lisp expressions that evaluate to strings. For example, a portion of the template for the predicate "applied-for-benefits" is:

```
(<cr> <cr><np>". "
"On "
(datelist-to-string
     (access '(fs-application fs-application-dates)
              *current-case-record*))
(caption respondent abbreviated-name)
" applied for FS benefits for a household"
" consisting of "
(fs-application number-of-family-members)
" family members. "
```

Hard returns and paragraph numbers are indicated by <cr> and <np>, respectively. The expression (caption respondent abbreviated-name) refers to the abbreviated-name subfield of the respondent field of the caption form. LawClerk's template language is currently being revised to be more comprehensible to technically naive users.

LawClerk's model of a judge's justification for a decision is a *goal tree* [BS84], consisting of the rule invocations and judicial rulings that justify the outcome of the case, constructed in the process of backchaining. The syntax for the justification is a simplified version of the explanation syntax used in GREBE [Bra91b]:

```
<explanation> ::= ( <goal> ruling) |
                  ( <goal>
                       <rule-name>
                       <explanation>*) |
                  ( <function-tuple>
                       has-value
                       <value>) |
                  ( <function-tuple>
                       succeeded) |
                  ( (unless <goal>)
                       failed
                       <goal>)
<goal> ::= <tuple> | not(<tuple>)
<tuple> ::= ( <predicate> <symbol>+ )
<function-tuple> ::= ( <function-name>
                       <Lisp-expression>* )
<value> ::= <symbol> | <number>
```

The simplest explanation, "(<goal> ruling)," merely indicates that the judge ruled that <goal> is true. This explanation does not attempt to represent the factors that the judge used to justify this ruling. In explanations of the second type, (<goal> <rule-name><explanation>*), <goal> follows from rule <rule-name> given the explanations for <rule-name>'s antecedents.

Since LawClerk performs incremental mapping from truth-value assignments to document sections, creation of a static goal tree is not necessary for the construction of the judicial document. However, the goal tree permits the system to provide answers to "Why?" and "How?" questions by the user.[4]

6 The Pragmatics of Issue-Oriented JDAS's

The development of LawClerk was guided by lengthy discussions with administrative law judges, staff attorneys, and judges, primarily at the Colorado Court of Appeals (where the author was formerly employed) and the Colorado Division of Administrative Hearings. In the course of these discussions, and as a result of an initial evaluation of

[4] Only "Why?" questions are currently implemented in LawClerk 1.0.

the Food Stamp Fraud Consultant by the ALJ's of the Colorado Division of Administrative Hearings, several conclusions emerged.

First, no issue-oriented JDAS is likely to be widely accepted unless it is capable of being easily updated to reflect changes in law or institutional custom. Support personnel in administrative agencies and trial courts typically have limited technical experience, so the process of updating the JDAS must be understandable even to a technically naive user.

Second, it is nontrivial in practice to determine a class of judicial documents amenable to issue-oriented JDAS's within a given decisional body. Characteristics of suitable documents include the following:

- A predictable set of issues. An issue-oriented JDAS is feasible only if the issues that can occur in a case can be preenumerated and text templates associated with each.

- A high volume of cases. Development of an issue-oriented JDAS is economical only if the development costs can be amortized across a large number of cases.

- Variable combinations of issues. An issue-oriented JDAS is useful only if there is sufficient variability in the combinations of issues arising in cases that simpler document-oriented techniques are impractical.

Appellate court decisions are poorly suited to issue-oriented JDAS's because the issues that arise on appeal tend to be very unpredictable. Highly routine orders are also unsuited to issue-oriented JDAS's when there is insufficient variability to justify the overhead of modeling the decisional process. By contrast, administrative decisions are often well suited to issue-oriented JDAS's, as illustrated by the experience of JEDA. Similarly, orders and decrees in courts of general jurisdiction and appellate courts are well-suited to the extent that they involve permutations of a predictable set of issues. However, considerable analysis and negotiation is generally necessary in practice to determine a class of judicial documents having the right balance between predictability (of issues arising in most cases) and variability (of combinations of issues occurring in any given case) and sufficiently high case volume to justify development of a JDAS.

A third issue, discussed above, is that some decisional bodies adhere to a policy of strict judicial parsimony under which a single legal issue is resolved on the basis of exactly one legal rule, whereas others permit legal issues to be resolved on multiple grounds. Whether the control strategy of the justifier in applying legal rules should be exhaustive search or satisficing search depends on this choice of legal idiom. A general-purpose shell should be capable of supporting both strategies.

Fourth, JEDA and LawClerk employ a mixed-initiative data entry scheme under which information other than rulings can be provided in data entry forms, and rulings are obtained interactively. However, some ALJ's have expressed a preference for an interface in which all information, including factual findings and rulings, is entered in a single form after which the JDAS assembles the opinion noninteractively. Once again, general-purpose shell should provide support for both approaches.

Status of the LawClerk Project

In view of the results of the preliminary evaluation, LawClerk is currently undergoing the following revisions:

- Revision of the justifier's control strategy to accommodate both parsimonious and multiple-ground decision styles.

- Provision for both mixed-initiative and non-interactive justifier modes.

- Improved editing capability. LawClerk currently includes an editor for legal and prima-facie rules, and an editor for text templates associated with legal predicates. However, neither is yet sufficiently self-explanatory to be usable by a technically naive judicial staff technician. In addition, an editor for entry forms has yet to be developed, although the declarative representation of these forms makes devising such an editor straightforward.

- Extension of the justifier's explanation capability, which currently includes presenting all arguments for and against a given issue and answering "Why?" questions, to include "How?" and "Why not?" question types.

- Addition of a facility to permit the user to retract or change actions.

The revised version of LawClerk will be tested in Colorado Division of Administrative Hearings, the Wyoming Office of Administrative Hearings, or one or more Wyoming District Courts

7 Conclusion

This paper has described an issue-oriented approach to judicial document assembly that uses an explicit representation of legal rules to construct a justification consistent with the judge's ruling which is then used to create a judicial document. LawClerk is a domain-independent issue-oriented JDAS shell that uses a declarative representation declarative representation of rules, issue templates, and entry forms that permits it apply to a variety of different domains and to accommodate easily changes in the law or in the idiom of a particular judge or document type. A preliminary evaluation of the Food Stamp Fraud Consultant, an application implemented in LawClerk, indicates that general purpose JDAS's require considerable flexibility in control strategy and data entry modes and must be capable of being updated by technically naive support personnel.

References

[Bra91a] L. K. Branting. Building explanations from rules and structured cases. *International Journal of Man-Machine Studies*, 34:797–837, 1991.

[Bra91b] L. K. Branting. *Integrating Rules and Precedents for Classification and Explanation: Automating Legal Analysis*. PhD thesis, University of Texas at Austin, 1991.

[BS84] B. Buchanan and E. Shortliffe. *Rule-Based Expert Systems*. Addison-Wesley Publishing Co., Menlo Park, 1984.

[Lau92] M. Lauritsen. Tevchnology report: Building legal practice systems with today's commericial authoring tools. *Law and Artificial Intelligence*, 1(1), 1992.

[PRK89] V. P. Pethe, C. P. Rippey, and L. V. Kale. A specialized expert system for judicial decision support. In *Proceedings of the Second International Conference on Artificial Intelligence and Law*, pages 190–194, Vancouver, B.C., June 13-16 1989.

[Rip91] Judge Charles P. Rippey, 1991. Personal communication.

[RO91] David B. Rottman and Brian J. Ostrom. Caseloads in the state courts. *State Court Journal*, 15(2), Spring 1991.

[Sne89] Aidney C. Snellenburg. New approaches to reducing court delay and congestion. *State Court Journal*, 13(3), summer 1989.