LegalRuleML: Design Principles and Foundations

Tara Athan[1], Guido Governatori[2(✉)], Monica Palmirani[3], Adrian Paschke[4], and Adam Wyner[5]

[1] Athan Services, West Lafayette, USA
[2] NICTA Queensland, Brisbane, Australia
guido.governatori@nicta.com.au
[3] CIRSFID, University of Bologna, Bologna, Italy
[4] Corporate Semantic Web, Freie Universitat, Berlin, Germany
[5] University of Aberdeen, Aberdeen, UK

Abstract. This tutorial presents the principles of the OASIS LegalRuleML applied to the legal domain and discusses why, how, and when LegalRuleML is well-suited for modelling norms. To provide a framework of reference, we present a comprehensive list of requirements for devising rule interchange languages that capture the peculiarities of legal rule modelling in support of legal reasoning. The tutorial comprises syntactic, semantic, and pragmatic foundations, a LegalRuleML primer, as well as use case examples from the legal domain.

Keywords: LegalRuleML · RuleML · Legal rule modelling · Meta model

1 Introduction

The objective of the LegalRuleML Technical Committee (TC) is to extend RuleML with formal features specific to legal norms, guidelines, policies and reasoning; that is, the TC defines a standard (expressed with XML-schema and Relax NG) that is able to represent the particularities of the legal normative rules with a rich, articulated, and meaningful markup language. The features are:

- defeasibility of rules and defeasible logic;
- deontic operators (e.g., obligations, permissions, prohibitions, rights);
- semantic management of negation;
- temporal management of rules and temporality in rules;
- classification of norms (i.e., constitutive, prescriptive);
- jurisdiction of norms;
- isomorphism between rules and natural language normative provisions;

G. Governatori—NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

W. Faber and A. Paschke (Eds.): Reasoning Web 2015, LNCS 9203, pp. 151–188, 2015.
DOI: 10.1007/978-3-319-21768-0_6

– identification of parts of the norms (e.g., bearer, conditions);
– authorial tracking of rules.

Some matters are out of the scope of the TC and LegalRuleML such as specifications of core or domain legal ontologies. For the full motivation for LegalRuleML and its relationships with other approaches the reader is referred to [5,30].

The main principles of LegalRuleML are as follows.

Multiple Semantic Annotations: A legal rule may have multiple semantic annotations, where these annotations represent different legal interpretations. Each such annotation appears in a separate annotation collection as internal or external metadata. A range of parameters provide the interpretation with respect to provenance, applicable jurisdiction, logical interpretation of the rule, and others.

Tracking the LegalRuleML Creators: As part of the provenance information, a LegalRuleML document or any of its fragments can be associated with its creators. This is important to establish the authority and trust of the knowledge base and annotations. Among the creators of the document can be the authors of the text, knowledge base, and annotations, as well as the publisher of the document.

Linking Rules and Provisions: LegalRuleML includes a mechanism, based on IRI, that allows many to many (N:M) relationships among the rules and the textual provisions: multiple rules are embedded in the same provision, several provisions contribute to the same rule. This mechanism may be managed in the metadata collections, permitting extensible management, avoiding redundancy in the IRI definition, and avoiding errors in the associations.

Temporal Management: LegalRuleML's universe of discourse contains a variety of entities: provisions, rules, applications of rules, references to text, and references to physical entities. All of these entities exist and change in time; their histories interact in complicated ways. Legal RuleML represents these temporal issues in unambiguous fashion. In particular, a rule has parameters which can vary over time, such as its status (e.g., strict, defeasible, defeater), its validity (e.g., repealed, annulled, suspended), and its jurisdiction (e.g., only in EU, only in US). In addition, a rule has temporal aspects such as internal constituency of the action, the time of assertion of the rule, the efficacy, enforcement, and so on.

Formal Ontology Reference: LegalRuleML is independent from any legal ontology and logic framework. However it includes a mechanism, based on IRIs, for pointing to reusable classes of a specified external ontology.

LegalRuleML is Based on RuleML: LegalRuleML reuses and extends concepts and syntax of RuleML wherever possible, and also adds novel annotations. RuleML includes Reaction RuleML.

Mapping: LegalRuleML is mappable to RDF triples for Linked Data reuse.

2 Functionalities

Specifically, LegalRuleML facilitates the following functionalities.

(F1) Supports modelling different types of rules. There are constitutive rules, which define concepts or institutional actions that are recognised as such by virtue of the defining rules (e.g. the legal definition of "transfer property ownership"); and there are prescriptive rules, which regulate actions or the outcome of actions by making them obligatory, permitted, or prohibited.

(F2) Represents normative effects. There are many normative effects that follow from applying rules, such as obligations, permissions, prohibitions, and more articulated effects. Rules are also required to regulate methods for detecting violations of the law and to determine the normative effects triggered by norm violations, such as reparative obligations, which are meant to repair or compensate violations. These constructions can give rise to very complex rule dependencies, because the violation of a single rule can activate other (reparative) rules, which in turn, in case of their violation, refer to other rules, and so forth.

(F3) Implements defeasibility [13,31,34]. In the law, where the antecedent of a rule is satisfied by the facts of a case (or via other rules), the conclusion of the rule presumably, but not necessarily, holds. The defeasibility of legal rules consists of the means to identify exceptions and conflicts along with mechanisms to resolve conflicts.

(F4) Implements isomorphism [7]. To ease validation and maintenance, there should be a one-to-one correspondence between collections of rules in the formal model and the units of (controlled) natural language text that express the rules in the original legal sources, such as sections of legislation.

(F5) Alternatives: often legal documents are left ambiguous on purpose to capture open–ended aspects of the domain they are intended to regulate. At the same time legal documents are meant to be interpreted by end users. This means that there are cases where multiple (and incompatible) interpretations of the same textual source are possible. LegalRuleML offers mechanisms to specify such interpretations and to select one of them based on the relevant context.

(F6) Manages rule reification [13]. Rules are objects with properties, such as Jurisdiction, Authority, Temporal attributes [21,22,29]. These elements are necessary to enable effective legal reasoning.

3 Criteria of Good Language Design

The syntax design should follow from semantic intuitions from the subject matter domain - labelling entities, properties, and relations as well as some of the type constraints amongst them that guide how the labels are combined and used.

Criteria of Good Language Design are:

– Minimality, which requires that the language provides only a small set of needed language constructs, i.e., the same meaning cannot be expressed by different language constructs.

- Referential transparency, which means that the same language construct always expresses the same semantics regardless of the context in which it is used.
- Orthogonality, where language constructs are independent of each other, thus permitting their systematic combination.
- Pattern-based design, where design patterns are a distillation of common wisdom in organizing the structural parts, the grammar and the constraints of a language. Some of them are listed in [9] and as XML Patterns[1]. Inside of LegalRuleML we introduce five design patterns.
- Meta-model based, where the meta-model for a language, also called the abstract syntax, defines the vocabulary for describing the language, including syntactic categories.

LegalRuleML was designed based on such principles. In particular its vocabulary is inspired by terms from the legal domain, which then facilitates its use by users familiar with the domain.

The LegalRuleML meta-model captures the common meaning of domain terms as understood in the legal field, formalizes the connections among the various concepts and their representation in the language, and provides an RDF-based abstract syntax. RDFS [8] is used to define the LegalRuleML metamodel, and graphs of the RDFS schemas accompany the following discussions about the domain concepts.[2]

4 Modelling Norms

According to scholars of legal theory [34], norms can be represented by rules with the form

$$\text{if } A_1, ..., A_n \text{ then } C$$

where $A_1, \ldots, A_n$ are the pre-conditions of the norm, C is the effect of the norm, and *if ... then ...* is a normative conditional, which are generally defeasible and do not correspond to the if-then material implication of propositional logic. Norms are meant to provide general principles, but at the same time they can express exceptions to the principle. It is well understood in Legal Theory [14,34] that, typically, there are different types of "normative conditionals", but in general normative conditionals are defeasible. Defeasibility is the property that a conclusion is open in principle to revision in case more evidence to the contrary is provided. Defeasible reasoning is in contrast to monotonic reasoning of propositional logic, where no revision is possible. In addition, defeasible reasoning allows reasoning in the face of contradictions, which gives rise to *ex falso quodlibet* in propositional logic. One application of defeasible reasoning is the ability to model exceptions in a simple and natural way.

[1] http://www.xmlpatterns.com/.
[2] https://tools.oasis-open.org/version-control/browse/wsvn/legalruleml/trunk/schemas/rdfs/#_trunk_schemas_rdfs_.

4.1 Defeasibility

The first use of defeasible rules is to capture conflicting rules/norms without making the resulting set of rules inconsistent. Given that $\neg expression$ means the negation of *expression*, the following two rules conclude with the negation of each other

$$body_1 \Rightarrow head$$
$$body_2 \Rightarrow \neg head$$

Without defeasibile rules, rules with conclusions that are negations of each other could give rise, should $body_1$ and $body_2$ both hold, to a contradiction, i.e., *head* and $\neg head$, and consequently *ex falso quodlibet*. Instead, defeasible reasoning is sceptical; that is, in case of a conflict such as the above, it refrains from taking any of the two conclusions, unless there are mechanisms to solve the conflict (see the discussion below on the superiority relation). Notice that an application of this is to model exceptions. Exceptions limit the applicability of basic norms/rules, for example:

$$body \Rightarrow head$$
$$body, exception\_condition \Rightarrow \neg head$$

In this case, the second rule is more specific than the first, and thus it forms an exception to the first, i.e., a case where the rule has extra conditions that encode the exception, blocking the conclusion of the first rule. Often, exceptions in defeasible reasoning can be simply encoded as

$$body \Rightarrow head$$
$$exception\_condition \Rightarrow \neg head$$

In the definition of rules as normative conditionals made up of preconditions and effect, we can see a rule as a binary relationship between the set of preconditions (or body or antecedent) of the rule, and the (legal) effect (head or conclusion) of the rule. Formally, a rule can be defined by the following signature:

$$body \times head$$

We can then investigate the nature of such a relationship. Given two sets, we have the following seven possible relationships describing the "strength" of the connections between the body and the head of a rule:

body always *head*
body sometimes *head*
body not complement *head*
body no relationship *head*
body always complement *head*
body sometimes complement *head*
body not *head*

In defeasible logic we can represent the relationships using the following formalisation of rules (rule types):

$$body \rightarrow head$$
$$body \Rightarrow head$$
$$body \rightsquigarrow head$$
$$body \rightarrow \neg head$$
$$body \Rightarrow \neg head$$
$$body \rightsquigarrow \neg head$$

There is no need to have a rule for the case where there is no relationship between the head and the body. The following table summarises the relationships, the notation used for them, and the strength of the relationship.[3]

body always *head*	$body \rightarrow head$	Strict rule
body sometimes *head*	$body \Rightarrow head$	Defeasible rule
body not complement *head*	$body \rightsquigarrow head$	Defeater
body no relationship *head*		
body always complement *head*	$body \rightarrow \neg head$	Strict rule
body sometimes complement *head*	$body \Rightarrow \neg head$	Defeasible rule
body not *head*	$body \rightsquigarrow \neg head$	Defeater

The meaning of the different types of rules is as follows:

For a *strict rule* $body \rightarrow head$ the interpretation is that every time the body holds then the head holds.

For a *defeasible rule* $body \Rightarrow head$ the reading is when the body holds, then, typically, the head holds. Alternatively we can say that the head holds when the body does unless there are reasons to assert that the head does not hold. This captures that it is possible to have exceptions to the rule/norm, and it is possible to have prescriptions for the opposite conclusion.

For a *defeater* $body \rightsquigarrow head$ the intuition is as follows: defeaters are rules that cannot establish that the head holds. Instead they can be use to specify that the opposite conclusion does not hold. In argumentation two types of defeaters are recognized: defeaters used when an argument attacks the preconditions of another argument (or rule); other defeaters used when there is no relationship between the premises of an argument (preconditions of a rule or body) and the conclusion of the argument (effect of the rule or head).

Given the possibility to have conflicting rules, i.e., rules with opposite or contradictory heads, we have, for example

$$body_1 \Rightarrow head$$
$$body_2 \Rightarrow \neg head$$

Systems for defeasible reasoning include mechanisms to solve such conflicts. Different methods to solve conflicts have been proposed: *specificity*, *salience*, and

[3] The syntax presented here is based on Defeasible Logic, see [4,27].

a *preference relation*. According to specificity, in case of a conflict between two rules, the most specific rule prevails over the less specific one, where a rule is more specific if its body subsumes the body of the other rule. For salience, each rule has an attached salience or weight, where in case of a conflict between two rules, the one with the greatest salience or weight prevails over the other. A preference relation (also known as superiority relation) defines a binary relation over rules, where an element of the relation states the relative strength between two rules. Thus, in case of a conflict between two rules, if the preference relation is defined over such rules, the strongest of the two rules wins over the other.

Various researchers have taken different views on such methods. Specificity corresponds to the well know legal principle of *lex specialis*. [32] argues that specificity is not always appropriate for legal reasoning and that there are other well understood legal principles such as *lex superior* and *lex posterior* that apply instead. [32] cites cases in which the *lex specialis* principle might not be the one used to solve the conflict, for example, a more specific article from a local council regulation might not override a less specific constitutional norm. [32] proposes to use a dynamic preference relation to handle conflicting rules. The preference relation is dynamic in the sense that it is possible to argue about which instances of the relation hold and under which circumstances. [3] proposes that instances of the superiority relation appear in the head of rules, namely:

$$body \Rightarrow superiority$$

where *superiority* is a statement with the form

$$r_1 > r_2$$

where r_1 and r_2 are rule identifiers.

Reference [12] proposes Carneades as a rule-based argumentation system suitable for legal reasoning which uses weights attached to the arguments (rules) to solve conflicts and to define proof standards. [17] shows how to use the weights to generate an equivalent preference relation, and, consequently, how to capture the proposed proof standards. In addition, [17] shows that there are situations where a preference relation cannot be captured by using weights on the rules.

To handle defeasibility, LegalRuleML has to capture the superiority relation and the strength of rules. For the superiority relation, LegalRuleML offers the element `<Overrides>`, which defines a relationship of superiority `cs2` overrides `cs1`, where `cs2` and `cs1` are Legal Statement identifiers.[4] These elements are included through `hasQualification` roles.

```
<lrml:hasQualification>
  <lrml:Overrides over="#cs1" under="#cs2"/>
</lrml:hasQualification>
```

For the representation of the strength of rules, LegalRuleML has two options:

The first is to include it in a `<Context>` element, where a `<Context>` specifies a context in which the rule is applied:

[4] LegalRuleML defines a Legal Statement as an expression of a Legal Rule or a part of a Legal Rule where a Legal Rule is a formal representation of a Legal Norm.

```
<lrml:Context key="ruleInfo2">
  <lrml:appliesStrength iri="deovo:defeasible2"/>
  <lrml:toStatement keyref="#cs1"/>
</lrml:Context>
```

The second (and optional) way to express the qualification of the rule is directly inside of the rule, through a `hasStrength` role. The difference is that `<Context>` localises the strength of a rule, while `hasStrength` in effect relates the strength to the rule in all contexts:

```
<lrml:hasStrength>
  <lrml:Defeater key="str4"/>
</lrml:hasStrength>
```

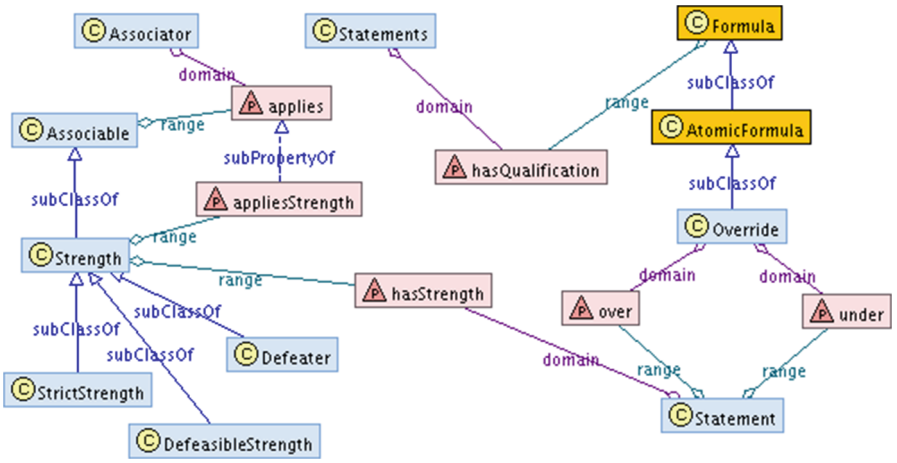


Fig. 1. Partial Metamodel for Defeasible Concepts. LegalRuleML and RuleML classes are labelled with a 'C' in a circle, LegalRuleML properties with a 'P' in a triangle. The Formula and AtomicFormula classes are imported from RuleML.

4.2 Constitutive and Prescriptive Norms

As we have discussed, a Legal Rule can be seen as binary relationship between its antecedent (a set of formulas, encoding the pre-conditions of a norm, represented in LegalRuleML by a formula, where multiple pre-conditions are joined by some logical connective) and its conclusion (the effect of the norm, represented by a formula). It is possible to have different types of relations. In the previous section, we examined one such aspect: the strength of the link between the antecedent and the conclusion. Similarly, we can explore a second aspect, namely what type of effect follows from the pre-condition of a norm. In Legal Theory norms are classified mostly in two main categories: constitutive norms and prescriptive norms, which will be then represented as constitutive rules (also

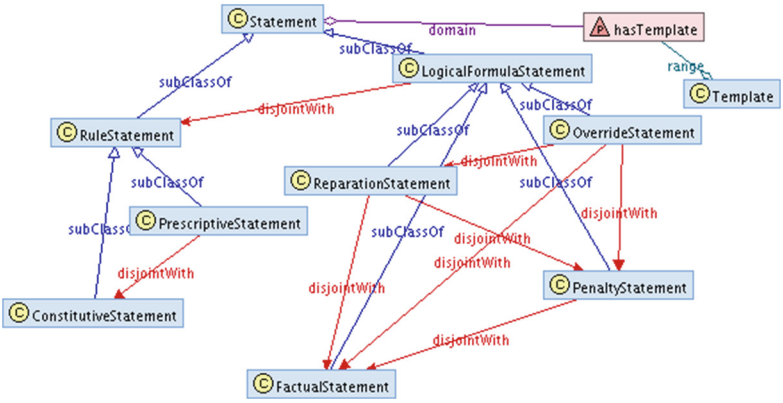


Fig. 2. Partial Metamodel for Statement Subclasses.

known as *counts-as rules*) and prescriptive rules.[5] The (partial) meta-model for the notions described in this section is depicted in Fig. 2.

The function of constitutive norms is to define and create so called institutional facts [36], where an institutional fact is how a particular concept is understood in a specific institution. Thus, constitutive rules provide definitions of the terms and concepts used in a jurisdiction. On the other hand, prescriptive rules dictate the obligations, prohibitions, permissions, etc. of a legal system, along with the conditions under which the obligations, prohibitions, permissions, etc. hold. LegalRuleML uses deontic operators to capture such notions (see Sect. 4.3). Deontic operators are meant to qualify formulas. A Deontic operator takes as its argument a formula and returns a formula. For example, given the (atomic) formula *PayInvoice*(*guido*), meaning 'Guido pays the invoice', and the deontic operator [OBL] (for obligation), the application of the deontic operator to the formula generates the new (deontic) formula [OBL]*PayInvoice*(*guido*), meaning that "it is obligatory that Guido pays the invoice".

The following is the LegalRuleML format for prescriptive rules. Notice, that in LegalRuleML, legal rules are captured by the broader class of Statement and the `hasTemplate` property links a prescriptive or constitutive statement (see Fig. 2 for the different types of statements available in LegalRuleML) to its template, a fragment of RuleML syntax with root `ruleml:Rule` that denotes a class of rules.

```
<lrml:PrescriptiveStatement key="ps1">
  <lrml:hasTemplate>
    <ruleml:Rule key=":key1">
      <lrml:hasStrength>
        strength of the rule
      </lrml:hasStrength>
      <ruleml:if>
```

[5] Reference [14] identify more types of norms/rules. However, most of them can be reduced to the two types described here insofar as the distinction is not on structure of the rules but it depends on the meaning of the content (specific effect) of the rules, while keeping the same logical format.

```
        formula, including deontic formula
      </ruleml:if>
      <ruleml:then>
        <lrml:SuborderList>
          list of deontic formulas
        </lrml:SuborderList>
      </ruleml:then>
    </ruleml:Rule>
  </lrml:hasTemplate>
</lrml:PrescriptiveStatement>
```

The difference between constitutive rules and prescriptive rules is in the content of the head, where the head of a prescriptive rule is a list of deontic formulas which is called a suborder list (see Sect. 4.3 below), and represented in LegalRuleML by the `<lrml:Suborder>` element. Syntactically, a suborder list of one element can be rendered in LegalRuleML as just the element. Prescriptive and constitutive rules can have deontic formulas in their set of preconditions (antecedent or body). The conclusion (head) of a constitutive rule cannot be a deontic formula, nor can it be a compound formula that contains a deontic formula.

```
<lrml:ConstitutiveStatement key="ps1">
  <ruleml:Rule key=":key1">
    <lrml:hasStrength>
      strength of the rule
    </lrml:hasStrength>
    <ruleml:if>
       formula, including deontic formula
    </ruleml:if>
    <ruleml:then>
       non-deontic formula
    </ruleml:then>
  </ruleml:Rule>
</lrml:ConstitutiveStatement>
```

4.3 Deontic

One of the functions of norms is to regulate the behaviour of their subjects by imposing constraints on what the subjects can or cannot do, what situations are deemed legal, and which ones are considered to be illegal. There is an important difference between the constraints imposed by norms and other types of constraints. Typically a constraint means that the situation described by the constraint cannot occur. For example, the constraint A means that if $\neg A$ (the negation of A, that is, the opposite of A) occurs, then we have a contradiction, or in other terms, we have an impossible situation. Norms, on the other hand, can be violated. Namely, given a norm that imposes the constraint A, yet we have a situation where $\neg A$, we do not have a contradiction, but rather a violation, or in other terms we have a situation that is classified as "illegal". From a logical point of view, we cannot represent the constraint imposed by a norm simply by A, since the conjunction of A and $\neg A$ is a contradiction. Thus we need a mechanism to identify the constraints imposed by norms. This mechanism is provided by modal operators, in particular, deontic operators.

Modal and Deontic Operators. Modal logic is an extension of classical logic with modal operators. A modal operator applies to a proposition to create a new proposition. The meaning of a modal operator is to "qualify" the truth of the proposition that the operator applies to. The basic modal operators are those of *necessity* and *possibility*. Accordingly, given a proposition p expressing, for example that "the snow is white" and the necessity modal operator [NEC], [NEC]p is the proposition expressing that "necessarily the snow is white". Typically, the necessity and possibility operators are the dual of each other, namely:

$$[\text{NEC}]p \equiv \neg[\text{POS}]\neg p$$
$$[\text{POS}]p \equiv \neg[\text{NEC}]\neg p$$

The modal operators have received different interpretations: for example, necessity can be understood as logical necessity, physical necessity, epistemic necessity (knowledge), doxastic necessity (belief), temporal necessity (e.g., always in the future), deontic necessity (obligatory), and many more.

In the context of normative reasoning and representation of norms the focus is on the concepts of deontic necessity and deontic possibility. These two correspond to the notions of Obligation, and Permission. In addition, we consider the notion of Prohibition, which corresponds to the operator of deontic impossibility. For something to be "deontically necessary" means that it holds in all situations deemed legal; similarly something is "deontically possible" if there is at least one legal state where it holds. Finally, "deontically impossible" indicates that something does not hold in any legal state. More specifically a legal state is a state where there are no violations. Thus LegalRuleML defines Obligation as a Deontic Specification[6] for a state, an act, or a course of action to which a Bearer is legally bound, and which, if it is not achieved or performed, results in a violation; similarly a Prohibition is a Deontic Specification for a state, an act, or a course of action to which a Bearer is legally bound, and which, if it is achieved or performed, results in a violation. A Permission is a Deontic Specification indicating that the Bearer has no Obligation or Prohibition to the contrary.

We will use [OBL] for the modal/deontic operator of Obligation, [PERM] for Permission, and [FOR] for Prohibition (or Forbidden).

Standard deontic logic assumes the following relationships between the operators:

$$[\text{OBL}]p \equiv \neg[\text{PERM}]\neg p$$

If p is obligatory, then its opposite, $\neg p$, is not permitted.

$$[\text{FOR}]p \equiv [\text{OBL}]\neg p$$

If p is forbidden then its opposite is Obligatory. Alternatively, a Prohibition can be understood as Obligation of the negation.

The following is an example of mathematical statement of a Prescriptive Rule:

$$p_1, \ldots, p_n, [\text{DEON}_1]p_{n+1}, \ldots, [\text{DEON}_m]p_{n+m} \Rightarrow [\text{DEON}]q$$

[6] Deontic Specification is the class that includes the various deontic notions used in LegalRuleML.

The antecedent, $p_1, \ldots, p_n, [\text{DEON}_1]p_{n+1}, \ldots, [\text{DEON}_m]p_{n+m}$, conditions the applicability of the norm in the consequent $[\text{DEON}]q$; that is, when the antecedent conditions are met, then the consequent is the *deontic* effect of them. Thus, given the antecedent, the rule implies $[\text{DEON}]q$.

The operators of Obligation, Prohibition and Permission are typically considered the basic ones, but further refinements are possible, for example, two types of permissions have been discussed in the literature on deontic logic: *weak permission* (or *negative permission*) and *strong permission* (or *positive permission*). Weak permission corresponds to the idea that some A is permitted if $\neg A$ is not provable as mandatory. In other words, something is allowed by a code only when it is not prohibited by that code [38]. The concept of strong permission is more complicated, as it amounts to the idea that some A is permitted by a code if and only if such a code explicitly states that A is permitted, typically as an exception to the prohibition of A or the obligation of its contrary, i.e., $\neg A$. It follows that a strong permission is not derived from the absence of a prohibition, but is explicitly formulated in a permissive (prescriptive) norm [2]. An example of an explicit permissive norm is manifested by a "U-turn permitted" sign exposed at a traffic light, which derogates the (general) prohibition to U-turn at traffic lights.

Refinements of the concept of obligation have been proposed as well. For example it is possible to distinguish between *achievement* and *maintenance* obligations, where an *achievement* obligation is an obligation that is fulfilled if what the obligation prescribes holds at least once in the period when the obligation holds, while a *maintenance* obligation must be obeyed for all the instants when it holds (see [18] for a classification of obligations).

LegalRuleML is neutral about the different subclasses of the deontic operators; to this end LegalRuleML is equipped with a mechanism to point to the semantics of various Deontic Specifications in a document. The first mechanism is provided by the `iri` attribute of a Deontic Specification for example:

```
<lrml:Obligation
    key="oblig1"
    iri="http://example.org/deontic/vocab#achievementobligation">
    ...
</lrml:Obligation>
```

The second alternative is to use an `Association` to link a Deontic Specification to its meaning using the `applyModality` element, namely:

```
<lrml:Association>
  <lrml:appliesModality
      iri="http://example.org/deontic/vocab#maintenanaceobligation"/>
  <lrml:toTarget keyref="#oblig101"/>
</lrml:Association>‘
```

Furthermore, Obligations, Prohibitions and Permissions in LegalRuleML are directed operators [24], thus they have parties (e.g. Bearer), specifying, for example, who is the subject of an Obligation or who is the beneficiary of a Permission.

```
<lrml:Obligation iri="http://example.org/deontic/vocab#obl1">
  <ruleml:slot>
    <lrml:Bearer iri="http://example.org/deontic/vocab#oblBearer"/>
```

```
      <ruleml:Ind>Y</ruleml:Ind>
    </ruleml:slot>
    <ruleml:Atom key=":atom2">
      <ruleml:Rel iri="#rel2"/>
      <ruleml:Ind>X</ruleml:Ind>
    </ruleml:Atom>
  </lrml:Obligation>
```

Violation, Suborder, Penalty and Reparation. Obligations can be violated; according to some legal scholars, the possibility of being violated can be used to define an obligation. A violation means that the content of the obligation has not been met. It is important to notice that a violation does not result in an inconsistency. A violation is, basically, a situation where we have

$$([\text{OBL}]p) \text{ and } \neg p$$

One of the characteristics of norms is that having violated them, a penalty can be introduced to compensate for the violation, where a penalty is typically a Deontic Specification. To model this feature of norms and legal reasoning [20] introduced what is called here a suborder list, and [16] showed how to combine them with defeasible reasoning for the modelling of (business) contracts. As we have mentioned above, a suborder list is a list of deontic formulas, e.g., formulas of the form [D]A, where [D] is one of [OBL] (Obligation), [FOR] (Prohibition, or forbidden), [PERM] (Permission) and [RIGHT] (Right). Syntactically, a suborder list of one element can be rendered in LegalRuleML as just the element. To illustrate the meaning of suborder lists, consider the following example:

$$[\text{OBL}]A, [\text{OBL}]B, [\text{FOR}]C, [\text{PERM}]D$$

The expression means that A is obligatory, but if it is violated, i.e., we have its opposite $\neg A$, then the obligation comes into force to compensate for the violation of [OBL]A with [OBL]B. If also this Obligation of B is violated, then we have [FOR]C, the Prohibition of C. At this stage, if we have a Violation of such a Prohibition, i.e., we have C, then the Permission of D kicks in. Obligations and Prohibitions should not be preceded by Permissions and Rights in a suborder list, for the semantics of suborder lists is such that an element holds in the list only if all the elements that precede it in the list have been violated. It is not possible to have a Violation of a Permission, so it cannot serve a purpose in the suborder list. Accordingly, an element following a permission in a suborder list would never hold. See [19] for a full discussion on the issue of permissions and suborder lists. [16,20] also discuss mechanisms to combine the suborder lists from different rules. For example, given the rules

$$\begin{aligned} \mathit{body} &\Rightarrow [\text{OBL}]A \\ \neg A &\Rightarrow [\text{OBL}]B \end{aligned}$$

Here the body of the second rule is the negation of the content of the obligation in the head of the first rule. It is possible to merge the two rules above in the following rule

$$\mathit{body} \Rightarrow [\text{OBL}]A, [\text{OBL}]B$$

stating that one compensates for the violation of the obligation of A with the obligation of B. This suggests that suborder lists provide a simple and convenient mechanism to model penalties. It is not uncommon for a legal text (e.g., a contract) to include sections about penalties, where one penalty is provided as compensation for many norms. To model this and to maintain the isomorphism between a source and its formalisation, LegalRuleML includes a `<PenaltyStatement>` element, the scope of which is to represent a statement of a penalty as a suborder list (including the trivial non-empty list of a single element).

```
<lrml:PenaltyStatement key="pen1">
  <lrml:SuborderList>
    list of deontic formulas
  </lrml:SuborderList>
</lrml:PenaltyStatement>
```

LegalRuleML not only models penalties, but aims to connect the penalty statement with the corresponding `Reparation` element:

```
<lrml:Reparation key="rep1">
  <lrml:appliesPenalty keyref="#pen1"/>
  <lrml:toPrescriptiveStatement keyref="#ps1"/>
</lrml:Reparation>
```

With the temporal model of LegalRuleML (see Sect. 5.4), we can model a unique prescriptive statement (e.g., a prohibition) and several penalties that are updated over time according to the modifications of the law. Dynamically, the legal reasoner can point out the correct penalty according to the time of the crime (e.g., an obligation to pay statutory damage $500 in 2000, $750 in 2006, $1000 in 2010).

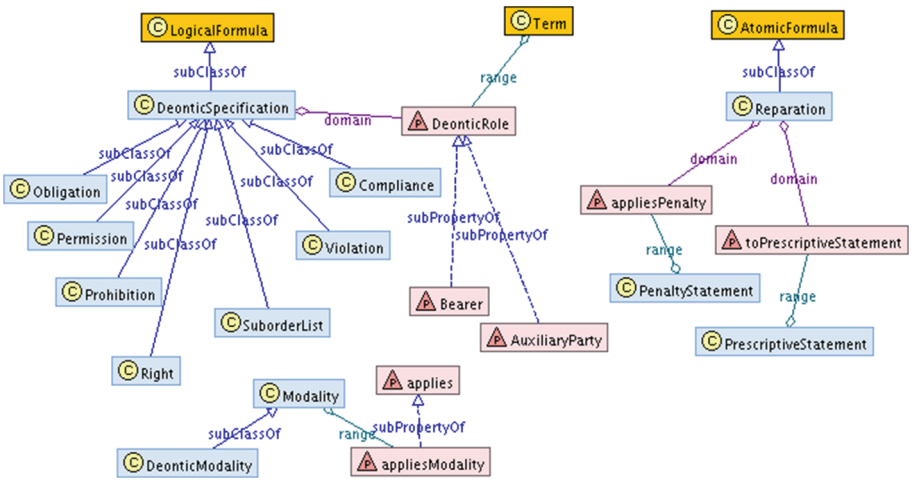
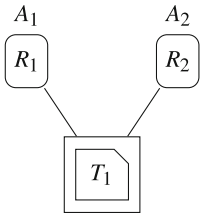


Fig. 3. Partial Metamodel for Deontic Concepts. LogicalFormula, Term and AtomicFormula classes are imported from RuleML.

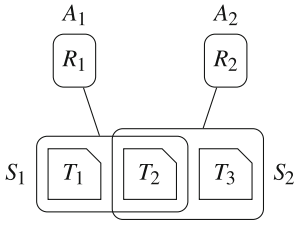
4.4 Alternatives

In the legal interpretation theory [37] norms are interpreted by the judges in order to apply them to the concrete cases. Sometime the legal interpretation theories conflict and diverge from each other [11,23,33]. Linguistic elements are added to this also for different reasons such as jurisdiction (e.g., national and regional level) or for competences (e.g., civil or criminal court). The practice of law over time has developed its own catalogue of *hermeneutical principles*, a range of techniques to interpret the law, such as catalogued and discussed in [35]. In addition, in Linguistics, issues about interpretation have long been of central concern (see among others [10,26]), where the need for interpretation arises given that the meanings (broadly construed) of "linguistic signs", (e.g., words, sentences, and discourses), can vary depending on participants, context, purpose, and other parameters. Interpretation is, then, giving the meaning of the linguistic signs for a given set of parameters.

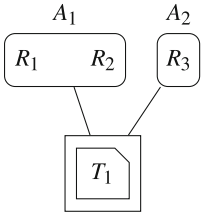
LegalRuleML endeavours not to account for how different interpretations arise, but to provide a mechanism to record and represent them. We have four different templates:



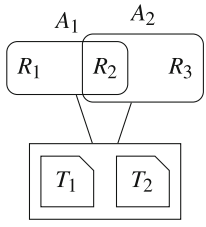
Case 1: Same legal provision(s), T_1, and different alternatives (A_1 and A_2).



Case 2: Different alternatives (A_1 and A_2) that share one or more pieces of text, T_2, but others are not shared (T_1 and T_3).



Case 3: Different alternatives (A_1 and A_2) sharing the same legal provision(s) (T_1), but embedding different rules (R_1 and R_2 for A_1 and R_3 for A_1).



Case 4: Different alternatives that share the same legal provision(s), but one or more rules are in common (e.g., R_2).

The element `<lrml:Alternatives>` permits to express all these interpretation templates. The following LegalRuleML fragments illustrate how to represent the four cases above (the first case shows the normalized serialization, while the rest show the compact serialization).

Case 1:

```
<lrml:Alternatives key="alt1">
  <lrml:fromLegalSources>
    <lrml:LegalSources>
      <lrml:hasLegalSource keyref="#ref1"/>
    </lrml:LegalSources>
  </lrml:fromLegalSources>
  <lrml:hasAlternative keyref="#ps1"/>
  <lrml:hasAlternative keyref="#ps2"/>
</lrml:Alternatives>
```

Case 2:

```
<lrml:Alternatives key="alt2">
  <lrml:LegalSources>
    <lrml:hasLegalSource keyref="#ref1"/>
    <lrml:hasLegalSource keyref="#ref2"/>
  </lrml:LegalSources>
  <lrml:hasAlternative keyref="#ps1"/>
  <lrml:hasAlternative keyref="#ps2"/>
</lrml:Alternatives>
```

Case 3:

```
<lrml:Alternatives key="alt3">
  <lrml:LegalSources>
    <lrml:hasLegalSource keyref="#ref1"/>
  </lrml:LegalSources>
  <lrml:hasAlternative keyref="#ss1"/>
  <lrml:hasAlternative keyref="#ss2"/>
</lrml:Alternatives>

<lrml:Statements key="ss1">
  <lrml:ConstitutiveStatement keyref="#ps1"/>
  <lrml:ConstitutiveStatement keyref="#ps2"/>
</lrml:Statements>

<lrml:Statements key="ss2">
  <lrml:ConstitutiveStatement keyref="#ps3"/>
</lrml:Statements>
```

Case 4:

```
<lrml:Alternatives key="alt3">
  <lrml:LegalSources>
    <lrml:hasLegalSource keyref="#ref1"/>
    <lrml:hasLegalSource keyref="#ref2"/>
  </lrml:LegalSources>
  <lrml:hasAlternative keyref="#ss1"/>
  <lrml:hasAlternative keyref="#ss2"/>
</lrml:Alternatives>

<lrml:Statements key="ss1">
  <lrml:ConstitutiveStatement
      keyref="#ps1"/>
  <lrml:ConstitutiveStatement
```

```
      keyref="#ps2"/>
</lrml:Statements>

<lrml:Statements key="ss2">
  <lrml:ConstitutiveStatement
      keyref="#ps1"/>
  <lrml:ConstitutiveStatement
      keyref="#ps3"/>
</lrml:Statements>
```

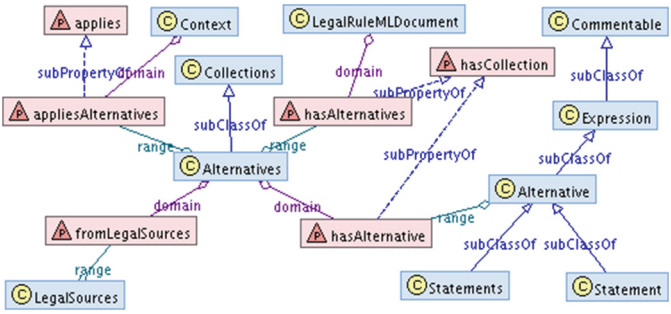


Fig. 4. Partial Metamodel for Alternatives Concepts.

A possible use of the LegalRuleML alternatives mechanism is in legal disputes where the alternatives can be used to model the (different) interpretations of a piece of legislation by the parties involved in the dispute; a comprehensive illustration of this is provided in Sect. 8 based on [6].

5 Meta Data of the Norms

5.1 Sources and Isomorphism

For legal rule modelling, it is important to maintain the connection between the formal norms and the legally binding textual statements that express the norms for several reasons. Legal knowledge engineers and end users should know and be able to track the textual source of the formal representation. Furthermore, because the legal text is the only legally binding element, the connection between text and the rule(s) (or fragment of rule) guarantees the provenance, authoritativeness, and authenticity of the rules modelled by the legal knowledge engineer. In addition, legal experts (judges, lawyers, legal operators) request a mechanism to connect text and rules for legibility and validation of the rules. Finally, because the legal sources of rules change over time, the formal rules need to be updated according to the textual changes; as there is usually no automatic mechanism to correlate and track modifications to rules, the connection between text and rules helps to do so. For these reasons LegalRuleML

includes a mechanism for managing this connection, which is called "isomorphism" in the AI & Law community. The mechanism must support a fine granularity (rules, fragments of rules, atoms, fragments of atoms connected with provisions, fragments of provisions, letters, numbers, paragraphs, sentences, and words) as well as to represent temporal modifications. LegalRuleML dedicates two collections (`<lrml:References>`, `<lrml:LegalSources>`) to annotate the original legal sources. In Sect. 6 the mechanism for creating an N:M relationship with rules (e.g., many rules associated with one textual provision; many legal source fragments for one rule) will be described.

`<lrml:References>` is the collection dedicated to record non-IRI based identifier sources, and the attribute `refIDSystemName` is able to annotate the naming convention used. In the following example we refer to the Akoma Ntoso relative IRI of the section 504 of the US Code, following the naming convention of the XML-schema in Akoma Ntoso[7]:

```
<lrml:References refType="http://example.legalruleml.org/lrml#LegalSource">
  <lrml:Reference
      refersTo="ref1"
      refID="/akn/us/act/uscode/eng@/main#title17-chp5-sec504-clsa-lst1-pnt1"
      refIDSystemName="AkomaNtoso3.0-2015-04-16"/>
</lrml:References>
```

`<lrml:LegalSource>` is the construct dedicated to record the IRI based identifier sources. The following example define the source of the U.S. Code, section 504, paragraph 1, title 17 published in the Cornell University portal http://www.law.cornell.edu/:

```
<lrml:LegalSources>
  <lrml:LegalSource
    key="ref2"
    sameAs="http://www.law.cornell.edu/uscode/text/17/504#psection-1"/>
  </lrml:LegalSource>
</lrml:LegalSources>
```

The list of the resources connected with the legal rules that are modelled in a LegalRuleML document are defined once in the first part of the XML file. This minimizes redundant definitions of the resources and avoids errors. As we see later, using the attribute value specified in `@key`, rules (or fragments of a rule) can be connected to References or LegalSources. The `<lrml:Association>` construct links LegalSources and References with rules (or fragment of rule), thus implementing the N:M relationship.

[7] Akoma Ntoso is an XML vocabulary for representing legal, legislative, parliamentary and judiciary documents in a structured and semantic manner. Akoma Ntoso is managed by the LegalDocML TC of OASIS. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=legaldocml.

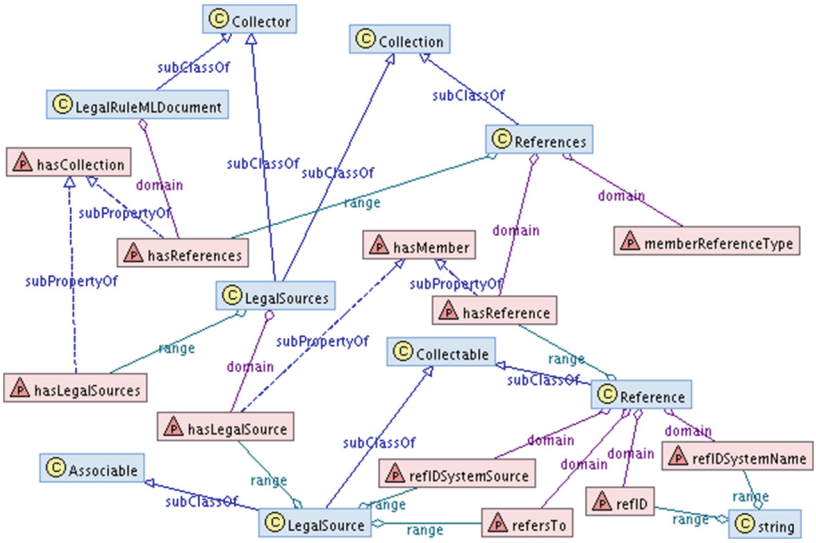


Fig. 5. Metamodel for LegalSource concepts.

5.2 Jurisdiction and Authority

The `Jurisdiction` element is a geographic area or subject-matter over which an Authority applies its legal power. It annotates the legal rules that are applicable to a given region (e.g., the rules applicable only in the United States of America in contrast to other countries in the world).

```
<lrml:Jurisdictions>
  <lrml:Jurisdiction key="us"
    sameAs="http://example.org/jurisdiction#unitedStatesOfAmerica"/>
</lrml:Jurisdictions>
```

We can use Jurisdiction also to specify a limited subject-matter, for instance, legal rules which are applicable only to the executive departments.

```
<lrml:Jurisdictions>
  <lrml:Jurisdiction key="exd"
    sameAs="http://example.org/jurisdiction#executiveDepartments"/>
</lrml:Jurisdictions>
```

Similarly, authority qualifies the rules with respect to the authenticity of the provenance of the formal model. Authority is a person or organization with the power to create, endorse, or enforce Legal Norms.

```
<lrml:Authorities>
  <lrml:Authority key="congress"
    sameAs="unibo:organization.owl#congress">
    <lrml:hasType iri="lrmlv:Legislature"/>
  </lrml:Authority>
</lrm:Authorities>
```

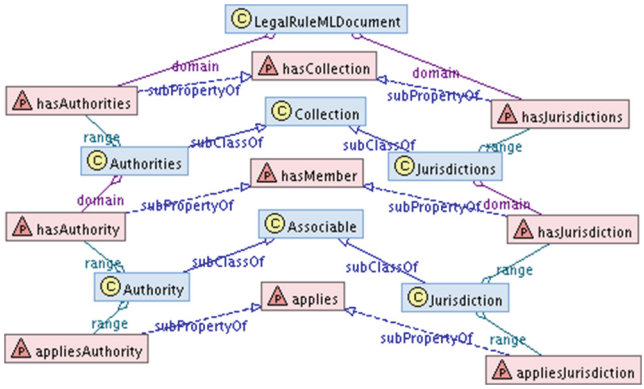


Fig. 6. Metamodel for authority and jurisdiction metadata concepts.

5.3 Agent, Figure, Role

An Agent is an entity that acts or has the capability to act. An Agent could be a physical person, a database, or a bot; for this reason we have the sub-element `<lrml:hasType>` that expresses the category of agent.

```
<lrml:Agents>
  <lrml:Agent key="mp"
    sameAs="http:example.org/agents#MonicaPalmirani">
    <lrml:hasType iri="http://example.org/types#Person"/>
  </lrml:Agent>
  <lrml:Agent key="ta"
    sameAs="http://example.org/agents#TaraAthan"/>
</lrml:Agents>
```

The Agent usually is the author of the rule model and he/she/it can act in a particular function (e.g., as senator). A Figure in LegalRuleML is an instantiation of a function by an Actor, and an Actor could be an Agent or a Figure.

```
<lrml:Figures>
  <lrml:hasMemberType
    iri="http://example.org/figure-types#LegislativeFigure"/>
  <lrml:Figure key="fs">
    <lrml:hasFunction iri="http://example.org/functions#Senator"/>
    <lrml:hasActor keyref="#ta"/>
  </lrml:Figure>
</lrml:Figures>
```

In the end we associate the Actor that fills a Role (using `<lrml:filledBy>`) for a particular rule.

```
<lrml:Roles>
  <lrml:Role key="role1" iri="http://example.org/roles#author">
    <lrml:filledBy keyref="#mp"/>
    <lrml:filledBy keyref="#ta"/>
    <lrml:forExpression keyref="#rule1a"/>
  </lrml:Role>
  <lrml:Role key="role2" iri="http://example.org/roles#author">
    <lrml:filledBy keyref="#mp"/>
    <lrml:forExpression keyref="#atom2a"/>
    <lrml:forExpression keyref="#atom2b"/>
  </lrml:Role>
</lrml:Roles>
```

Using this mechanism we can filter all the rules modelled by a particular Actor when he/she/it acts as a particular figure; for instance, we can filter for all the rules modelled by President Obama when he is acting as chief executive and not as the commander-in-chief of the United States Armed Forces.

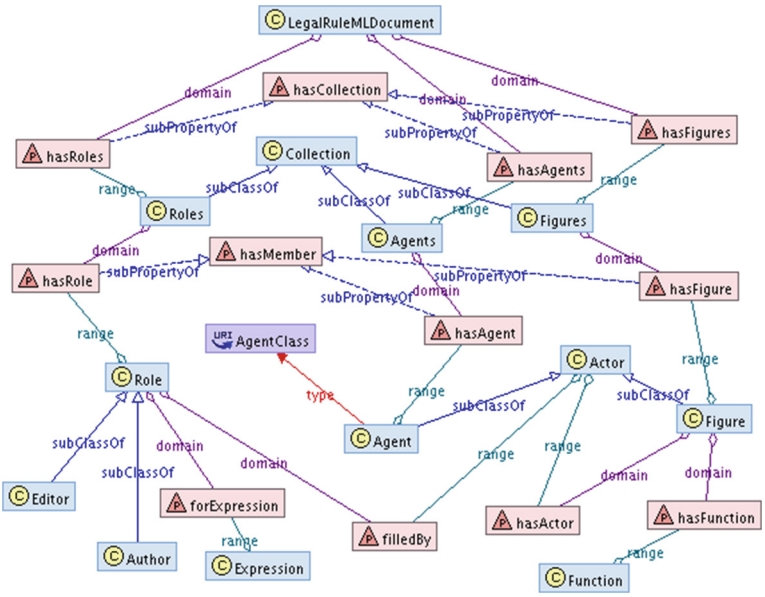


Fig. 7. Partial metamodel for agent, figure and role metadata concepts. AgentClass is imported as URI from dublin core.

5.4 Time and Events

Legal texts are often amended as a society or judicial system evolves. Norms and rules are valid in a particular interval of time and with respect to three main legal axes: when they come into force (entry or enforceability), when they effect

the intended or desired result (efficacy), and when they apply (applicability). In this section, we model the external temporal dimensions of the norms (e.g., when the norm is valid) and not the temporal dimensions of the complex events that are the content of the textual provision (e.g., when a person is to present a tax application). Therefore, we only model the intervals and temporal parameters that define the period of validity of the rules. Moreover, in keeping with the sources, it is important to link the temporal parameters to any part of a rule (e.g., `atom`, `rel`, `ind`, `if`, `then`, etc.) with a very fine granularity. The following fragment shows the definition of the instant time using the `<ruleml:Time>` element wrapped by the `<lrml:Times>` collection element:

```
<lrml:Times>
  <ruleml:Time key="t1">
    <ruleml:Data xsi:type="xs:dateTime">
      1978-01-01T01:01:00.0Z
    </ruleml:Data>
  </ruleml:Time>
  <ruleml:Time key="t2">
    <ruleml:Data xsi:type="xs:dateTime">
      1989-03-01T01:01:00.0Z
    </ruleml:Data>
  </ruleml:Time>
</lrml:Times>
```

The time instants are combined in intervals according with the legal temporal characteristics, e.g. enforceability, efficacy, applicability. In the following case the tblock1 defines the interval $[t_1, t_2]$ of efficacy.

```
<lrml:TemporalCharacteristics key="tblock1">
  <lrml:TemporalCharacteristic key="e1-b">
    <lrml:forStatus iri="lrmlv:Efficacious"/>
    <lrml:hasStatusDevelopment iri="lrmlv:Starts"/>
    <lrml:atTime keyref="#t1"/>
  </lrml:TemporalCharacteristic>
  <lrml:TemporalCharacteristic key="e1-e">
    <lrml:forStatus iri="lrmlv:Efficacious"/>
    <lrml:hasStatusDevelopment iri="lrmlv:Ends"/>
    <lrml:atTime keyref="#t2"/>
  </lrml:TemporalCharacteristic>
</lrml:TemporalCharacteristics>
```

After this definition of the time interval or instant, it is possible to associate them to the legal sources using the `<lrml:Association>` element or the `<lrml:Context>` element (see Sect. 6) for associating the temporal parameters with any part of the rule formalization.

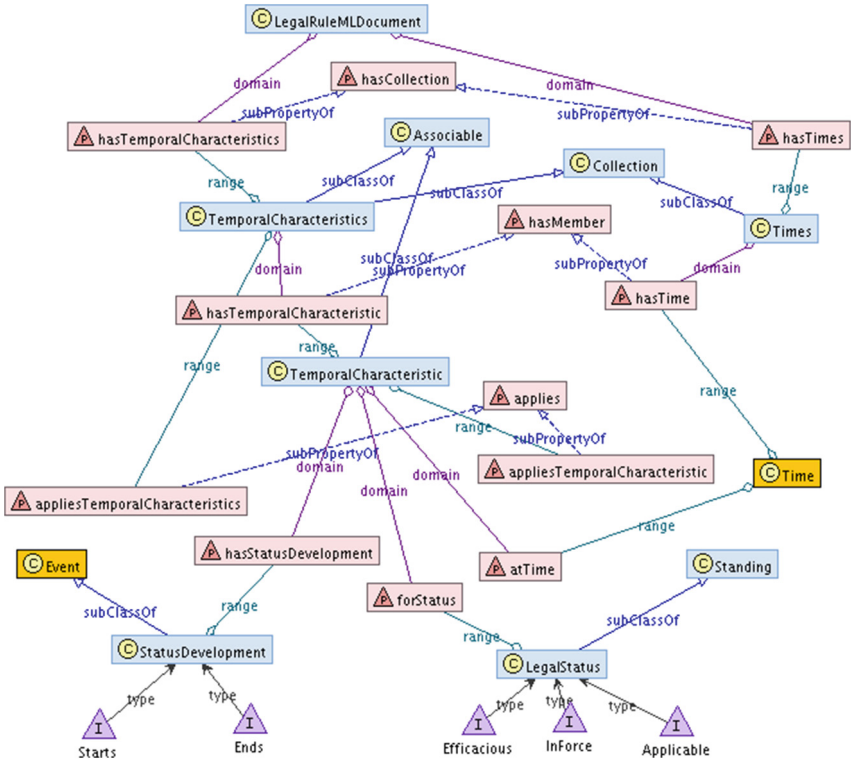


Fig. 8. Partial Metamodel for Temporal Metadata Concepts. Individuals are represented by triangular icons with the letter 'I'. Event and Time classes are imported from RuleML.

6 Association and Context

6.1 Association

To avoid redundancy, we have the element `<Association>` which can be used to group meta information referring to several rules or portions of them. In the following example we have two associations inside of the collection element `<Associations>`. The first `<Association>` applies the temporal parameters of `tblock1` to the prescriptive statements 1 and 2. In the second one authority and jurisdiction properties are applied to prescriptive statements 1 and 3:

```
<lrml:Associations key="sourceBlock1">
  <lrml:Association>
    <lrml:appliesTemporalCharacteristics keyref="#tblock1"/>
    <lrml:toTarget keyref="#ps1"/>
    <lrml:toTarget keyref="#ps2"/>
  </lrml:Association>
  <lrml:Association>
    <lrml:appliesAuthority keyref="ex:#congress"/>
    <lrml:appliesJurisdiction keyref="ex:#us"/>
    <lrml:toTarget keyref="#ps1"/>
    <lrml:toTarget keyref="#ps3"/>
```

```
    </lrml:Association>
  </lrml:Associations>
```

This LegalRuleML language construct permits a large flexibility without replicating the information and so maintains the XML representation neatly, cleanly, compactly, and with fewer redundancies and errors. The parameters that we can associate are:

```
<lrml:appliesModality iri="deovo:obl"/>
```

for expressing modality;

```
<lrml:appliesSource keyref="#sec504-clsc-pnt1"/>
```

for connecting LegalSources or References;

```
<lrml:appliesTemporalCharacteristics keyref="#tblock1"/>
```

for connecting temporal parameters;

```
<lrml:appliesStrength iri="lrmlv:Defeasible"/>
```

for qualifying the strength of a rule according to the defeasibility categorization;

```
<lrml:appliesAuthority keyref="authorities:congress"/>
```

for assigning the authority of the editor of the rule;

```
<lrml:appliesJurisdiction keyref="jurisdictions:us"/>
```

for assigning the jurisdiction to a rule.

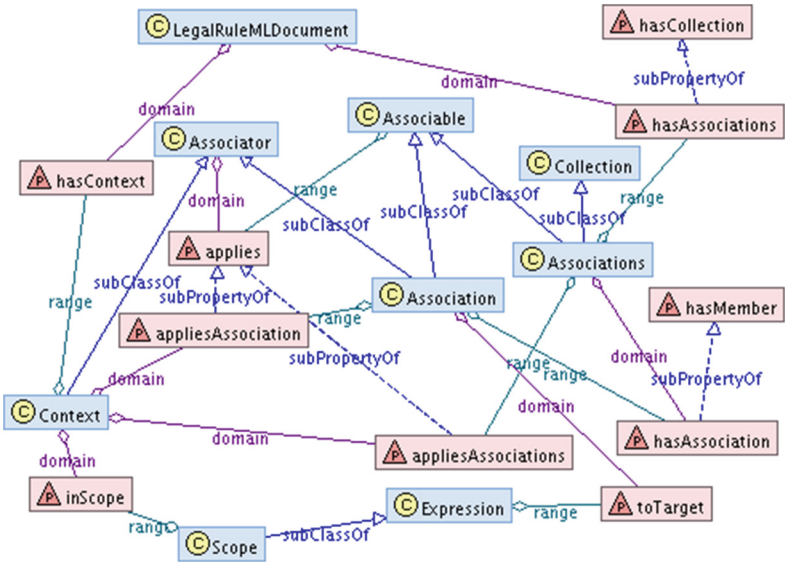


Fig. 9. Partial metamodel for context concepts.

6.2 Context

A rule may be differently interpreted according to a variety of parameters associated with a particular situation. For instance, sometimes an alternative interpretation of a textual source of a rule (and its associated formalisation) is associated with a jurisdiction, e.g., regional, national, or international levels, meaning that in one jurisdiction, the rule is interpreted one way, while in another jurisdiction, it is interpreted in another way. Similarly, temporal parameters (e.g., efficacy, enforceability) can change over time due to the normative modifications, and these changes can also affect the strength of the norms.

To represent such parameters, we introduce the `<lrml:Context>` element, which permits the description of all the characteristics that are linked to a particular rule (e.g., `rule1`) using the operator `<applies*>`, substituting the * with different relationships (see Sect. 6.1). Additionally to the previous relationships we add also the following:

```
<lrml:appliesAssociations keyref="#assoc1"/>
<lrml:appliesAlternatives keyref="#alt2"/>
```

The mechanism combines the relationships and the target rules, and it acts as a bridge between metadata and rules or fragments of them. The following example shows rules `rule1` and `rule4` connected with a LegalSource section 504, point 2, under the authority of Congress, valid in the jurisdiction of the USA, associated with the association `#assoc1` and constrained by the alternatives represented in `#alt2`.

```
<lrml:Context key="ruleInfo4" hasCreationDate="#t1">
  <lrml:appliesSource keyref="#sec504-clsc-pnt2"/>
  <lrml:appliesTemporalCharacteristics keyref="#tblock1"/>
  <lrml:appliesStrength iri="lrmlv:Defeater"/>
  <lrml:appliesAuthority keyref="authorities:congress"/>
  <lrml:appliesJurisdiction keyref="jurisdictions:us"/>
  <lrml:appliesAssociations keyref="#assoc1"/>
  <lrml:appliesAlternatives keyref="#alt2"/>
  <lrml:inScope keyref="#rule1"/>
  <lrml:inScope keyref="#rule4"/>
</lrml:Context>
```

7 Concrete XML-based Syntax Design

The concrete XML-based syntax for LegalRuleML was designed based on the principles in Sect. 3, as well as certain design principles that are specific to XML-based syntaxes.

7.1 XML Elements vs. Attributes

A common design decision for XML-based languages is whether to use an XML element or an attribute to represent a particular abstract syntactic feature. General guidelines are:

- If the information in question could be itself marked up with elements, put it in an element, because attributes cannot contain such complex content;
- If the information is suitable for attribute form (i.e., not complex), but could end up as multiple attributes of the same name on the same element, use child elements instead, avoiding list datatypes for attributes;
- If the information is required to be in a standard XML schema attribute type such as ID, IDREF, ENTITY, KEYREF, use an attribute;
- If the information should not be normalized for white space, use elements (XML processors normalize attributes in ways that can change the raw text of the attribute value.).

Additional general markup conventions developed in RuleML are adopted in LegalRuleML, providing common principles for the merged language hierarchy.

7.2 Node and Edge Elements

There is a distinction between *type* (also called *node*) elements and *role* (also called *edge*) elements, the element name of the former starting with an upper case letter, and the latter with a lower case letter. Node elements correspond to classes of the metamodel while edge elements correspond to relationships between members of these classes. Edge elements correspond, in general, to "object" properties in the metamodel, where the range is a subclass of `rdfs:Resource`. Node elements alternate with edge elements, forming a bipartite pattern, often called a striped syntax (e.g., the striped RDF/XML syntax).

7.3 Specialization of Language Constructs with Attributes and Header Elements

Main XML elements are used for representing general language constructs as recursive trees while XML attributes and nonrecursive *header* elements are used for distinguishing specializations of a given main element. (Attributes are also used for rendering names that are IRIs, as in RDF.) Syntactic and semantic variation can thus be achieved by different attribute values and header elements rather than requiring a different element name. Consider the case of M general language constructs, all of which may be specialized by P attributes, each with N predefined values. With this approach, a vocabulary of size $M + N * (P + 1)$ is able to express $M * N^P$ specialized language constructs. In practice, not all specializing components are appropriate for all general language constructs, so the actual reduction of vocabulary is not as dramatic as the example, but still significant.

7.4 Generic Elements

In addition to predefined values, a number of RuleML and LegalRuleML attributes are allowed values which are IRIs, providing extension points for user-defined syntactic and semantic variation. A *generic* element is a main element

whose semantics is underspecified unless an attached attribute or header element provides a predefined value or an IRI pointer to a user-defined semantics (e.g., `<Obligation>` is a generic deontic operator.) In contrast, non-generic main elements have either a fixed semantics (e.g., `<References>`), or a default semantics specified by a profile reference which may be modified through the use of semantic variant attributes (e.g., `<ruleml:And>`).

7.5 Normalized and Compact Serialization

In many cases, edge elements are redundant because they could be reconstructed based on the type or position of the parent and child node elements. RuleML syntax allows such edges to be optionally skipped, called the *stripe-skipped* serialization. LegalRuleML syntax allows the two extreme cases - either no edges are skipped in the document (the *normalized* serialization) or all skippable edges in the document are omitted (the *compact* seralization). The normalized serialization may be reconstructed from a document in stripe-skipped or compact serialization by applying the *normalizer* XSLT transformation.

7.6 Design Patterns

Inside of LegalRuleML we employ five well-known design patterns:

- *container*, which is a structure of elements having independent existence (e.g., `<Context>` can include several `<Association>` sub-elements);
- *collection*, a subpattern of container that is in the form of a list of elements of the same type (e.g., `<Roles>` that is a sequence of `<Role>` elements);
- *recursive element* (e.g., `<Obligation>` can include other `<Obligation>` elements);
- *marker*, an element that uses attribute `@sameAs` for identifying a source, e.g.,
```
<lrml:LegalSource key="sec504-clsa-pnt1"
    sameAs="UScode:title17-chp5-sec504-clsa-lst1-pnt1"/>
```
- *composite* elements that are made up of different dependent parts, (e.g., a rule `<Rule>` consists of an antecedent `<if>` and conclusion `<then>`).

7.7 IRI References, CURIES, and the Xsd:ID Datatype

Syntactic labels are attached to fragments of LegalRuleML syntax with the `@key` attribute, and are referenced with `@keyref`. On LegalRuleML elements, the datatype of `@key` values is `xsd:ID`, as is used in HTML for same-document references, while the datatype of `@keyref` is either an IRI reference (`xsd:anyURI`) or a CURIE [1].

The names of elements and attributes in the XML syntax of LegalRuleML are inspired by terms from the legal domain, which then facilitates the use by users familiar with this domain. The LegalRuleML meta-model captures the common meaning of such terms as understood in the legal field and provides an IRI for each metamodel term within the LegalRuleML metamodel namespace.

These IRIs may be used whenever it is appropriate to refer to a "resource", in the sense of RDF, including as values of LegalRuleML attributes.

The element names of the LegalRuleML XML-based syntax are qualified names, and all LegalRuleML attributes are unqualified.[8] An XSLT transformation has been defined that converts a LegalRuleML document in the XML-based syntax into RDF that employs the LegalRuleML metamodel vocabulary[9].

In the following section we illustrate the connections among the various concepts and their representation in the language.

8 Examples

We use a fragment of the US Code, Title 17, sec. 504, point (c) on copyright infringement for presenting how LegalRuleML can model complex legal norms in elegant way. Section 504 was modified seven times over several years. However only three versions are relevant in our scenario: (i) the version entered into force at Oct. 19, 1976; (ii) the version entered into force at Oct. 31, 1988; (iii) the version entered into force at Dec. 9, 1999 that is valid till today. The original version is:

> 17 USC Sec. 504
> (c) Statutory Damages.
> (1) Except as provided by clause (2) of this subsection, the copyright owner may elect, at any time before final judgement is rendered, to recover, instead of actual damages and profits, an award of statutory damages for all infringements involved in the action, with respect to any one work, for which any one infringer is liable individually, or for which any two or more infringers are liable jointly and severally, in a sum of not less than $250 or more than $10,000 as the court considers just. For the purposes of this subsection, all the parts of a compilation or derivative work constitute one work.
> (2) In a case where the copyright owner sustains the burden of proving, and the court finds that infringement was committed willfully, the court in its discretion may increase the award of statutory damages to a sum of not more than $50,000. In a case where the infringer sustains the burden of proving, and the court finds, that such infringer was not aware and had no reason to believe that his or her acts constituted an infringement of copyright, the court in its discretion may reduce the award of statutory damages to a sum of not less than $100.

The Copyright Act establishes conditions to protect various types of intellectual property or work, by preventing, in general, the use of such works without a license and by providing exceptions to the general provision.

[8] Certain qualified attributes in external namespaces are imported into LegalRuleML.
[9] https://tools.oasis-open.org/version-control/browse/wsvn/legalruleml/trunk/schemas/xslt/triplifyMerger-ids.xsl.

For the purpose of this tutorial, the conditions can be paraphrased using the following prescriptive rule:

R1: if a piece of work is covered by copyright, then it is forbidden to use it.

and its companion constitutive rule

C1: an infringer is defined as somebody who used a piece of work when it was forbidden to use it.

The provisions in Section 504 can now be paraphrased as follows:

- R2: if the copyright owner claims statutory damages then the penalty for the infringer is to pay statutory damages of between $250 and $10,000.
- R3: if the copyright owner sustains the burden of proof and the infringer infringes copyright willfully then the penalty for the infringer is to pay statutory damages of between $250 and $50,000.
- R4: if the infringer sustains the burden of proof and the infringer infringes NOT willfully then the penalty for the infringer is to pay statutory damages of between $100 and $10,000.
- Defeasability: $R4 > R3 > R2$.

Over time the penalties change as follow:

	Statutory Damages			
Interval of efficacy of the norm	min	max	Willfully	Not Willfully
[1976-10-19, 1995-03-01[	$250	$10,000	$50,000	$100
[1995-03-01, 2001-02-01[	$500	$20,000	$100,000	$200
[2001-02-01, ∞ [	$750	$30,000	$150,000	$200

The prescriptive rule that represents the first case is the following:[10]

```
<lrml:PrescriptiveStatement key="ps2-tblock1">
  <ruleml:Rule key=":rule2-tblock1" closure="universal">
    <ruleml:if>
      <ruleml:And>
        <ruleml:Atom keyref=":rule0-ruleml-Atom1"/>
        <ruleml:Atom key=":rule2-tblock1-ruleml-Atom1">
          <ruleml:Rel iri="glevo:claimStatutoryDamages">
            claims statutory damages
          </ruleml:Rel>
          <ruleml:Var type="lovo:copyrightOwner">X</ruleml:Var>
        </ruleml:Atom>
      </ruleml:And>
    </ruleml:if>
    <ruleml:then>
      <lrml:Reparation keyref="#rep1-tblock1"/>
    </ruleml:then>
  </ruleml:Rule>
</lrml:PrescriptiveStatement>
```

[10] The full LegalRuleML representation of section 504 is available from https://tools.oasis-open.org/version-control/browse/wsvn/legalruleml/trunk/examples/tutorial/USC_17_504_context.lrml.

The `<lrml:Reparation keyref="#rep1-tblock1"/>` is a reference to the following fragment of code that connects `penalty1` related to the time `tblock1` with the prescriptive rule that is violated:

```
<lrml:Reparation key="rep1-tblock1">
  <lrml:appliesPenalty keyref="#penalty1-tblock1"/>
  <lrml:toPrescriptiveStatement keyref="#ps1"/>
</lrml:Reparation>
```

Finally the penalty is modelled as follows to represent the range of the sanction:

```
  <lrml:PenaltyStatement key="penalty1-tblock1">
    <lrml:Obligation key="penalty1-tblock1-obl1">
      <ruleml:slot>
        <lrml:Bearer iri="deovo:oblBearer"/>
        <ruleml:Var>Y</ruleml:Var>
      </ruleml:slot>
      <ruleml:slot>
        <lrml:AuxiliaryParty iri="deovo:auxParty"/>
        <ruleml:Var>X</ruleml:Var>
      </ruleml:slot>
      <ruleml:Atom key=":penalty1-tblock1-obl1-axm1">
        <ruleml:Rel iri="lovo:payStatutoryDamages"/>
        <ruleml:slot>
          <ruleml:Ind iri="lovo:payMin"/>
          <ruleml:Ind>$250</ruleml:Ind>
        </ruleml:slot>
        <ruleml:slot>
          <ruleml:Ind iri="lovo:payMax"/>
          <ruleml:Ind>$10,000</ruleml:Ind>
        </ruleml:slot>
      </ruleml:Atom>
    </lrml:Obligation>
  </lrml:PenaltyStatement>
```

As a further illustration of the LegalRuleML modelling capabilities we propose a real life case (taken from the Italian legal system and jurisprudence, originally discussed in [15]) depending on multiple (alternative) interpretation of a norm, and we show possible formalisations of the case and the interpretations. We are going to use the formal representations to illustrate the LegalRuleML mechanisms to cope with the phenomenon of multiple interpretations. The case is based on a dispute of Art. 1, Comma 2, Law 379/1990. The article recites.

> The benefit referred to in comma 1 shall be paid in an amount equal 80 per cent of five-twelfths of the income earned and reported for tax purposes by the freelancer in the second year preceding the year of application.[11]

The case 18/96, Bologna Tribunal, Imola Section, concerns the interpretation of the conjunction in *the income earned* ***and*** *reported for tax purposes....*

[11] L'indennità di cui al comma 1 viene corrisposta in misura pari all'80 per cento di cinque dodicesimi del reddito percepito e denunciato ai fini fiscali dalla libera professionista nel secondo anno precedente a quello della domanda.

A fundamental and unalienable principle of legal language is its close connection with natural language; in particular, the interpretation of a textual provision should be the ordinary meaning conveyed by the text of the provision taking into account its context in the act in which it appears and the purpose or object underlying the act. For example, in the Italian legal systems this connection is prescribed by Article 12 of the Preleggi, Italian Civil Code, stating.

> In applying a statute, the interpreter should not attribute to it a meaning different from that made evident by the proper meaning of the words and by their connection, as well as by the intention of the law maker.[12]

Accordingly, the literal interpretation of the norm is given by the rule

$$earned(x, y-2) \wedge reported(x, y-2) \Rightarrow \left[\mathsf{OBL}^{auxiliary=freelancer}_{bearer=employer}\right] paybenefit(f(x), y) \quad (1)$$

The arguments of the predicates *earned* and *reported* are the income x earned/reported in the year in the second argument $(y-2)$. Similarly for *paybenefit* where the function f encodes the computation of the value of the benefit based on the value of the income x. However, according to the Italian taxation legislation in force at the time of the dispute the income received in one year is reported for tax purpose the year after the year it has been earned. Thus, for example, the income earned in 1995 is reported in 1996. This principle can be formulated as follows:

$$earned(x, y) \rightarrow reported(x, y+1) \quad (2)$$

$$reported(x, y) \rightarrow earned(x, y-1) \quad (3)$$

Consider now the *Income* constant obtained by applying the Russell's definite description operator (ι) on the conjunction in the left-hand side of (1).

$$Income = \iota x(earned(x, y) \wedge reported(x, y)) \quad (4)$$

The conclusion is that the constant *Income* is not denoting, i.e., the interpretation of *Income* is $\emptyset$, thus there is no income "entity" that is earned and reported in one and the same year. Hence, the left hand side of the rule in (1) never holds, and the rule never fires, against the intentions of the legislator.

Based on the textual provision two possible interpretations are possible: in the first interpretation the temporal expression "in the second year preceding the year of application" refers to the income earned in the second year preceding the application, while in the second interpretation it refers to the income reported for tax purposed in the second year preceding the application. For example, for an application in year 1998, the first interpretation bases the computation on

[12] Nell'applicare la legge non si può ad essa attribuire altro senso che quello fatto palese dal significato proprio delle parole secondo la connessione di esse, e dalla intenzione del legislatore.

the income earned in 1996 (and reported in 1997), while for the second interpretation, the value of the benefit is computed starting from the income reported in 1996 (and earned in 1995). Accordingly, the first interpretation, the interpretation proposed by the freelancer in the case, can be formalised by the rule

$$earned(x, y-2) \Rightarrow \left[\mathsf{OBL}^{auxiliary=freelancer}_{bearer=employer}\right] paybenefit(f(x), y) \tag{5}$$

Similarly the second interpretation, the interpretation proposed by the employer, can be represented by the rule[13]

$$reported(x, y-2) \Rightarrow \left[\mathsf{OBL}^{auxiliary=freelancer}_{bearer=employer}\right] paybenefit(f(x), y) \tag{6}$$

The task of the Judge was to decide which of the two interpretations has to be used for the application of the norm. In the case the Judge argued in favour of the interpretation advanced by the freelancer.

We presented three possible interpretations of the norm, the literal interpretation, the interpretation of the freelancer and the interpretation of the employer. Here we are going to present the LegalRuleML fragments required to encode the formalisations corresponding to the three interpretations. The formalisations of these three statements can be represented as prescriptive rules which are encoded by `<lrml:PrescriptiveStatement>` elements in LegalRuleML, each containing one `<ruleml:Rule>` Template. The following fragment corresponds to the literal interpretation, i.e., (1)

```
<lrml:PrescriptiveStatement key="literal">
  <ruleml:Rule closure="universal" key=":literal-template">
    <ruleml:if>
      <ruleml:And>
        <ruleml:Atom key=":atom-earned">
          <ruleml:Rel iri="lovo:earned"/>
          <ruleml:Var>income</ruleml:Var>
          <ruleml:Expr>
            <ruleml:Fun iri="glevo:subtract"/>
             <ruleml:Var>year</ruleml:Var>
            <ruleml:Data xsi:type="xs:integer">2</ruleml:Data>
          </ruleml:Expr>
        </ruleml:Atom>
        <ruleml:Atom key=":atom-reported">
          <ruleml:Rel iri="lovo:reported"/>
          <ruleml:Var>income</ruleml:Var>
          <ruleml:Expr>
            <ruleml:Fun iri="glevo:subtract"/>
            <ruleml:Var>year</ruleml:Var>
            <ruleml:Data xsi:type="xs:integer">2</ruleml:Data>
          </ruleml:Expr>
        </ruleml:Atom>
      </ruleml:And>
```

[13] Alternatively, we could use $earned(x, y-3) \Rightarrow \left[\mathsf{OBL}^{auxiliary=freelancer}_{bearer=employer}\right] paybenefit(f(x))$, while, from a formal point of view, it is semantically equivalent to (6) it is less close in meaning to the textual provision than its counterpart: the temporal reference in the argument would "third year preceding the year of the application".

```
    </ruleml:if>
    <ruleml:then>
      <lrml:Obligation key="obl-paybenefit">
        <ruleml:slot>
          <lrml:Bearer/>
          <ruleml:Var>Employer</ruleml:Var>
        </ruleml:slot>
        <ruleml:slot>
          <lrml:AuxiliaryParty/>
          <ruleml:Var>Freelancer</ruleml:Var>
        </ruleml:slot>
        <ruleml:Atom>
          <ruleml:Rel iri="lovo:paybenefit"/>
          <ruleml:Expr>
            <ruleml:Fun iri="glevo:80_percent_of_five-twelfths_of"/>
            <ruleml:Var>income</ruleml:Var>
          </ruleml:Expr>
          <ruleml:Var>year</ruleml:Var>
        </ruleml:Atom>
      </lrml:Obligation>
    </ruleml:then>
  </ruleml:Rule>
</lrml:PrescriptiveStatement>
```

Since LegalRuleML is built on top of RuleML we can reuse all RuleML facilities, in particular we can use `<ruleml:Expr>` and `<ruleml:Fun>` to encode the computation of the benefit to be paid to the freelancer.

The next snippet captures the interpretation of the freelancer, i.e., (5).

```
<lrml:PrescriptiveStatement key="freelancer">
  <ruleml:Rule closure="universal" key=":freelancer-template">
    <ruleml:if>
      <ruleml:Atom keyref=":atom-earned"/>
    </ruleml:if>
    <ruleml:then>
      <lrml:Obligation keyref="#obl-paybenefit"/>
    </ruleml:then>
  </ruleml:Rule>
</lrml:PrescriptiveStatement>
```

Notice that inside this statement we can use `keyref`s to refer to the elements already defined in the statement corresponding to the literal interpretation. Similar considerations apply to the statement modelling (6), the employer's interpretation, below.

```
<lrml:PrescriptiveStatement key="employer">
  <ruleml:Rule closure="universal" key=":employer-template">
    <ruleml:if>
      <ruleml:Atom keyref=":atom-reported"/>
    </ruleml:if>
      <ruleml:then>
        <lrml:Obligation keyref="#keyobl-paybenefit"/>
      </ruleml:then>
  </ruleml:Rule>
</lrml:PrescriptiveStatement>
```

The following LegalRuleML Constitutive Statement represents the principle expressed in (2), that earned income will be reported in the following year. Because a Constitutive Statement defines concepts and does not prescribe behaviours, the consequent of its `<ruleml:Rule>` Template does not contain deontic operators.

```
<lrml:ConstitutiveStatement key="tax1">
 <ruleml:Rule closure="universal">
   <ruleml:if>
     <ruleml:Atom>
       <ruleml:Rel iri="lovo:earned"/>
       <ruleml:Var>income</ruleml:Var>
       <ruleml:Var>year</ruleml:Var>
     </ruleml:Atom>
   </ruleml:if>
   <ruleml:then>
     <ruleml:Atom>
       <ruleml:Rel iri="lovo:reported"/>
       <ruleml:Var>income</ruleml:Var>
       <ruleml:Expr key=":year+1">
         <ruleml:Fun iri="glevo:add"/>
         <ruleml:Var>year</ruleml:Var>
         <ruleml:Data xsi:type="xs:integer">1</ruleml:Data>
       </ruleml:Expr>
     </ruleml:Atom>
   </ruleml:then>
 </ruleml:Rule>
</lrml:ConstitutiveStatement>
```

Similarly, the following fragment represents the principle that reported income was earned in the previous year, as expressed in (3).

```
<lrml:ConstitutiveStatement key="tax2">
 <ruleml:Rule closure="universal">
   <ruleml:if>
     <ruleml:Atom>
       <ruleml:Rel iri="lovo:reported"/>
       <ruleml:Var>income</ruleml:Var>
       <ruleml:Var>year</ruleml:Var>
     </ruleml:Atom>
   </ruleml:if>
   <ruleml:then>
     <ruleml:Atom>
       <ruleml:Rel iri="lovo:earned"/>
       <ruleml:Var>income</ruleml:Var>
       <ruleml:Expr key=":year-1">
         <ruleml:Fun iri="glevo:subtract"/>
         <ruleml:Var>year</ruleml:Var>
         <ruleml:Data xsi:type="xs:integer">1</ruleml:Data>
       </ruleml:Expr>
     </ruleml:Atom>
   </ruleml:then>
 </ruleml:Rule>
</lrml:ConstitutiveStatement>
```

After the renderings of the alternative interpretations and the relationships between the predicates *earned* and *reported* given by the three constitutive

rules, we have to specify that they are mutually exclusive formalisation of the same norm. This can be achieved by the following Alternatives element that represents a mutually-exclusive collection of renderings of the Legal Norms from the Legal Source `#ls1`. The `<lrml:LegalSource>` with key `#ls1`, not shown in the text, contains the references to the actual text of the norm.

```
<lrml:Alternatives key="maternity-alts">
  <lrml:Comment> These alternatives are mutually
     incompatible formalizations of the same legal source: keyref="#ls1".
  </lrml:Comment>
  <lrml:hasAlternative keyref="#literal" />
  <lrml:hasAlternative keyref="#freelancer" />
  <lrml:hasAlternative keyref="#employer" />
</lrml:Alternatives>
```

A `<lrml:Context>` element is used to render a collection of Associations, e.g. the Association of a Legal Source with a rendering of it as a LegalRuleML Statement, or to constrain other Contexts with respect to Alternatives. The following Context establishes a constraint that at most one of the Alternatives from the collection `#maternity-alts` may be selected by each Context:

```
<lrml:Context key="maternity-alts-ctxt">
  <lrml:appliesAssociations keyref="#asn-alts"/>
  <lrml:appliesAlternatives keyref="#maternity-alts"/>
</lrml:Context>
```

The Context metadata, e.g. authorship, source, authority, temporal and jurisdictional properties, are specified in an external (to the Context) Association element with identifier `asn-alts`, not shown in the paper, which is referenced using `keyref`. Similarly other Context elements (also not shown in the paper) are given with the metadata about the authors of the various Statements. This permits to establish the provenance of the interpretations.

In the following fragment, a particular Alternative – that proposed by the freelancer – is selected, leading to the generation of the corresponding `<ruleml:Rule>` from the rule Template `:freelancer-template`.

```
<lrml:Context key="adjudication">
  <lrml:appliesAssociation keyref="#asn-adjudication"/>
  <lrml:inScope keyref="#freelancer"/>
</lrml:Context>
```

Unlike the first Context element, this one contains an `<lrml:inScope>` element. Such Contexts render interpretations that select one or more Statements as their scope of interpretation. When a Context is processed for presentation or inference, Legal Rules[14] are generated from the `<ruleml:Rule>` Templates of in-scope Statements, annotated and optionally modified semantically by the Associations of the Context.

[14] In this paper, we focus on Prescriptive and Constitutive Statements, which always lead to generated Legal Rules. However, in the general case, e.g. `<lrml:FactualStatement>`, something other than a Legal Rule may be generated when a Statement is in scope.

In this example the external Association `asn-adjudication` links the metadata for the adjudication of the case with a particular rendering of the norm, the rendering `freelancer`, corresponding to the interpretation proposed by the freelancer and confirmed by the judge[15].

9 Conclusion

The tutorial introduces LegalRuleML, a markup up language with a rich set of features and vocabulary. The language is guided by design principles and illustrated with some examples. LegalRuleML is intended to model legal rules and to facilitate reasoning with them by fulfilling the most important requirements in the legal domain such as the use of deontic operators, defeasible logic, and temporal parameters along with the qualification of the norms (e.g., constitutive, prescriptive, reparation, penalty) and the connection between legal sources and metadata of the rules. In addition to an XML syntax, LegalRuleML provides a methodology for analysing legal texts and for formally representing norms. LegalRuleML permits the representation of alternative interpretations of the same part of legal text, adhering to legal practice. The <lrml:Association> structure helps to compose different properties and to connect such compositions with rules or fragment of rules (e.g., Atom). The metamodel of LegalRuleML is the main pillar of the vocabulary design, helping to guide consistent modelling over time and allowing the language to evolve and be extended. However, sometimes LegalRuleML is too verbose, flexible, or detailed, making it difficult to properly manually manage the markup. The flexibility the XML-schema is especially difficult, for it does not impose some conceptual constraints that are important for the analysis. For these reasons, some tools are now emerging to help legal knowledge engineers, who many not be familiar with XML or RDF principles, to correctly apply LegalRuleML. Other tools can be applied to LegalRuleML representations and reason with them. RAWE is a web editor that supports a legal knowledge engineer to model norms starting from the original legal text [28]. SPINdle is a legal reasoner that implements defeasible reasoning and the temporal reasoning [25]. PROVA is an open-source rule language that can be used by LegalRuleML to manage the temporal parameters and to integrate with Reaction RuleML (https://prova.ws/). There are also tools provided in the LegalRuleML OASIS repository to serialize RDF files in favour of the Semantic Web linked open data model. Considering these tools, the application of LegalRuleML is promising; it is well supported by a robust design, a firm basis in legal theory, a sound XML syntax, and illustrations of how the language is applied.

[15] The full example is available from https://tools.oasis-open.org/version-control/browse/wsvn/legalruleml/trunk/examples/approved/maternity_alternatives_compact.lrml.

References

1. Adida, B., Birbeck, M., McCarron, S., Herman, I.: RDFa core 1.1 - third edition. http://www.w3.org/TR/rdfa-core/#s_curies
2. Alchourrón, C.E., Bulygin, E.: Permission and permissive norms. In: Krawietz, W., et al. (eds.) Theorie der Normen, pp. 349–371. Duncker & Humblot, Berlin (1984)
3. Antoniou, G.: Defeasible logic with dynamic priorities. Int. J. Intell. Syst. **19**(5), 463–472 (2004)
4. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation results for defeasible logic. ACM Trans. Comput. Logic **2**(2), 255–287 (2001)
5. Athan, T., Boley, H., Governatori, G., Palmirani, M., Paschke, A., Wyner, A.: OASIS LegalRuleML. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law, pp. 3–12, New York (2013)
6. Athan, T., Governatori, G., Palmirani, M., Paschke, A., Wyner, A.: Legal interpretations in LegalRuleML. In: Villata, S., Peroni, S., Palmirani, M. (eds.) Proceedings of the Semantic Web for the Law and Second Jurix Doctoral Consortium Workshops (SW4LAW+JURIX-DC 2014). CEUR Workshop Proceedings, vol. 1296, CEUR-WS.org (2014)
7. Bench-Capon, T., Coenen, F.P.: Isomorphism and legal knowledge based systems. Artif. Intell. Law **1**(1), 65–86 (1992)
8. Brickley, D., Guha, R.V.: RDF schema 1.1. http://www.w3.org/TR/rdf-schema/
9. Dattolo, A., Di Iorio, A., Duca, S., Feliziani, A.A., Vitali, F.: Structural patterns for descriptive documents. In: Baresi, L., Fraternali, P., Houben, G.-J. (eds.) ICWE 2007. LNCS, vol. 4607, pp. 421–426. Springer, Heidelberg (2007)
10. de Saussure, F.: Cours de Linguistique Générale. Payot, Lausanne (1916)
11. Dworkin, R.: The model of rules I. In Taking Rights Seriously. Harvard University Presss, Cambridge, MA (1977)
12. Gordon, T., Prakken, H., Walton, D.: The Carneades model of argument and burden of proof. Artif. Intell. **171**, 875–896 (2007)
13. Gordon, T.F.: The Pleadings Game-An Artificial Intelligence Model of Procedural Justice. Springer, New York (1995)
14. Gordon, T.F., Governatori, G., Rotolo, A.: Rules and norms: Requirements for rule interchange languages in the legal domain. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 282–296. Springer, Heidelberg (2009)
15. Governatori, G.: Un modello formale per il ragionamento giuridico. Ph.D. thesis, CIRFID, Università di Bologna (1997)
16. Governatori, G.: Representing business contracts in RuleML. Int. J. Coop. Inf. Syst. **14**(2–3), 181–216 (2005)
17. Governatori, G.: On the relationship between Carneades and defeasible logic. In: van Engers, T. (ed.) Proceedings of the 13th International Conference on Artificial Intelligence and Law (ICAIL 2011), pp. 31–40. ACM Press (2011)
18. Governatori, G.: Business process compliance: An abstract normative framework. IT Inf. Technol. **55**(6), 231–238 (2013)
19. Governatori, G., Olivieri, F., Rotolo, A., Scannapieco, S.: Computing strong and weak permissions in defeasible logic. J. Philos. Logic **42**(6), 799–829 (2013)
20. Governatori, G., Rotolo, A.: Logic of violations: A Gentzen system for reasoning with contrary-to-duty obligations. Australas. J. Logic **4**, 193–215 (2006)
21. Governatori, G., Rotolo, A.: Changing legal systems: legal abrogations and annulments in defeasible logic. Logic J. IGPL **18**(1), 157–194 (2010)

22. Governatori, G., Rotolo, A., Sartor, G.: Temporalised normative positions in defeasible logic. In: Proceedings of the 10th International Conference on Artificial Intelligence and Law (ICAIL 2005), pp. 25–34. ACM (2005)
23. Hart, H.: The Concept of Law, 2nd edn. Clarendon Press, Oxford (1994)
24. Herrestad, H., Krogh, C.: Obligations directed from bearers to counterparts. In: Proceedings of the Fifth International Conference on Artificial Intelligence and Law (ICAIL 1995), pp. 210–218 (1995)
25. Lam, H.-P., Governatori, G.: The making of SPINdle. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 315–322. Springer, Heidelberg (2009)
26. Lappin, S. (ed.): The Handbook of Contemporary Semantic Theory. Blackwell Publishers, Cambridge (1997)
27. Nute, D.: Handbook of Logic in Artificial Intelligence and Logic Programming, volume 3, chapter Defeasible Logic, pp. 353–395. Oxford University Press, Oxford, 1994
28. Palmirani, M., Cervone, L., Bujor, O., Chiappetta, M.: RAWE: an editor for rule markup of legal texts. In: Fodor, P., Roman, D., Anicic, D., Wyner, D., Palmirani, M., Sottara, D., Lévy, F. (eds.) Joint Proceedings of the 7th International Rule Challenge, the Special Track on Human Language Technology and the 3rd RuleML Doctoral Consortium. CEUR Workshop Proceedings, Seattle, USA, 11–13 July 2013, vol. 1004, CEUR-WS.org (2013)
29. Palmirani, M., Governatori, G., Contissa, G.: Temporal dimensions in rules modelling. In: Winkels, R. (ed.) JURIX. Frontiers in Artificial Intelligence and Applications, vol. 223, pp. 159–162. IOS Press, Amsterdam (2010)
30. Palmirani, M., Governatori, G., Rotolo, A., Tabet, S., Boley, H., Paschke, A.: LegalRuleML: XML-based rules and norms. In: Palmirani, M. (ed.) RuleML - America 2011. LNCS, vol. 7018, pp. 298–312. Springer, Heidelberg (2011)
31. Prakken, H., Sartor, G.: A dialectical model of assessing conflicting argument in legal reasoning. Artif. Intell. Law **4**(3–4), 331–368 (1996)
32. Prakken, H., Sartor, G.: Argument-based extended logic programming with defeasible priorities. J. Appl. Non Class. Logics **7**(1), 25–75 (1997)
33. Raz, J.: Between authority and interpretation: on the theory of law and practical reason. Oxford University Press, Oxford (2009)
34. Sartor, G.: Legal reasoning: A cognitive approach to the law. In: Pattaro, E., Rottleuthner, H., Shiner, R.A., Peczenik, A., Sartor, G. (eds.) A Treatise of Legal Philosophy and General Jurisprudence, vol. 5. Springer, Berlin (2005)
35. Scalia, A., Garner, B.A.: Reading Law: The Interpretation of Legal Texts. West, Minneapolis (2012)
36. Searle, J.R.: The Construction of Social Reality. The Free Press, New York (1996)
37. Nicos Stavropoulos. Legal interpretivism. In Edward N. Zalta, editor, The Stanford Encyclopedia of Philosophy. Summer 2014 edition (2014)
38. Georg Henrik von Wright: Norm and action: A logical inquiry. Routledge and Kegan Paul, London (1963)