Knowing Documents

Marc Lauritsen
Harvard Law School
Cambridge, Massachusetts 02138
LAURIT@HULAW1.HARVARD.EDU

Abstract

Drawing upon scholarship on legal drafting, current document assembly technology, and aspects of the Standard Generalized Markup Language, this article discusses the forms of knowledge at play in the creation of legal documents. It also examines the notion of self-describing documents and their potential role in new modes of expressing and delivering knowledge pertinent to legal drafting.

1. Introduction

An experienced attorney sits down to draft an employment agreement for the new treasurer of a mid-sized business. She has ascertained from the client employer what terms and provisions are desired in general. With the aid of her notes, some reference books, agreements in prior matters, a word processor, and several hours of deliberation, the attorney completes a suitable draft. After the client and counsel for the employee review the document, prompting some modest revisions, the agreement is signed and goes into effect.

What knowledge was brought to bear by the lawyer in the creation of this document? What did she know about documents in general, about legal documents, about contracts, about employment contracts? What can be known about such things? Are the things that can be known about document drafting meaningfully or usefully distinguishable from other forms of legal knowledge?

Computer-based practice systems are increasingly being used by lawyers to draft documents like this employment agreement. They are typically the product of one or more legal experts, and embody aspects of their knowledge. In what ways do these programs express and communicate knowledge? How do we begin to describe what these practice systems "know"? What informational work, or intellectual labor, is being done when the systems are in operation? What cognitive resources (information, knowledge, intelligence) fuel those processes?

Research and scholarship in artificial intelligence and law have largely focussed on either the retrieval and analysis of documents already in existence or the processes of reasoning that precede the commitment of decisions, promises, claims, and other legal actions to written form. With a few exceptions, noted later, the AI-and-law community has paid little heed to the activity of document drafting per se.

By the same token, although a mature legal document assembly software industry has emerged, the increased expressive power of its offerings has not engendered systematic attention to the knowledge dimensions of the drafting process. Idiosyncratic and ad hoc representation schemes abound in today's products.

The explorations contained in this article are motivated by the author's belief that our understanding of legal cognition and the utility of our computer-based practice tools would both be advanced by more explicit attention to the knowledge dynamics of everyday lawyering. After briefly reviewing some legal drafting literature, today's document assembly technology, and techniques of generalized document markup, I offer some observations about "document knowledge" and "knowledgeable documents".

2. The art of legal drafting

One is tempted to think that the actual creation of a legal document is a ministerial act, the simple writing down of things already decided. To be sure, one might say, lawyers think while they're writing, but the writing is secondary to the substantive legal analysis or planning tasks more properly at the center of lawyerly attention. A related point of view, somewhat at odds with the foregoing, is that legal writing—beyond certain trivial dimensions of format and orthography—entails processes of creativity and imagination that are too sublime or author-specific to be captured in any general formulae or finite set of rules.

Although never given much academic attention, there is a substantial body of work dedicated to counteracting the above attitudes. One of the finest contributors to the legal drafting literature was the late Reed Dickerson. His *Fundamentals of Legal Drafting* [1986] is a wonderful collection of insights and information about this essential, albeit underinvestigated, lawyering activity.

As Dickerson defines it, "legal drafting is the crystallization and expression in definitive form of a legal right, privilege, duty, status, or disposition. It is the development and preparation of legal instruments such as constitutions, statutes, regulations, ordinances, contracts, wills, conveyances, indentures, trusts, and leases." (p.3) Drafting is thus more concerned with instruments that have a significant "architecture" than with less structured forms of legal writing, such as briefs, memoranda, and letters.

Legal drafting theory concerns itself with the qualities that characterize well-formed legal instruments, and derivatively, with the hallmarks of sound drafting practice. In the first category fall such desiderata as legal and factual correctness, completeness, consistency, simplicity, clarity, and readability. Well-drafted documents are free of errors and are calculated to achieve their intended effect. Stylistically, they avoid verbosity, redundancy, jargon, vagueness, cliches, archaic expressions, and legalisms. Good documents are not boring, colorless, overwritten, or filled with hedging terms, frequent use of the passive voice, or overly long sentences. They observe sound principals of arrangement.

Drafting theorists offer process criteria as well as those descriptive of the end result. Good drafting practice entails careful analysis and planning, thorough research, systematic consideration of alternatives, and punctilious proof reading.

Surprisingly few articles deal with the creative use of computers in the legal drafting context. Layman Allen's and Charles Saxon's work on normalized drafting has encompassed computer programs for eliciting and generating multiple structural interpretations of legal rules. [Allen and Saxon, 1991]. Eve Wilson's article [Wilson, 1989] about the Justus' Clerk program is written from the drafting perspective, paying particular attention to the role of precedents in legal composition.

3. Document assembly technology

It has been well over a decade since Jim Sprowl [Sprowl, 1979], Larry Farmer, and others (e.g., Saxon, 1981) undertook their pioneering work in legal document assembly. Building on those conceptual foundations, and exploiting the intervening explosion of desktop computing power, law office software vendors have fielded several dozen products with document assembly functions. These general developments, and two specific products, are examined in [Lauritsen, 1992]. (The same article also reviews the rather sparse history of document assembly in AI-and-law circles.)

The present generation of document assemblers provides a diversity of ways to perform the basic functions of building templates, eliciting information from a user, and assembling documents. The minimal feature set for a serious software contender in this field has grown considerably, and the high-end products afford means for resourceful authors to store and communicate a wide range of knowledge about documents.

Despite considerable successes, today's document assembly software falls short of its potential on several counts. As Thomas Gordon [1989] has pointed out, most commercial products lack the advantages of declarative knowledge representations, such as automated explanation, do not handle defaults and exceptions very well, and provide no support for reasoning in multiple interpretative contexts. They are still following the procedural markup paradigm associated with Sprowl, wherein master documents are expressed in terms of if-then structures, repeat loops, and variables. Neither the documents nor the legal-factual circumstances occasioning their particular configurations are explicitly modeled.

4. Standard Generalized Markup Language

Standard Generalized Markup Language (ISO 8879) ("SGML") was published as an official international standard in 1986. It provides a framework for describing the structure and content of documents independently of any particular physical format. Following the notational conventions (or metasyntax) of SGML, document type definitions (DTDs) can be formulated, and specific document instances thereafter marked up in accordance with them. SGML-aware parsers and other programs can then validate, format, and otherwise process the marked up documents.

A document type definition declares the elements of which a conforming document can be composed and specifies a content model for each element. Content models define elements in terms of plain text and/or subsidiary elements, along with their permissible or required occurrences, combinations, and sequences. (SGML uses a syntax for content models that is similar to that of "regular expressions," which can in turn be viewed as describing finite state automata.)

Once a document type has been defined, the hierarchical structure of an individual document can be indicated via nested sets of start tags and end tags. (In practice, markup minimization conventions make many tags unnecessary. For instance, an end-of-chapter tag implies that the last paragraph in that chapter has also ended.)

A document type definition for the employment agreement described at the beginning of this article might look something like Figure 1. (Both this example and the discussion are oversimplified.)

Each element declaration in a DTD contains an element name and a content model. Most content models in this example reference other elements. Those linked by commas must appear in the order specified; those linked by ampersands may appear in any order. Elements marked with ? are optional; those marked with + may occur one or more times; those marked with * may occur zero or more times. CDATA is a reserved word representing an arbitrary string of characters.

A document instance of this type (not shown due to shortage of space) would be marked up with paired tags (such as <duties> or <benefits>) at the beginning and end of each of the defined elements that it contains.

```
<!DOCTYPE emp_agreement [
<!ELEMENT emp_agreement      --     (caption, preamble, duties,
                                    compensation, benefits,
                                    confidentiality, termination,
                                    other_provisions, signatures)>
<!ELEMENT caption            --     ("Employment Agreement")>
<!ELEMENT preamble           --     (effective_date & employer_name
                                    & employee_name)>
<!ELEMENT duties             --     (princ_duties, other-duties,
                                    extent_of_services?)>
<!ELEMENT compensation       --     (salary, commission?, bonus?)>
<!ELEMENT benefits           --     (health, disability, vacation,
                                    sick_leave)>
. . .                               . . .
<!ELEMENT other_provs        --     (construction &
                                    entire_agreement & severability
                                    & misc*)>
<!ELEMENT signatures         --     (employer_name & employee_name
                                    & execution_date)>
. . .                               . . .
<!ELEMENT princ_duties       --     (title+ & purpose &
                                    supervision? & region?)>
. . .                               . . .
<!ELEMENT employer_name      --     (CDATA)>
<!ELEMENT employee_name      --     (CDATA)>
<!ELEMENT misc               --     (CDATA)>
. . .                               . . .
]>
```

Figure 1

Much of the attention being paid to SGML these days revolves around the use of generalized markup techniques to facilitate electronic publishing of finished texts. But, as developed further in Section 6, the tools and methodologies it represents have equally dramatic applicability to processes of document configuration and assembly.

SGML has significant applications elsewhere in the law. For instance, the electronic filings under the United States' Securities and Exchange Commission's EDGAR system must be in SGML format. Eve Wilson [1992] describes how a hypertext front end can access an SGML-tagged law report by mapping from a document type definition into the source file. It is a safe assumption that generalized markup will become a major factor in the legal technology field. The SGML movement is a major step in the direction of formalizing the syntactic structure of cases, statutes, and other legal documents — we in the AI-and-law community will want to be involved in that process.

5. Document knowledge

In attempting to characterize the many things we can know about documents, it is useful to make some preliminary distinctions and observations:

What a document expresses vs. what is known about it

As a subject or product of legal reasoning, a document generally is of interest because of the knowledge or information it expresses (or omits), and not because of its structure, provenance, or other attributes. Apart from occasional self-references—e.g., to their names, authors, dates of creation, or purposes—documents are about extrinsic phenomena. But the expressive content of a document is only part of what a legal drafter (or drafting system) needs to be concerned with. We know things about documents that are not ordinarily expressed by or contained in those documents. It is not always enough just to know what a document says. We are often interested in the logical framework into which its content falls.

Class vs. instance

A second basic distinction is that between a document class and its instances, or, stated otherwise, between a type of document and particular exemplars of that type. When a document class or type is modeled, the model (variously called a master document, template, form, script, skeleton, boilerplate, or something else) can be regarded as a document about particular exemplars: a metadocument. An SGML document type definition, for instance, is a canonical metadocument. (Precedents, or examples, in contrast, are fellow instances of an implied class of documents.)

All forms of documentary knowledge can be ascribed to a particular level of generality. We can know things about documents in general, about certain broad categories of documents, and about particular types. There is no fixed hierarchy of superclasses and subclasses: it all depends upon on the characteristics or dimensions in terms of which we wish to abstract and generalize.

What counts as an instance is defined by one's measure of adequate concreteness. Are identical copies of the "same" document different "instances" of that document? What about two different formats? Versions? Perhaps the most common (and sensible) delimiter of difference is at the word level: if the words are different, the documents are different.

Defining a metadocument and configuring an instance are two fundamentally distinct tasks, but, as noted later, they are usefully interwoven in computer-aided drafting contexts.

The end product document does not generally retain any knowledge about how it was configured, how it might be (or have been) configured differently, or even whether it is entirely in conformity with any document model. The template's structure is used as temporary scaffolding that is only inferentially present in the final product.

The class/instance distinction is similar to that between document assembly "engines" or authoring environments, and the applications which are built using them. The former do embody some forms of knowledge (or metaknowledge, such as the fact that it is often the case that several answers to a given legal or factual question are mutually exclusive), but mostly they only afford opportunities for knowledge to be expressed. In this respect they are like expert system shells or spreadsheet programs: empty vessels awaiting substantive content. The main question to ask of a document assembly authoring environment is "what knowledge can it represent?," and of an application written in it, "what knowledge does it represent?"

Knowing about instances and classes

There is, of course, much that can be known about a given document instance. We can know about its logical content: what words, or other tokens, does it contain, and in what order? What language is it in? We can know about its physical form: How long is it? What page format, typographical style, print color(s) does it have? The document can be described at the most concrete level as a map of bits (ink dots, pixels, or other renderings) on a two-dimensional surface, or, at the most abstract, as a hierarchy of structural elements.

Much more interesting, however, for present purposes are the things we can know about document classes and the processes by which given instances of them can be configured.

A document class is characterized by

→ a universe of permissible components, which may have other components as content, and

→ constraints on (and other information about) the occurrences, combinations, and orderings of those components.

In the case of a complaint in an action for payment on a note, the universe of captions, averments, and other textual components may be quite small, and the rules governing their configuration quite simple and well-defined. In the case of a major commercial transaction document set, the universe of components may be huge, and the grammar of their composition exquisitely complex.

Partial orderings

Much of the knowledge being applied in legal drafting is neither law-specific nor drafting-specific. Legal composition is parasitic on more general forms of knowledge, such as English orthography, grammar, and style. And drafting draws upon rules of substantive law, considerations of strategy, and other knowledge resources generally seen as informing the entire lawyering craft. No knowledge base designed for computer-aided document drafting would purport to cover these two vast domains of knowledge, yet they are essential factors.

Moreover, our knowledge about documents—whether tacit or expressed—is inevitably incomplete and fragmentary. The rules and guidelines governing legal composition do not suffice to deterministically drive a given document configuration task. Even the most documented documents can be severely underspecified. The codified rules routinely run out long before all possible alternative formulations of a non-trivial section have been exhausted.

The norms that form the subject of document knowledge, in other words, specify partial orderings. They constrain the drafting process, but don't determine it.

Not only is compliance with an articulated set of norms rarely sufficient to assure a complete or adequate document; most times full compliance is not even necessary. This is because the applicable norms involve varying degrees of obligation or prohibition. Many are nonbinding, advisory, or precatory. They express generalities, stylistic tendencies, defaults, suggestions.

Needless to say, knowledge can be useful without being definitive or comprehensive. The knowledge stock of an expert draftsman—or an expert system—may well consist of rough-cut analyses, suggestions, reminders, organizing principals, heuristic interpretations, simplifications, and abstractions of the relevant law.

Purpose

Legal documents are intended to have consequences. They represent communicative acts, brimming with intentionality. Behind each is an express or implied hierarchy of purposes. Assembling a legal document involves two intertwined teleologies: the purposes of the document itself and the purpose of generating a document that best addresses its purposes. In the real world, a lot of strategic energy is consumed by efforts to protect against (and sometimes to perpetrate) intentional bias and ambiguity.

Document configuration

Configuring a document is a recursive process of selecting and sequencing textual components. A document set is composed of one or more documents. Each document consists of one or more sections or other sub-documents. Sections (perhaps further divided into sub-sections) generally contain paragraphs, which are made up of sentences. Sentences are ordered sets of phrases, decomposable into other phrases and/or single words, which are strings of characters.

Document configuration presupposes <u>some</u> pre-existing set of candidate building blocks, and constraints on their permissible combination and ordering. At this level of abstraction, then, document knowledge involves information about a universe of document components and how they can or should be combined in hierarchies.

As linguistic phenomena, it is natural to think of documents in grammatical terms. A template or document type definition defines a <u>grammar</u> to which documents must conform in order to count as well-formed instances of that type. Such document grammars, moreover, have both syntactical and lexical dimensions. The syntax of a document definition resides in the content models of its declared elements. The lexicon defines which formulations qualify as given atomic elements. Syntactical relationships can be expressed in a (potentially recursive) graph consisting of part-of links; lexical information is expressed with is-a links, orderable in an abstraction/generalization network. Tools like SGML make it possible to separate out this semantic network from a document's geometrical hierarchy of pages, columns, frames, and lines, while yet establishing a mapping between them.

Summing up

To sum up, whether expressed in a book, coded in a document assembly application, or stored in our heads, most knowledge pertinent to legal drafting falls into the broad category of considerations touching on the occurrences, combinations, and sequences of document components. Because components may be recursively nested, and because the kinds of information bearing on the relative permissibility or advisability of given configurations range from binding rules to merest suggestions, the realm of document knowledge is vast. We can have knowledge about "what goes where" at any structural level, and to any deontic degree. We can know what to use by default when no considerations seem to require otherwise. We can know what has worked in the past. We can know about laws, social conventions, patterns of human behavior, and other constellations of facts that bear on the desired contents of a document. We are eager to have all such knowledge at our disposal when drafting, to the extent it can be formalized economically.

6. Knowledgeable documents

Expressing our knowledge about legal documents with descriptive formalisms based on SGML or related standards could yield valuable benefits beyond platform- and vendor-independence. Documents that were knowledge-enriched through standardized markup would lend themselves to multiple modes of use: not only instance configuration, but nonsequential browsing, querying, access and version control, and maintenance. Document drafting could be conceived as largely a process of information retrieval. Lawyers would be more prepared to invest time in document systematization because the knowledge would be presentable in written form and be transportable from one system to another.

Document assembly applications specifically designed to work with "knowledgeable documents" could deliver high levels of functionality with minimal document-specific coding. Drawing upon the information encoded in document models and instances under draft, such applications would be able to do such things as

→ advise the user what document components are required or permitted at any given insertion point, down to the instantiation of a variable like <name of employee>

→ provide lists of candidate texts or modules for inclusion at such points, along with considerations pro and con

→ generate all permissible permutations of elements in a given context

→ validate the user's draft against applicable content models

→ allow the user to browse and manipulate the document in a collapsible outline mode

→ alert the user to and display reference material and annotations when present

→ keep track of configuration tasks remaining to be done

→ notice and highlight content and structural differences between versions of a document, going beyond the word-level differences offered by today's document comparison programs (and perhaps promoting greater fairness and efficiency in document negotiations)

Well-designed applications would maximize user initiative in direct interaction with the document model. The user should be permitted to start anywhere, jump around, be as concrete or general as desired in a given location, and leave sub-modules temporarily unconfigured. He or she should at all points be able to see the entire document so far as it has been configured, view it at different levels of abstraction, and, once a document is configured, get some visual representation of the permissible deformations or reformulations of particular parts.

Applications that "understand" descriptive markup should, moreover, facilitate user revision of document models themselves. Attorneys should be able to declare new content models, making entirely new elements possible, or revise the combinatorial, sequencing, or occurrence specifications of existing elements. They should be able to add or delete example formulations and annotations "on the fly," with the revised document models being immediately functional. They should be able to toggle among several alternative content model formulations, reflecting, say, different normalizations of a law bearing on the document.

Today's legal document assemblers offer only faint glimpses of what is described above, but there are promising reports of related work in other fields. Chamberlin [1988], for instance, describes an experimental what-you-see-is-what-you-get editor called Quill, based on the SGML document model, that allows users to interact directly with the logical structure of documents, while maintaining semantic integrity. Commercial products capable of reading and processing SGML documents have become available from vendors like ArborText, Electronic Book Technologies, Exoterica, IBM, Interleaf, SoftQuad, WordPerfect, and Xerox.

Bringing document knowledge to the surface through descriptive markup and applications that exploit it may have the collateral result of enabling more advanced forms of legal knowledge engineering in the document drafting arena. Rules and procedures can be written in terms of named objects, their attributes, and the contexts in which they appear. There are even apparent applications of deontic logic to the permissions and obligations constituting a legal document type. Since document norms are rarely definitive, and in practice often ignored or violated by users, it will fall to our knowledge systems to determine what should happen, given that some rules have been violated. Like society, real documents are often a compromise between abstract ideals and practical contingencies, requiring legal regimes that can function in the face of imperfect compliance.

A more fanciful extension of the possibilities developed here involves moving from document design to legal practice generally. The latter sphere of action, of course, often involves real-time strategic interaction with others that, unlike a document in draft, cannot be edited. But it may be that extensions to SGML such as HyTime (see Newcomb, 1991)—which provides a standard way to express scripted activities taking place in time and space (e.g., plays and operas)—have a role to play there. Choosing and sequencing informational moves in legal scenarios certainly lend themselves well to dramaturgical frames of reference. Musical composition programs may also offer useful models.

7. Conclusion

Emerging metalinguistic standards for expressing document knowledge and evolving forms of document processing technology ought to stimulate new ways to understand and support legal drafting work. In order to realize the potential of these conceptual and computational tools, however, we need to develop better awareness of the forms of knowledge involved in this characteristic lawyering activity. This article has sought on the one hand to draw attention to the richness and complexity of that knowledge, and, on the other, to suggest the usefulness of a strategy of explicit modeling using descriptive markup.

Acknowledgements

I am grateful to Daniel Evans, Thomas Gordon, Henry Perritt, and C. Howard Thomas for their helpful comments on an earlier draft of this article.

References

Allen, L. and Saxon, C. 1991. More IA Needed in AI: Interpretation Assistance for Coping With the Problem of Multiple Structural Interpretations. In *Proceedings of the Third International Conference on Artificial Intelligence and Law,* 53-61

Chamberlin, D. 1988. An Adaption of Dataflow Methods for WYSIWYG Document Processing. *Proceedings of the ACM Conference on Document Processing Systems*, 101-109

Dickerson, R. 1986. *The Fundamentals of Legal Drafting*. Boston:Little, Brown and Company.

Goldfarb, C. 1990. *The SGML Handbook.* Oxford:Clarendon Press.

Gordon, T. 1989. A Theory Construction Approach to Legal Document Assembly. In *Pre-Proceedings of The Third International Conference on Logic, Informatics, and Law*, 2:485-498. Florence.

Lauritsen, M. 1992. Building Legal Practice Systems with Today's Commercial Authoring Tools. 1 *Artificial Intelligence and Law* 87-102.

Newcomb, S., Kipp, N., and Newcomb, V. 1991. The "HyTime" Hypermedia/Time-based Document Structuring Language. *Communications of the A.C.M.* Vol. 34, No. 11, pp. 67-83

Saxon, C. 1981. *Computer Aided Drafting of Legal Documents*. Ph.D. dissertation at the University of Michigan. Ann Arbor:University Microfilms International.

Sprowl, J. 1979. Automating the Legal Reasoning Process: A Computer that uses Regulations and Statutes to Draft Legal Documents. 1 *Am. B. Found. Res. J.* 1-81

Wilson, E. 1989. Drafting Legal Documents with Justus' Clerk. In *Pre-Proceedings of The Third International Conference on Logic, Informatics, and Law*, 2:909-922. Florence.

Wilson, E. 1992. Why we need standards: an example from law, hypertext, and information retrieval. Presented at the Second International Conference on Substantive Technology in the Law School. Chicago.