Volume 7, No. 1 | January–February 2024

The Journal of Robotics, Artificial Intelligence & Law

fastcase FULL COURT PRESS

RAIL
The Journal of Robotics, Artificial Intelligence & Law
Volume 7, No. 1 | January–February 2024

THE JOURNAL OF ROBOTICS, ARTIFICIAL INTELLIGENCE & LAW (ISSN 2575-5633 (print) /ISSN 2575-5617 (online) at $495.00 annually is published six times per year by Full Court Press, a Fastcase, Inc., imprint. For customer support, please contact Fastcase, Inc., 729 15th Street, NW, Suite 500, Washington, D.C. 20005, 202.999.4777 (phone), or email customer service at support@fastcase.com.

Publishing Staff
Publisher: Morgan Morrissette Wright
Production Editor: Sharon D. Ray
Cover Art Design: Juan Bustamante

Cite this publication as:

The Journal of Robotics, Artificial Intelligence & Law (Fastcase)

A Full Court Press, Fastcase, Inc., Publication

Editorial Office

729 15th Street, NW, Suite 500, Washington, D.C. 20005
https://www.fastcase.com/

POSTMASTER: Send address changes to THE JOURNAL OF ROBOTICS, ARTIFICIAL INTELLIGENCE & LAW, 729 15th Street, NW, Suite 500, Washington, D.C. 20005.

Articles and Submissions

Direct editorial inquiries and send material for publication to:

Steven A. Meyerowitz, Editor-in-Chief, Meyerowitz Communications Inc., 26910 Grand Central Parkway, #18R, Floral Park, NY 11005, smeyerowitz@meyerowitzcommunications.com, 631.291.5541.

Material for publication is welcomed—articles, decisions, or other items of interest to attorneys and law firms, in-house counsel, corporate compliance officers, government agencies and their counsel, senior business executives, scientists, engineers, and anyone interested in the law governing artificial intelligence and robotics. This publication is designed to be accurate and authoritative, but neither the publisher nor the authors are rendering legal, accounting, or other professional services in this publication. If legal or other expert advice is desired, retain the services of an appropriate professional. The articles and columns reflect only the present considerations and views of the authors and do not necessarily reflect those of the firms or organizations with which they are affiliated, any of the former or present clients of the authors or their firms or organizations, or the editors or publisher.

QUESTIONS ABOUT THIS PUBLICATION?

For questions about the Editorial Content appearing in these volumes or reprint permission, please contact:

Morgan Morrissette Wright, Publisher, Full Court Press at morgan.wright@vlex.com or at 202.999.4878

For questions or Sales and Customer Service:

Customer Service
Available 8 a.m.–8 p.m. Eastern Time
866.773.2782 (phone)
support@fastcase.com (email)

Sales
202.999.4777 (phone)
sales@fastcase.com (email)

ISSN 2575-5633 (print)
ISSN 2575-5617 (online)

Domain-Specific Languages and Legal Applications

Alexis Chun, Meng Weng Wong, and Marc Lauritsen*

Despite the rise of low-code and no-code development tools and the maturation of large language model approaches in the software world, many legal software and application tools are still hand coded. One common bottleneck for legal software and application tools is the domain-specific, knowledge-based, and experience-based nature of legal practice, which makes legal tech a highly technical and multi-disciplinary endeavour. Developers often need to encode legislation, regulations, legal concepts, and other quasi-legal frameworks in order to ask users the right questions, provide appropriate guidance, accurately represent legal concepts, or generate the appropriate documents. The difficulty of faithfully expressing such frameworks within the confines of custom code or within existing languages (natural or programming), and the resources required to resolve it, impede innovation. This article analyses domain-specific languages (DSLs) as promising opportunities to lessen that difficulty, surveys 15 recent legal DSLs for semantic expressiveness and suitability for industry adoption according to an eight-point framework, and presents an innovative application of one such DSL to automatically generate a user-friendly web application, draw related visualizations to aid the drafter, and transpile to multiple targets for the convenience of researchers working in other languages.

Introduction

Readily available and inexpensive codified legal know-how is increasingly critical in both commercial and nonprofit contexts. Yet it often remains costly and time-consuming to produce.

Most practical legal applications are created and maintained using laborious hand-coding techniques, often including quite primitive methods. Most app makers (including one of the authors) are not professional software developers. That is true within private law firms and law departments as well as in nonprofit organizations. Legal application developers who follow the academic literature have long been aware of methodologies and theories to directly connect statements of the law with their programmed implementation, but few so far have taken advantage of them.

In short, much legal app development remains highly artisanal. Domain-specific languages (DSLs) may offer a solution.

Robotics, Artificial Intelligence & Law / January–February 2024, Vol. 7, No. 1, pp. 19–42.

ISSN 2575-5633 (print) /ISSN 2575-5617 (online)

This article is organized as follows. Following this introduction, the second section describes some common forms of interactive legal applications and their development processes, including two examples. The third section lays out some of the challenges developers face and imagined solutions. The fourth section introduces DSLs and their applicability. The fifth section 5 introduces the L4 DSL with example screenshots. And the sixth section concludes.

Contemporary Legal Knowledge Engineering

Expert systems and various forms of document automation are among the most common forms of knowledge-based software found in law offices in recent decades. A common pattern involves scripted "interviews" and modelled documents, which are typically fashioned using procedural code and manual document markup.

Tools like Neota Logic, BRYTER, Contract Express, HotDocs, and Legito provide integrated development environments within which such apps can be built and maintained. (There is a wealth of such tools. One site (https://www.docautodatabase.com/) recently identified over 200 in the document automation category alone.) Another collection (with over 5,000) of such applications in the nonprofit sector in the United States is at LawHelp Interactive (LHI), which provides interactive guidance and bespoke form assembly without charge to millions of users. (About a million packages of customized forms were generated in 2022.) Within its technology stack the main providers of end-user functionality are HotDocs, from CARET, and A2J Author, from the Center for Computer-aided Instruction.

Document automation applications are typically driven by the forms they need to generate (What information should be placed where under what circumstances?) and by informal know-how communicated by practitioners (What should users know about the process they are undergoing? What steps should be taken or avoided as a practical matter to reach an optimal outcome?). But sometimes they also need to explicitly reflect the detailed rules expressed in a statute or regulation. In those situations, scripted interviews and model documents are not sufficient.

Two Examples

The Uniform Child-Custody Jurisdiction and Enforcement Act (UCCJEA) has been adopted by 49 U.S. states, the District

of Columbia, Guam, Puerto Rico, and the U.S. Virgin Islands. It governs the rules whereby courts decide which have jurisdiction to adjudicate questions of child custody. Family law applications typically need to encode aspects of the UCCJEA in order to advise users and properly complete court forms.

One can find various online resources that attempt to summarize how the UCCJEA "works," such as shown in Figure 1.

A2J Author provides an easy-to-use environment via which non-programmers can script "guided interviews." A built-in mapper helps users visualize their creations. Some can quickly become unwieldy, such as that shown in Figure 2.

Figure 1

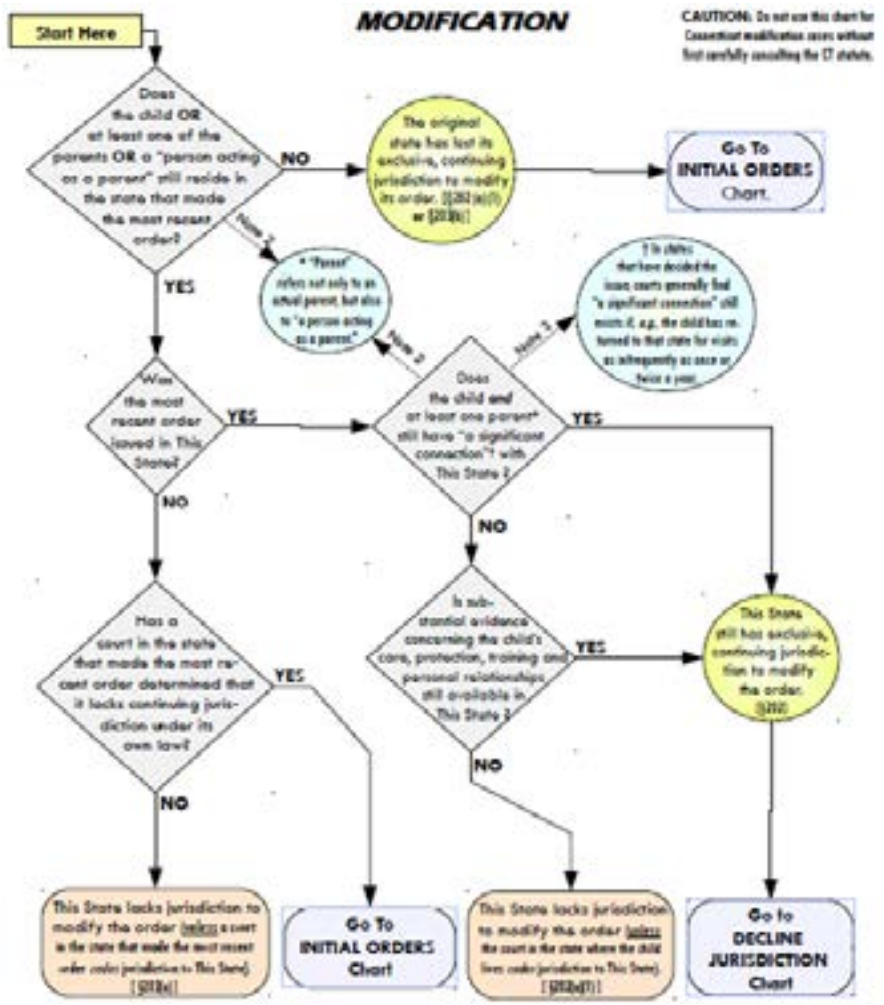
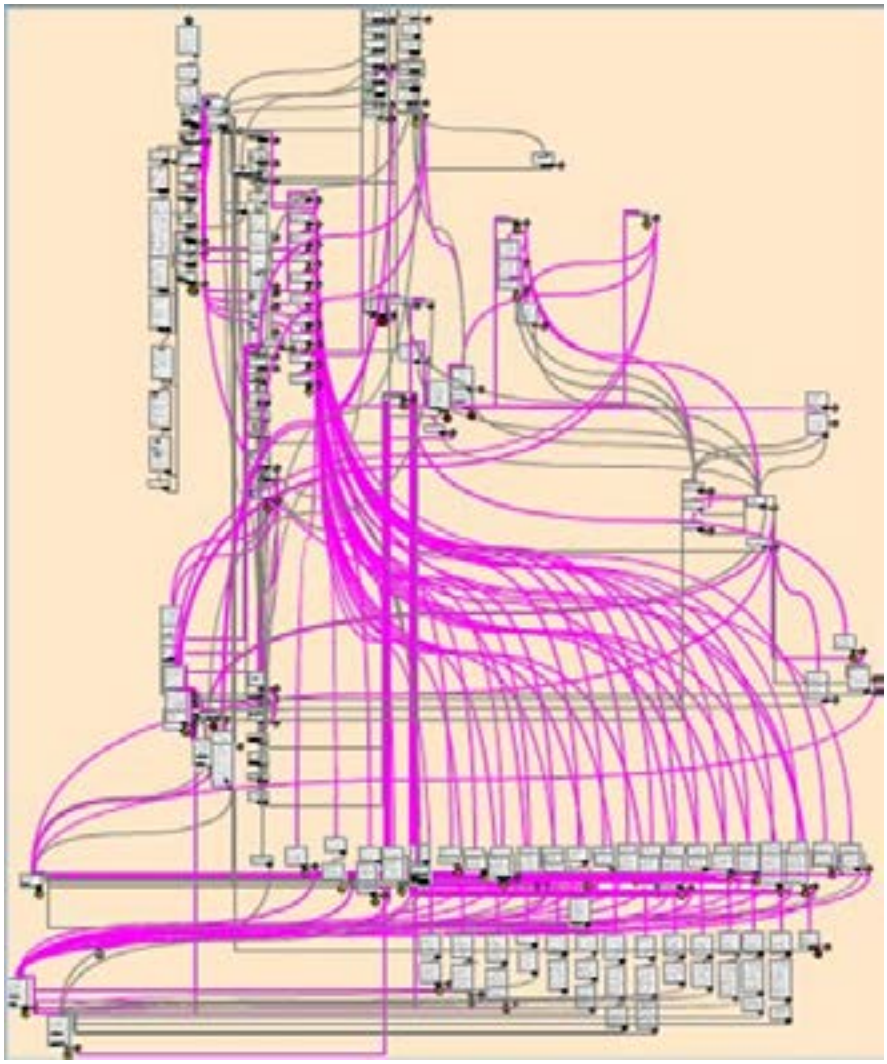


Figure 2

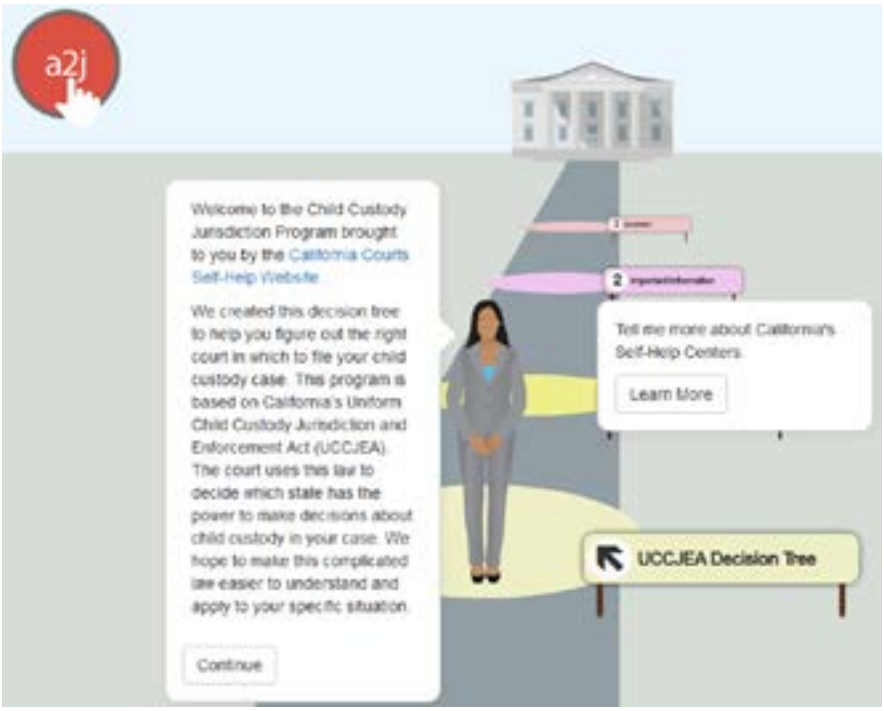


One LHI application, built in A2J Author, that one of the authors has assisted with was intended to simply guide an inquiring user as to which court likely has jurisdiction to handle questions about the custody of their children. Its first page looks like that shown in Figure 3.

Its associated "map" is quite sparse. See Figure 4.

The challenge for the developer (after a succession of earlier developers) was to confirm whether the app faithfully followed at

Figure 3



least key parts of the UCCJEA, and gave accurate guidance. That involved in part coming up with an external representation of its key provisions. One resorted to sticky notes in the attempt (see Figure 5).

Another less-than-satisfactory effort involved page-by-page documentation in Word, which also didn't capture the as-built logic of this application.

A second application needing to reflect the UCCJEA's logic was a HotDocs interview and template set for litigants seeking a divorce in Washington State. There, a domain expert (practicing lawyer) struggled to capture that logic so that it could be expressed in HotDocs code, and ended up finding Excel the best tool for doing so (see Figure 6).

That in turn was used by the HotDocs expert to create a set of computations that drive the interview and infer the proper result. For an example, see Figure 7.

Figure 4

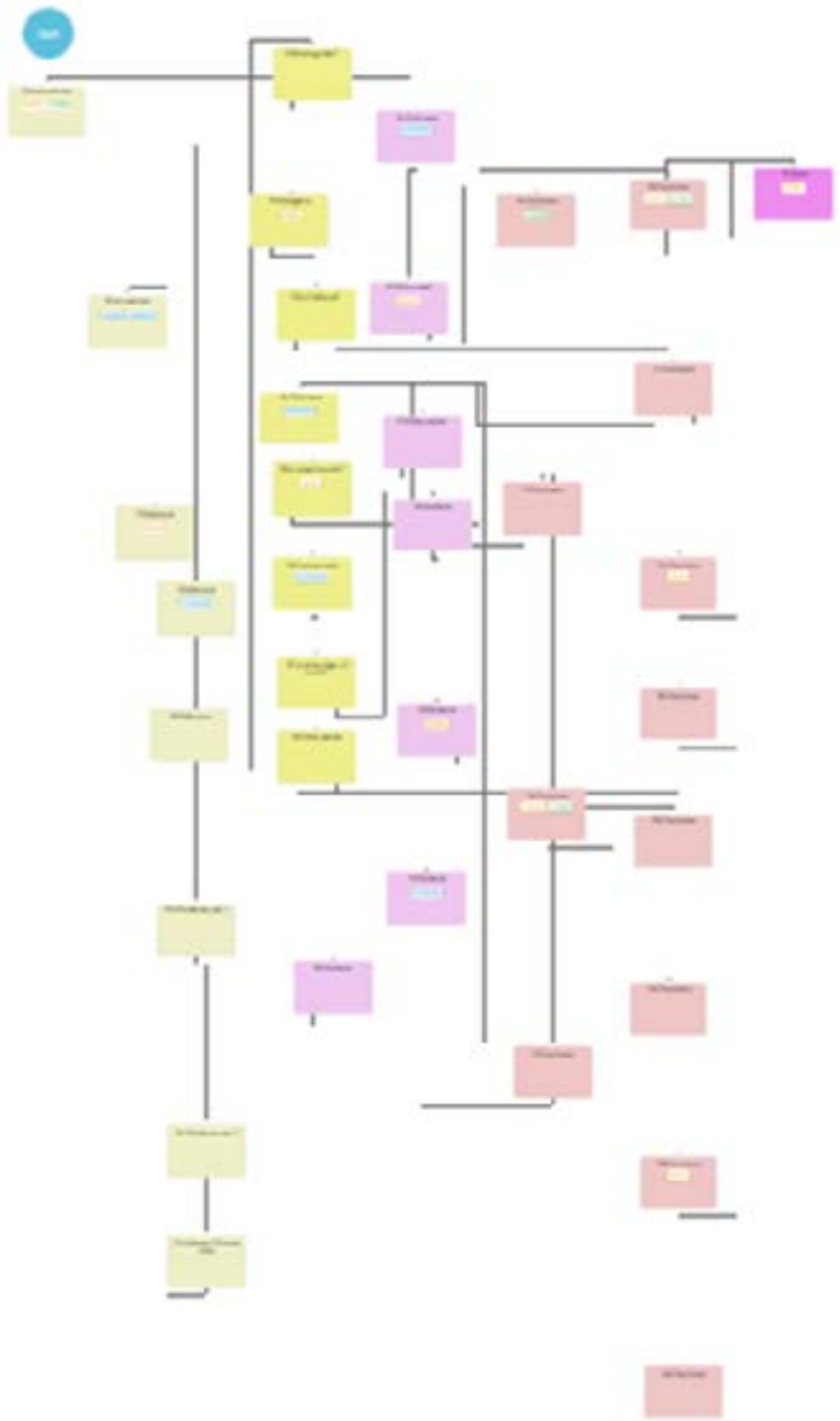


Figure 5

Figure 6

Figure 7

```
IF Child jur collect uccjea info together CO
   IF (VALUE(Children all residence location MC[1]) != "Washington" AND
      (NOT VALUE(Child jur all non WA residence 6 mos plus TF)) AND
      VALUE(Child jur all prior residence WA 6 months plus TF))
      "<p ""a, b and c"">"
      SET Counter TO 1
      WHILE ANSWERED(Child name last TE[Counter])
         IF AGE(Child birth date DA[Counter]) < 18
            RESULT + Child name full indexed CO[Counter] + "<p>"
         END IF
         INCREMENT Counter
      END WHILE
      RESULT + "<pe>"
   END IF
ELSE
   REPEAT Children Repeat
      FORMAT "a, b and c"
      FILTER Child jur WA home state w/in 6 months filter CO
      RESULT + Child name full CO
   END REPEAT
END IF
```

Opportunities and Challenges

The above examples are just briefly sketched to illustrate the challenges faced by developers and their collaborators. Namely,

1. There is no widely recognized methodology for reliably incorporating statutory rules into a custom programmed application.
2. There is no widely recognized method for confirming whether one's efforts to so incorporate legal "code" into such an application were successful. This raises serious quality control issues.[1]
3. Domain experts generally are not able to review application code itself to satisfy themselves about its completeness and consistency.
4. Such applications lack automated explainability. They do not readily interoperate with external specifications or code. (Most legal apps are poor at explaining themselves because we haven't educated them about *why* particular

questions are asked, guidance is offered, and documents are generated.)

5. From a computer science perspective, these development processes and ad hoc knowledge representation formats ignore decades of advances in information management, software engineering, and programming language theory.

The literature around computable contracts,[2] computational law,[3] and Rules as Code[4] points to a future in which the above problems have been solved. What does that future hold?

When building a legal assistance app that needs to reflect a defined set of rules (from a statute, regulation, or other source of governance), the developer can access both the natural language statement of those rules and an unambiguous, machine-readable equivalent. Software can bidirectionally move between both isomorphic forms. Those forms can be used as input to a design-time process that produces appropriate code for the destination platform. For instance, interviews can automatically be generated that ask the minimal set of questions needed to resolve a legal issue. (An optimal "question tree.") Alternatively, a run-time process could deliver needed logic to that platform via an application programming interface (API). Conversely, tools would be available to generate an external specification of the logic of an application for purposes of validation, maintenance, and debugging. Tools could automatically construct graphs, flowcharts, decision trees, and other visualizations to represent laws and contracts and aid end-user understanding of legal complexity. Such outputs could also be used to support in-session explanations of inferences performed against user inputs, meeting the goals of explainability and algorithmic transparency. The system would usefully identify all ultimate and intermediate conclusions described in a model, as well as all predicates and data elements playing roles in rules/inferences.

Moving upstream, certified software encodings could be published by government or other relevant authorities. The open-source movement of the past four decades, overlapping with the ideals of the rule of law, demand that digital legislation and regulations should be publicly available for free.[5] For example, a state agency could release a "code companion" library on Github for consumption by third-party app developers, minimizing the need for software developers to conduct legislative interpretation.

Current tools attempt to achieve some of these goals. Complex computations are not easily implemented in A2J Author, but HotDocs offers a reasonably complete programming environment for such things, including parameterized computations and local variables. Such things can be used to drive questioning and inferences; the challenge is writing, validating, and updating them! A2J Author is over 20 years old; HotDocs is over 30. The difficulties of revising software products to support fundamental new functionality are well known.

What next-generation technologies could help realize the vision?

DSLs to the Rescue

In the computer science and software engineering disciplines, DSLs are a widely accepted approach to making a particular problem domain more tractable to software and to developers. For example, the need to structure hypertext data begat HTML; the need to manage the visual styles and layout of web pages begat CSS; the need to read from and write to databases containing tabular data begat SQL. All are DSLs, defined as:[6]

> A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.

In recent years, academics and software developers have seized on DSLs as a promising way to enable the vision outlined above.[7] Computer scientists have proposed languages and libraries for law (FormaLex, Catala, OpenFisca); "smart contract" languages have appeared with the rise of blockchain technologies (Accord Project, Cardano, Deon Digital's CSL); and non-blockchain-oriented contract languages have also appeared (FCL from McMaster, Symboleo from uOttawa, Logical English from Imperial) many of which were inspired by Jones, Eber, and Seward's pioneering 2001 paper "Composing Contracts."[8] Others include Stipula, DCR Graphs, Orlando, McCarty's LLD, Eiger, and Blawx.[9]

Brief Survey of Legal DSLs

Due to space constraints, the capsule summaries presented in this section may not do justice to the full vision of each language but are intended to illustrate the range of approaches to legal DSLs. A comprehensive survey is beyond the scope of this article. DSLs such as ACC,[10] BCL,[11] CL,[12] DataLex,[13] Lexon,[14] and RegelSpraak[15] are omitted with regrets.

- **OpenFisca**: France, 2011; Python API; primarily numerical calculations for income tax and other quantitative domains; support for multiple versions of legislation and multiple jurisdictions (FR, US, UK, AU, NZ).
- **Catala**:[16] France, 2019; external DSL; strong support for numerical calculations and for isomorphic representation of statutes expressed in terms of default logic with exceptions.
- **FormaLex**:[17] Buenos Aires, 2011; based on LTL (Linear Temporal Logic) and intended to discover inconsistencies using model checking.
- **FCL**:[18] McMaster University, 2018; a type-theoretic approach to formalizing and reasoning over events, deontics, and real values.
- **Symboleo**:[19] uOttawa, 2020; emphasis on deontic logic, with support for events, pre- and post-conditions, and assertions.
- **Stipula**:[20] Italy, 2021; emphasis on timed deontics as state, with support for assets and the notion of agreement as synchronization.
- **Blawx**:[21] Canada, 2020; focus on usability through a GUI based on Scratch; well-formed statements can be constructed through drag-and-drop.
- **DCR Graphs**:[22] Copenhagen, 2011; a declarative, event-based process model developed in partnership with industry.
- **Eiger**:[23] Switzerland, 2022; embedded Haskell DSL, deployed at PwC Switzerland.
- **Orlando**:[24] United States, 2021; an academic project featuring a concise CNL, with strong visualization and explainability features, for conveyancing as the initial application domain.

- **Accord Project**:[25] United States, 2019; intended for blockchain use, provides an executable language, a data modeler, and a document assembler.
- **CSL**:[26] Copenhagen, 2012; a trace-based external DSL with support for events and deontics, adapted for blockchain use and currently the subject of commercialization efforts at the start-up deondigital.com.
- **Logical English**:[27] Imperial, 2020; web-based logic programming with syntactic sugar borrowed from the tradition of Controlled Natural Languages; uses the Event Calculus to track state over time.
- **Epilog**:[28] Stanford, 1980s; a member of the logic programming family with a focus on databases and unusual direct support for logic programming in an interactive web environment.
- **Language for Legal Discourse**:[29] Rutgers, 1989; a sophisticated theoretical basis for converting from legal natural language to a formalization.

Analytic Frameworks for Legal DSLs

Requirements for legal specification languages have been previously enumerated. Hvitved[30] identifies the following 16 requirements:

1. Contract model, contract language, and a formal semantics.
2. Contract participants.
3. (Conditional) commitments.
4. Absolute temporal constraints.
5. Relative temporal constraints.
6. Reparation clauses.
7. Instantaneous and continuous actions.
8. Potentially infinite and repetitive contracts.
9. Time-varying, external dependencies (observables).
10. History-sensitive commitments.
11. In-place expressions.
12. Parametrised contracts.
13. Isomorphic encoding.
14. Run-time monitoring.

15. Blame assignment.
16. Amenability to (compositional) analysis.

Athan et al.[31] identify the following functionalities:

1. Supports modelling different types of rules (constitutive v. prescriptive).
2. Represents normative effects (e.g., reparation and compensation).
3. Implements defeasibility (to handle conflicts between rules).
4. Implements isomorphism.
5. Alternatives (can represent multiple interpretations).
6. Manages rule reification (Jurisdiction, Authority, Temporal attributes).

We introduce a framework that consolidates the above formal requirements under semantics and expressiveness (criteria 1-3), and goes beyond to anticipate usability concerns and suitability for adoption in industry and government (criteria 4-8):

1. Equipped with a formal semantics describing the language in terms of its underlying logics (defeasible, default, temporal, deontic, etc.).
2. Capable of expressing a wide variety of contract genres (such as financial agreements, insurance policies, employment contracts, and leases).
3. Capable of expressing a wide variety of legislative and regulative genres (such as criminal law, building permits, privacy regulations, and even rules of court).
4. Open-source implementation available (some languages are given only as theoretical constructs without accompanying software; others are proprietary).
5. Syntactically "low code" and user friendly with documentation and integrated development environment (IDE) support (intended to be read and written by an individual without extensive training in programming or law).
6. Capable of producing explanations for its decisions, in text or via visual notations.
7. Application-oriented (intended for industry use).

Table 1

	1	2	3	4	5	6	7	8
OpenFisca			Q	T			T	
Catala	T		Q	T			T	
FormaLex	T		E	T				
FCL	T	E						
Symboleo	T	E		T				
Stipula	T	E		T		T		
Blawx			T	T	T	T	T	
DCR Graphs	T	E	E			T	T	
Eiger	T	T		T	E		T	
Orlando	T			T	T	T		
Accord		T		T	T		T	
CSL	T	T					T	
Logical English	T	T	T	T	T	T	T	
Epilog	T	T	T	T	T		T	
LLD		T	T					

T: true (blanks indicate insufficient information to conclude true; logic programmers may consider this negation-as-failure).
Q: the primary expression domain is quantitative calculations.
E: the primary expression domain is an event-oriented calculus.

8. Oriented toward interoperability (imports from and exports to other languages and standard formats such as LegalRuleML, BPMN, and DMN).

As of early 2023, using that framework, a rough assessment (*pace* the authors of the languages) produced the analysis shown in Table 1.

The analysis shows that many legal languages, while rigorously defined, are focused on relatively narrow areas of concern: either laws or contracts; either quantitative calculations or state-transition systems with an emphasis on deontics and verifiability. To realize the vision of wider adoption, additional requirements must be satisfied, which go beyond the charter of the typical academic research project.

In 2020, a research program was begun to develop a DSL for laws and contracts that meets all the above criteria.

The L4 DSL

The remainder of this article identifies L4 as a novel solution in the space of legal DSLs by informally outlining its semantics and expressive scope. A brief walkthrough of a real-world use of L4 is presented to illustrate how it supports innovative applications that fulfill the features and vision from the third section.

Semantics

The L4 DSL combines first-order logic for reasoning over "static" decisions such as numerical calculations and Boolean predicates, with the semantics of a state transition system for reasoning over "dynamic" events and obligations in time. The guards of the state transitions are expressed using the "static" logic. These two major sets of semantics—the "statics" and the "dynamics"—are visualized using circuit diagrams and process workflow diagrams, respectively. These semantics have been found to be sufficient to formalize all the case studies encountered so far.

Default Logic

The "static" rules have a concrete syntax that can be considered a sugared form of Prolog. Default reasoning is supported with the use of default branches in pattern matches. The runtime reasoner is augmented with two modes of operation: in "hard" mode, only user input is used to calculate decisions; in "soft" mode, input elements can be marked using the TYPICALLY key word; these defaults are provisionally accepted into decisions and treated as assumptions for the user to confirm or deny.

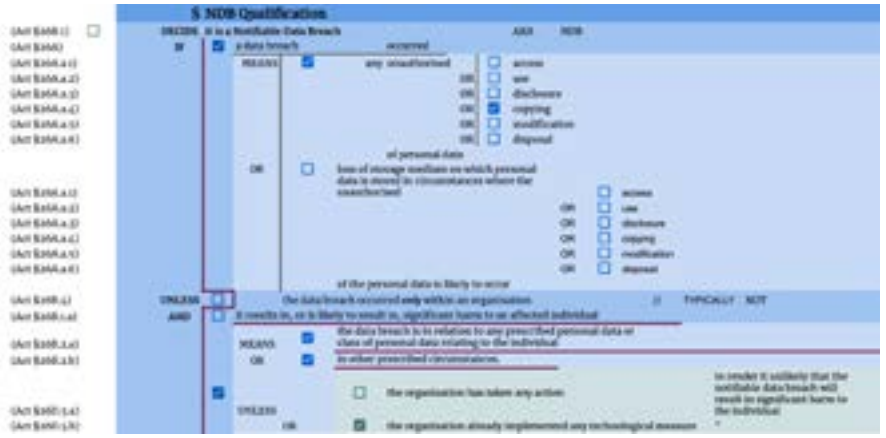
Interactions Between Rules

Legal clauses are frequently prefixed with "notwithstanding," "despite," and "subject to" modifiers. L4 interprets these modifiers as a priority ordering and adjusts rule application and result chaining accordingly. In this way L4 supports a limited form of defeasible logic.

Spreadsheet as Interactive Development Environment

With industry adoption in mind, L4 prioritizes a spreadsheet-based IDE over the traditional text editor. This innovation delegates

Figure 8



certain lexing and value-typing functionality from the parser to the IDE.

In 2021-2022, a case study called for the encoding of a portion of real-world privacy legislation. The source material spanned approximately 260 pages of text, including advisory guidelines and a compliance guide for organizations. The completed encoding occupied approximately 260 lines of code.

In this case study, the primary rules are as follows: a data breach, once discovered, must be assessed; and if it is assessed to be a notifiable data breach, it must be reported to both the relevant government body and to the affected individuals. Both rounds of obligations come with deadlines. The decision criteria for whether a breach is notifiable are complex.

An Example of Constitutive Rules

The *decision* as to whether a data breach is *notifiable* can be expressed using the constitutive rule shown above. The rule is essentially a Boolean proposition composed with the operators AND, OR, NOT, and UNLESS, where grouping is indicated using layout indentation.

The L4 tooling automatically generates the corresponding decision diagram in a variety of formats and semantic resolutions. The simplest format shows the decision nodes in a circuit diagram of parallel (OR) and series (AND) elements. The more detailed format includes the text of each node. This diagram makes it easy to

Figure 9

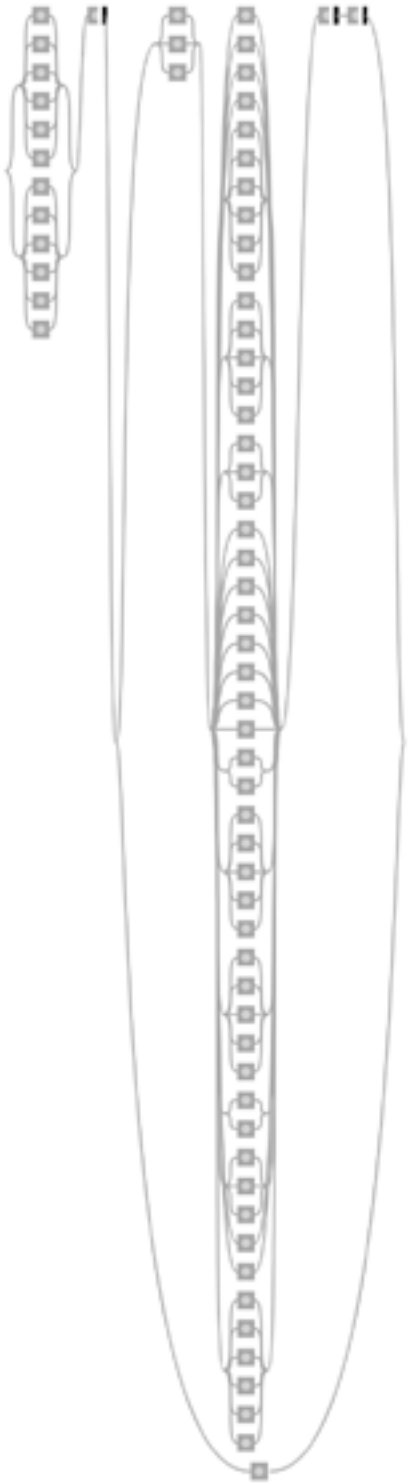


Figure 10

quickly form an impression of the overall shape and structure of the decision logic.

An Example of Prescriptive Rules

The *obligation* to assess whether a breach is notifiable can be expressed using the following regulative/prescriptive rule, which contains deontic, epistemic, and temporal elements.

The workflow diagram corresponding to the full rule set is rendered in the form of a Petri Net (see Figure 11). Other formalisms may follow in future.

Transpilation to Other Formats

Once these rules are parsed into the L4 interpreter's abstract syntax tree (AST) and related intermediate representation formats, they can be rewritten and transpiled to a variety of downstream representations. As of mid 2023, L4 supports output to JSON, Typescript, Purescript, and Python. On the road map are other languages and formats such as DocAssemble, Catala, OpenFisca, Blawx, Prolog, and Epilog, as well as interchange standards like LegalRuleML, BPMN, and DMN. In response to industry demand, other formats and technology stacks could be added to that list—Neota Logic, BRYTER, HotDocs, and others are potential transpilation targets so that enterprises already committed to a document assembly or contract life cycle management platform can integrate L4 with existing business processes. Any existing or future academic language can also be supported as a transpilation target, opening the door to research cross-compatibility.

Figure 11

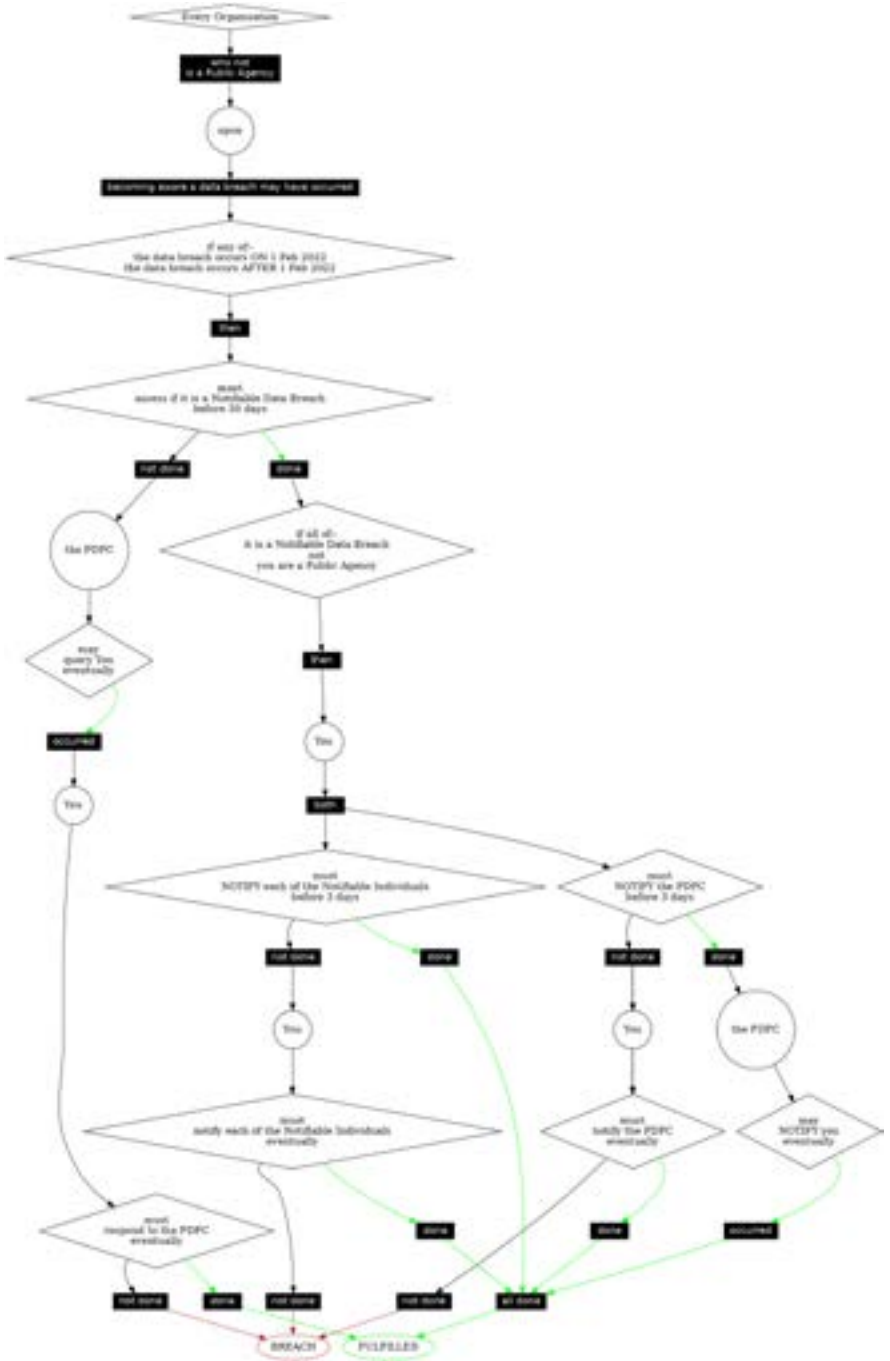

Every Organisation
who not
is a Public Agency
agree
becoming aware a data breach may have occurred
If any of:
the data breach occurs ON 1 Feb 2022
the data breach occurs AFTER 1 Feb 2022
then
must
assess if it is a Notifiable Data Breach
before 30 days
not done
done
the PDPC
If all of:
it is a Notifiable Data Breach
and
you are a Public Agency
must
query You
eventually
then
OCCURRED
Yes
Yes
both
must
NOTIFY each of the Notifiable Individuals
before 3 days
must
NOTIFY the PDPC
before 3 days
not done
done
not done
done
Yes
Yes
the PDPC
must
notify each of the Notifiable Individuals
eventually
must
notify the PDPC
eventually
must
NOTIFY you
eventually
must
respond to the PDPC
eventually
done
done
occurred
not done
done
not done
not done
all done
BREACH
FULFILLED

Figure 12

Automated Web App Generation

The encoding of legislation into L4 was a means to an end. In this case study, the goal was to automate the creation of a citizen-facing web application from the formalization. To that end, a reusable toolchain involving a transpiler to Purescript and a front-end in Vue was developed to convey the legal logic from the encoding to an interactive application for citizens and affected enterprises.

As the input spreadsheet is edited, a web application is regenerated live, with a typical rebuild time of less than 10 seconds. As end-users answer the questions presented in the app, the decision logic attempts to resolve the top-level answer to a "yes" or a "no." The L4 toolchain thus meets the description of an "application generator" as described by Cleaveland.[32]

This web app is not polished to commercial standards but was developed as a proof-of-concept to demonstrate the feasibility of the "Rules as Code" approach. The entire package can be bundled for further refinement and public-facing delivery.

All components of the system, including front-end IDE support (in Google Sheets, powered by Google Apps Script), the L4 parser/interpreter toolchain, the visualizers, the transpilers, and web app infrastructure are available on Github.

Conclusions

The pain points of the current legal application development model can be remedied by the adoption of a DSL-based engineering methodology. Software engineering principles like "separation of concerns" advise that rather than implementing the "business logic" of the law directly in operational software, one should abstract out representations of the law into an executable specification, in a DSL with the appropriate semantics. In recent years, following this motivation, DSLs have been developed in academia and by industry (typically with blockchain applications in mind), each one exploring a different theoretical approach. In 2019 the authors detected an opportunity to make a novel contribution, at the intersection of wide semantic expressivity, "low-code" usability, and a focus on adoption by industry and governments, through comprehensive tooling, open-source availability, and planned interoperability with existing systems. This article presents an encoding of real-world legislation into L4, presents some of the syntax for constitutive and prescriptive rules, and shows how a user-facing web application can be generated automatically.

It is straightforward to envision how the UCCJEA examples could benefit from this treatment: the encoding language does not have to be developed ad hoc; the development environment provides supporting visualizations to aid the drafter; and the accompanying tools are responsible for exporting to formats that can be consumed by downstream applications, if the natively generated applications are not already sufficient to serve the user. Keeping the legal rules explicit supports the goals of explainability and transparency which are increasingly important social priorities. The use of open DSLs to support legal applications is thereby shown to be a key ingredient of the vision outlined in this article.

Notes

* Alexis Chun is a co-founder of Legalese, a computational law startup that applies computer science to law, and the industry director at the Centre for Computational Law, Singapore Management University. Meng Weng Wong is a co-founder of Legalese and the principal investigator at the Centre for Computational Law. Marc Lauritsen, the president of Capstone Practice Systems and a former senior research associate at Harvard Law School, teaches courses at Suffolk University in which students build software applications

that help with legal work. This research/project is supported by the National Research Foundation, Singapore under its Industry Alignment Fund—Pre-positioning (IAF-PP) Funding Initiative. We thank Laurie Garber of the Northwest Justice Project and Bart Earle of Capstone Practice Systems for some of the examples in subsection Two Examples. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

1. *See, e.g.*, Marc Lauritsen & Quinten Steenhuis, "Substantive Legal Software Quality: A Gathering Storm?" 52-62 (June 17, 2019), https://dl.acm.org/doi/10.1145/3322640.3326706.

2. Harry Surden, "Computable Contracts" (2019), https://papers.ssrn.com/abstract=2216866.

3. Nathaniel Love & Michael Genesereth, "Computational Law," in Proceedings of the 10th International Conference on Artificial intelligence and Law (ICAIL '05), Association for Computing Machinery, New York, 205-09 (2005), https://dl.acm.org/doi/10.1145/1165485.1165517.

4. Pia Andrews & Tim De Sousa, "We Can Create Better Results When We Code the Rules," Apolitical (Jan. 22, 2020), https://apolitical.co/solution-articles/en/we-can-create-better-results-when-we-code-the-rules.

5. *See* Andrew Mowbray, Philip Chung & Graham Greenleaf, "Applying the Rule of Law in Automated Decision Systems Through Rules as Code" (AustLII's Submission to the Robodebt Royal Commission) (Feb. 10, 2023), https://papers.ssrn.com/abstract=4355989.

6. Arie van Deursen, Paul Klint & Joost Visser, "Domain-Specific Languages: An Annotated Bibliography," ACM SIGPLAN Notices 35, 6, 26-36, (June 2000), https://dl.acm.org/doi/10.1145/352029.352035.

7. *E.g.*, Jason Morris, "Spreadsheets for Legal Reasoning: The Continued Promise of Declarative Logic Programming in Law" (Apr. 15, 2020), https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3577239.

8. Simon Peyton Jones, Jean-Marc Eber & Julian Seward, "Composing Contracts: An Adventure in Financial Engineering," in Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity (FME '01), Springer-Verlag, Berlin, Heidelberg (2001), 435.

9. More on legal DSLs can be found in Shrutarshi Basu, Anshuman Mohan, James Grimmelmann & Nate Foster, "Legal Calculi (ProLaLa 2022—Programming Languages and the Law 2022)—POPL 2022," https://popl22.sigplan.org/details/prolala-2022-papers/6/Legal-Calculi; and Christopher D. Clack, "Languages for Smart and Computable Contracts" (2021), https://arxiv.org/abs/2104.03764.

10. Adam Zachary Wyner, "A Functional Program for Agents, Actions, and Deontic Specifications," Lect. Notes Comput. Sci. (2006), 239-56.

11. S. Neal, J. Cole, P. F. Linington, Z. Milosevic, S. Gibson & S. Kulkarni, "Identifying Requirements for Business Contract Language: A Monitoring Perspective," Seventh IEEE International Enterprise Distributed Object Computing Conference (2003), https://www.academia.edu/80906747/Identifying_requirements_for_Business_Contract_Language_a_monitoring_perspective.

12. Seyed M. Montazeri, Nivir K.S. Roy & Gerardo Schneider, "From Contracts in Structured English to CL Specifications," Electron. Proc. Theor. Comput. Sci. 68 (2011), 55.

13. Andrew Mowbray, Philip Chung & Graham Greenleaf, "Explainable AI (XAI) in Rules as Code (RaC): The DataLex Approach," SSRN Electron. J. (Apr. 25, 2022), https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4093026.

14. Lexon. Retrieved February 12, 2023 from http://www.lexon.tech/.

15. Stijn Hoppenbrouwers, "RegelSpraak: A CNL for Executable Tax Rules Specification" (January 2021), https://www.academia.edu/74358568/RegelSpraak_a_CNL_for_Executable_Tax_Rules_Specification.

16. Denis Merigoux, Nicolas Chataing & Jonathan Protzenko, "Catala: A Programming Language for the Law," Proc. ACM Program. Lang. 5, ICFP (2021), 77, 1 (Aug. 19, 2021), https://dl.acm.org/doi/10.1145/3473582.

17. Daniel Gorín, Sergio Mera & Fernando Schapachnik, "A Software Tool for Legal Drafting," Electron. Proc. Theor. Comput. Sci. 68, 71-86 (Sept. 13, 2011), https://arxiv.org/abs/1109.2658v1.

18. William M. Farmer & Qian Hu, "FCL: A Formal Language for Writing Contracts," in Quality Software Through Reuse and Integration, Stuart H. Rubin & Thouraya Bouabana-Tebibel (eds.), Springer International Publishing, Cham, 190-208 (2018), https://link.springer.com/chapter/10.1007/978-3-319-56157-8_9.

19. Sepehr Sharifi, Alireza Parvizimosaed, Daniel Amyot, Luigi Logrippo & John Mylopoulos, "Symboleo: Towards a Specification Language for Legal Contracts," in 2020 IEEE 28th International Requirements Engineering Conference (RE), 364-69 (2020), https://ieeexplore.ieee.org/document/9218159.

20. Silvia Crafa, Cosimo Laneve, Giovanni Sartor & Adele Veschetti, "Pacta sunt servanda: Legal contracts in *Stipula*," Sci. Comput. Program. 225, 102911 (Jan. 2023), https://www.sciencedirect.com/science/article/abs/pii/S0167642322001447?via%3Dihub.

21. Jason Morris, "Blawx: Rules as Code Demonstration," MIT Computational Law Report (Aug. 14, 2020), https://law.mit.edu/pub/blawxrulesascodedemonstration/release/1.

22. Thomas T. Hildebrandt & Raghava Rao Mukkamala, "Declarative Event-Based Workflow as Distributed Dynamic Condition Response Graphs," Electron. Proc. Theor. Comput. Sci. 69, 59-73 (Oct. 19, 2011), https://arxiv.org/abs/1110.4161v1.

23. Alexander Bernauer & Richard A. Eisenberg, "Eiger: Auditable, Executable, Flexible Legal Regulations" (Sept. 11, 2022), https://arxiv.org/abs/2209.04939.

24. Shrutarshi Basu & Nate Foster, "A Programming Language for Future Interests," Yale Journal of Law & Technology 24, 75, https://yjolt.org/programming-language-future-interests.

25. Accord Project, https://accordproject.org/.

26. Tom Hvitved, "Contract Formalisation and Modular Implementation of Domain-Specific Languages," Ph.D. thesis (2011).

27. Robert Kowalski, Jacinto Dávila Quintero & Miguel Calejo, "Logical English for Legal Applications" (2021), https://www.doc.ic.ac.uk/~rak/papers/LE_for_LA.pdf.

28. Epilog, http://logic.stanford.edu/epilog/homepage/index.php.

29. L. Thorne McCarty. Position Paper: LLD Is All You Need (2022).

30. Tom Hvitved, "Contract Formalisation and Modular Implementation of Domain-Specific Languages," Ph.D. thesis (2011).

31. Tara Athan, Guido Governatori, Monica Palmirani, Adrian Paschke & Adam Wyner, "LegalRuleML: Design Principles and Foundations" (2015), https://link.springer.com/chapter/10.1007/978-3-319-21768-0_6.

32. J.C. Cleaveland, "Building Application Generators," IEEE Software 5, 4, 25-33 (July 1988), https://ieeexplore.ieee.org/document/17799.