Multiphase Transformation in the Legal Text-to-Program Approach

Vytautas Čyras / Friedrich Lachmayer
Vilnus University, Lithuania / University of Innsbruck, Austria

Keywords: legal knowledge representation, semantic information systems, visualization, visual metaphor, legal task, phase

Abstract: The paper explores the building of a bridge between legal texts and their representations in computers. We believe that the path connecting the legal text to logic programming requires intermediate steps. Thus an approach in legal informatics is identified which is called Multiphase Transformation (MuPT). Intermediaries are important, as additional goals can be revealed. We use the multiarch bridge metaphor as a form of knowledge visualization. Here symbolization precedes strict mathematical formalizations; the latter are set aside for the future. The paper is inspired by the work of Ken Satoh et al. who translated the Japanese Presupposed Ultimate Fact Theory (the JUF theory) into logic programming. We can characterize JUF as a ‘one-bridge’ theory. We present reflections on law as a whole, whereas the JUF theory specifically concerns civil law.

Introduction

The premise of this paper is based on the idea that it is (almost) impossible to proceed directly in one step from legal texts to logic programming, e.g. Prolog. We hold that intermediate steps are needed. Thus we advocate a ‘multibridge’ of some kind and address text-to-program transformations:

\[
\text{legal text} \rightarrow \text{program}
\]

where ‘program’ means a technical language – computer code (or simply code) referring to software and hardware. A one-bridge approach metaphor can be used to specify the task; see Fig. 1. The transformation translates legal rules into computer code.

A spectrum of works in legal informatics addresses this problem – in which different issues are tackled. The variety of approaches arises from different answers to the following questions, just to mention a few:

1. What is the volume of the legal text? A sentence, a section of 1 page, a law of 20 pages, a statute, a corpus of texts, or sources of the whole legal system?
2. What is the program: a few Prolog statements, a program of 1000 lines, or a country-wide information system?
Logic representation in the domain of law was addressed many decades ago by various scholars. Primarily we recall Ilmar Tammelo (1978), who was successful in representing (short) legal texts in the prefix notation of binary operators. However, whilst this is feasible in the case of a clear statement, difficulties arise with complex texts and a scalability problem is faced. Here we recall the early attempts of artificial intelligence research on understanding natural language. You can succeed in a world of toy blocks, but it is hardly feasible to represent the meaning in the general case. However, partial success can be reached. Florian Holzer (2010) considered the legal visualization of the German penal law. Grabmair and Ashley (2005) examined two transformations, firstly, the code (statute) text is transformed into an Intermediate Norm Representation, and then to a rulebase.

The present paper is inspired by the work of Ken Satoh et al. (2009) on the formalization of the Japanese Presupposed Ultimate Fact Theory, called the JUF theory. We characterize Satoh et al.’s approach as a ‘one-bridge’ theory.

We start advocating the multibridge approach with the observation that legal knowledge representation is needed as an intermediate step; see Fig. 2.

![Fig. 2: Legal knowledge representation is an intermediary stage. It is needed to build a multibridge from the legal text to program](image)

Following is a notation for the input/output chain:

$$\text{Legal text} \rightarrow \text{legal knowledge representation} \rightarrow \text{program}$$

The idea is to build several bridges to cross the “problem valley”. We emphasize the importance of intermediate steps; see Fig. 3. Hence the proposed approach can be termed *Multiphase Transformation* (MuPT).

![Fig. 3: The Multiphase Transformation approach – a multibridge. Intermediaries are important](image)

The four steps in Fig. 3 are represented by input → output pairs:

Step 1. *Microcontenting*: legal text → textual microcontents

Step 2. *Visualising*: textual microcontents → visualization

\[1\]
Step 3. *Formalising:* visualization $\rightarrow$ formalization

Step 4. *Representing:* formalization $\rightarrow$ representation in a technical language

The four steps can be worded in other ways or even divided into smaller steps. The boundaries of the intermediate steps may be blurred. A grey area can be divided into further sub-phases.

The text-to-program transformation introduced above is at the *parole* level. The transformation can be viewed in the *langue* metalevel context; see Fig. 4.

Fig. 4: Two levels – *parole* and *langue* – are comprised of entities and their rules

Therefore the intermediaries can also be viewed in the parole and langue levels; see Fig. 5.

Fig. 5: The Multiphase Transformation in the parole and langue context

Further we add an ontology level and language reference area level. The ontology level is comprised of the core ontology – how to build domain ontologies (Breuker et al. 2006) – and legal ontologies. Therefore the transformations are both multiphase and multilevel (Fig. 6).
The illustrations in this paper are experimental – a proof of the concept. They are intentionally not drawn very formally, however in the future it is also reasonable to develop a formal syntax for the explored and still open matters. Our visualizations are a form of knowledge representation, which is targeted at human beings rather than computers.

**On Approaches to the Transformation “Legal Text → Program”**

We distinguish the following kinds of program (Fig. 7):

1. IRS – *information retrieval systems*, also called IR systems (Schweighofer 1996)
2. KBS – *knowledge-based systems*. In the present context they are also called legal knowledge-based systems or legal expert systems (Leith 2010)
3. *Legal machines* such as traffic lights, cash machines, gates, barriers, etc.
4. *Information systems* (IS) such as enterprise resource planning (ERP) systems, management information systems (MIS), etc.
5. Other kinds of systems

Grey areas exist. IRS and KBS differ in knowledge representation in the sense of computer science. The boundary is not sharp since IRS are becoming more intelligent and contain KBS elements such as search engines and ontologies. In KBS, after the legal-to-program transformation, the pro-
gram does not necessarily become the law. Business rules (Vasilecas et al. 2009) in information systems represent different concepts, such as integrity constraints, epistemological, consolidation or derivation axioms, etc. They constrain database state transitions and can serve as enterprise policies. However, business rules do not necessarily have the spirit of legal rules that are viewed as human behavior rules in the ought-to-be world.

Programs

<table>
<thead>
<tr>
<th>Legal machines</th>
<th>Knowledge-based systems (KBS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose</strong>: regulation by program.</td>
<td><strong>Answer</strong>: an advice to the legal problem.</td>
</tr>
<tr>
<td><strong>Representation</strong>: legal text $\rightarrow$ program where «program is law».</td>
<td><strong>Representation</strong>: legal text $\rightarrow$ program where «program is not law».</td>
</tr>
<tr>
<td><strong>Purpose</strong>: implement (enforce) rules (primarily technical ones).</td>
<td><strong>Purpose</strong>: give advice.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Information retrieval systems (IRS)</th>
<th>Information systems (IS)</th>
<th>Other kinds of systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Answer</strong>: text corpora</td>
<td><strong>E.g., court case management systems</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Representation</strong>: legal source $\rightarrow$ e-document</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 7: A dichotomy that distinguishes between legal machines, KBS, IRS and IS

In contrast, in legal machines the program becomes law after the text-to-program transformation. This is the reason to concentrate on legal machines. They are purposed at regulation by computer code. Here we confer to the concept “[computer] code is law” as advocated by Lawrence Lessig (2006) in the context of cyberspace regulation.\(^1\)

Semantic information systems (Fill 2009) are also in our focus. However management information systems in legal domains, which do not introduce controversies with regard to text-to-program transformation, are not in the scope of the present research.

The semantics of legal knowledge, which is represented in a system, is a criterion to distinguish knowledge-based systems from information retrieval systems. The spectrum of semantic features is wide, and therefore this criterion is not sharp. It should be noted that search engines in KBS tend to become more intelligent; see a variety of search applications by Google search, www.google.com.

---

\(^1\) In real space, we recognize how laws regulate – through constitutions, statutes, and other legal codes. In cyberspace we must understand how a different “code” regulates – how the software and hardware (i.e., the “code” of cyberspace) that make cyberspace what it is also regulate cyberspace as it is. (Lessig 2006, p.5)
The spectrum of legal machines as defined in Fig. 7 is wide, therefore the boundary between legal machines and KBS is not sharp either. As an example, assume a taxpayer who violates tax law, the tax office IS that fines the taxpayer can be classified as both a legal machine and a KBS.

Building information retrieval addresses the transformation “legal source \(\rightarrow\) e-document”. For example, nowadays legal gazettes are being replaced by authentic electronic publications in databases and online systems such as RIS\(^2\) in Austria or EUR-Lex\(^3\) in the European Union. In IRS the answer to a query is legal text corpora. While in building IRS, the text-to-program transformation is not of primary importance.

In contrast, KBS face the “legal text \(\rightarrow\) program” problem as the query answer is a guidance to the legal problem. In a KBS, legal knowledge is represented in the program (a knowledge base + inference engine, for short). In this way, knowledge engineers meet the problem of understanding the contents of fundamental legal concepts, such as the nature of law, legal methods, legal interpretation, justice, values, etc. Although jurists know the contents, the notion of knowledge has different meanings in computer science and law. Wisdom differentiates knowledge in law. Ill-structured knowledge, vague norms and open texture are immanent to law. Therefore we take into account Leith’s (2010) scepticism about the value of expert systems in law. Underlying objections to expert systems are the “clear rule” idea and “the notion of being able to formalise knowledge through some logical or semi-logical formalism”.

A computational model of law is a temporary stop, while building the bridge from the legal domain to computer science. Thus, a vague legal rule is mapped to a strict rule in the model.

**Systems development life-cycle\(^4\) in software engineering.** Intermediate steps in the legal text-to-program transformation can follow the life-cycle stages of knowledge-based systems. The model of Mark Stefik (1995) in Fig. 8 identifies 5 stages: Identification, Conceptualization, Formalization, Implementation, and Testing. This model is a variation of the waterfall model\(^5\), Requirements-Design-Implementation-Verification-Maintenance, which is known in software engineering.

\(^2\) The Legal Information System of the Republic of Austria, a computer-assisted information system on Austrian law, http://www.ris.bka.gv.at/

\(^3\) http://eur-lex.europa.eu/ – Access to European Union law


\(^5\) http://en.wikipedia.org/wiki/Waterfall_model
Legalese within information systems. Each IS, independent of its intelligence, is relevant to law to a certain extent. In the identification and specification stages, the purposes of the IS are identified. These purposes have to comply with the legal order. Part of IS requirements can be classified as legal requirements. Hence, a bridge from law to informatics is starting to be built.

Legislative drafting. The importance of formalization can also be observed in legislative drafting and traceability (Fig. 9). In this context, Paliulionienė (2008) and Ėaplinskas (2009) used the frame logic (F-logic, Kifer et al. 1993) to represent legal documents in a knowledge base.

---

**Fig. 8:** Stages in the development of knowledge-based systems. This diagram emphasizes the development of knowledge in KBS from a product-oriented point of view (Stefik 1995, p. 350)

**Fig. 9:** Computer supported legislative drafting and traceability. The conceptual model of the legal act is tested for compliance with other acts (adapted from Paliulionienė (2008))
Intermediate Phases

A one-bridge approach is not always feasible. Intermediate structures serve as semantic bridges and are necessary. A one-arch bridge is impossible to build when the gap is too wide. Another case is when the “banks” are blurred. This occurs, for instance, when crossing wide “marshes”.

In Fig. 3 we identified the steps (phases) and called them Microcontenting, Visualising, Formalising, and Representing; they are described below. To cross the “valley” a team of experts in both law and informatics is needed.

Step 1 leads from the legal text to textual microcontents (Fig. 3). In this step, the linear legal text is divided into smaller portions that are organized, but not necessarily linearly. Steps 1 and 2 are explored, e.g. by Grabmair, legal informatics communities such as JURIX, IRIS, JURISIN, etc.

Step 2 takes the textual microcontents to visualization. This is concerned, for instance, by a whole research of Lachmayer⁶, see e.g. Garntschnig and Lachmayer (1979) and Jordan and Lachmayer (1987). Another example can be taken from Holzer (2010) concerning the German penal law; here two kinds of visualization are distinguished: (a) schematic (logical) and (b) situational (scenery) legal visualization. The schematic legal visualization structures the concepts, and the situational legal visualization represents the situation, its elements and relationships. The pictures thus obtained are not linear representations. Both methods are target at learning purposes. Opportunities and risks of the visualizations have also to be taken into account.

Step 3 leads from visualization to formalization; see e.g. Čyras (2010). This phase can be divided further. Symbolization is a good intermediate island.

Step 4 leads from formalization to knowledge representation in a computer. From a knowledge engineer’s view this step is a complex task that can also be divided into sub-phases. Various software engineering methodologies can be employed, see e.g. Čaplinskas (2009). The result is implementation in the programming language that you choose, such as Prolog, LKIF, etc.

Emergence of Intermediate Steps

Intermediary structures as semantic bridges emerge in the following themes: communication, tertium comparationis and abstraction. The latter concepts form a kind of ontology for intermediaries.

⁶ www.legalvisualization.com
In communication the transformation leads from one language to another. Thus the purpose is to build a bridge from the legal text bank to the technical language bank.

_Tertium comparationis_ (Latin – the third part of the comparison) is the quality that two things that are being compared have in common.\(^7\) This is the case of a relation which is not directly drawn from one element to another but through a third (Fig. 10 a).

In abstraction intermediate steps are also observed: first, by discerning parallel elements and comparing them, second, by seeing concrete examples such as sportsmen, politicians, actors, etc. Third, by producing schemes, fourth, through formal symbolizations, and fifth by developing ideas (Fig. 10 b).

---

**Example – Intermediate Phases in the JUF theory**

The one bridge of Satoh can be divided into two phases. The first is ‘formalising the JUF theory’; see Satoh et al. (2009, Section 2), where the JUF tree is an intermediate result. The second step is ‘translation into logic programming’ (Section 3), the result of which is a Prolog program (Fig. 11). It is not so important at the moment which logic programming language is used.

---

Satoh’s work concerns the non liquet (unknown truth value) problem in civil law. Jurists are familiar with assumption rules and the burden of persuasion – who has to prove what. The formalization is an attempt to invoke computer science to solve the non liquet truth value.

We can also identify the next steps: ‘developing the PROLEG system’ (Section 4) and ‘using the JUF theory’, in particular by judges. These steps would lead further away from the “other bank”.

Intermediate Steps in Context

Legal knowledge representation is not an ultimate goal – either in law or in legal informatics. Logic programming is just one task within the context. Logic programming can occur, for example, as an argument task in judiciary, as a law enforcement task in execution, or as a technical filter in legislation. Logical analysis of presumptions and burdens of proof in legal argument have been addressed, for example, by Prakken and Sartor (2008).

Legal knowledge representation appears in the context of other tasks. The four steps in (1) above can be identified and called differently. Therefore multi-arch bridges have neither a uniform arch number nor a unique construction.

Both knowledge representation and knowledge visualization frameworks have to answer key questions such as ‘what’, ‘why’ and ‘how’; see, e.g., Eppler and Burkhard (2006, p. 553), Čaplinskas (2009), and Zachman (1987):

1. Knowledge type (What? What type of knowledge is visualized (object)?)
2. Visualization goal (Why? Why should that knowledge be visualized (purpose)?)
3. Visualization format (How? How can the knowledge be represented (method)?)

The landscape of legal tasks can be subject to standardization similarly to software life-cycles. The questions including ‘who’, ‘where’ and ‘when’ have to be answered in software engineering in the requirements analysis phase. Therefore software life-cycle phases including specification, design and implementation can also be applied to legal tasks.

Legal rules are implemented in software in a form of technical rules. As a sample technical rule consider a bank machine’s rule: if a person provides the PIN code then the bank machine gives that person money. More complicated technical rules result, for instance, in the domain of role-based access to data and computer resources.

The four MuPT steps can be viewed as edges in the context that is depicted with the legal tasks network metaphor in Fig. 12. Here tasks are represented by edges, where each node represents a state of affairs and can
be named by a respective deliverable, and each edge represents a transition from one state to another.

![Diagram](image)

**Fig. 12: Intermediate steps are represented by edges in the context that is visualized with a task network metaphor**

**Reasons for Intermediate Steps**

One-bridge leading from law to legal expert systems was just a programmers’ dream some decades ago. However realizations face difficulties; see Leith’s (2010) critique.

**Obstacles for one-bridge.** We should mention the following difficulties:

1. The open texture problem has to be considered. Balancing in law is more important than rules.
2. Legal text allows different interpretations. Grammatical interpretation is not the sole method in law. The teleological, systemic and historical interpretations also have to be recalled.
3. A vast amount of legal knowledge is out of the text. Legal theory, doctrine, textbooks, etc. are the sources of legal knowledge, hence, explicit and implicit knowledge coexist. Technical domains are characterized by a smaller proportion of implicit and explicit knowledge.

Law as a phenomenon is rich in functions. Classifying legal tasks is out of scope of the present paper. A top-down classification can be started according to different criteria. The criteria can be taken from legal theory – the functions of law and their forms of realization.

The following lessons for intermediate phases can be learned from software engineering. One-arch technology such as writing a program from scratch is only feasible for small programs. For complicated programs usually technology is used which considers the whole software life-cycle.

**Way back and the V model.** The bridge concept is comprised of two paths: first from one side to the other and then back again. A one way only – as in a catapult – is enough in cases where you do not need to return; in this case you may burn the bridge. The to-and-return metaphor can be
compared with the V model (read as Vee model) in systems development (Forsberg 2005). The descending phases serve to implement the system, and the rising ones to verify and validate the requirements.

**Importance of islands.** Text-to-program transformation phases can be compared with building bridges through a marsh. A constructor builds a series of bridges to smaller or larger islands. An island is often an improvement to a city’s appearance: Seine & Island → Paris, Tiber & Island → Rome, Danube & Island → Vienna, etc. In the legal domain, document transformation needs team cooperation, therefore the islands serve as stops to produce intermediate deliverables.

The bridge metaphor consists of the following elements: (a) an initial situation – the “one bank”; (b) a target situation – the “other bank”; and (c) a connection between them – the “bridge”. Therefore the metaphor can be formalized in terms of a relationship and a graph edge: a start node \( u \), a target node \( v \), and the edge \((u,v)\).

### An Example of Observing Intermediate Steps

Contissa and Laukyte (2008) examine the legal knowledge representation of the French intellectual property law; we depict their experience as shown in Fig. 13. Legal texts serve as a bank from which to depart. The first island is a pseudo-code of the texts. The other bank is event calculus formalization in Prolog. The ontology of the intellectual property law which is represented in LKIF language backs the development. The Legal Knowledge Interchange Format (LKIF) is a Semantic Web based language designed for legal applications. It is built upon emerging XML-standards.

---

8 http://en.wikipedia.org/wiki/V-Model

9 http://en.wikipedia.org/wiki/Estrella_(project)
Conclusions

This paper presents personal reflections about the phases on the way to logic programming and legal knowledge representation. Each phase may have ill-defined inputs and outputs. Therefore a one-bridge approach to problem solving is hardly feasible.

Intermediaries appear within multistep and multilevel transformations: ontology – langue – parole – referential area. Intermediaries can be explained by the roles of communication, tertium comparationis and abstraction in the process of cognition.

Decomposing legal informatics tasks into a series of subtasks can provide opportunities to reveal additional purposes and rethink the outputs. This can contribute to legislative workflow.

Literature