Using UML for Modeling Procedural Legal Rules: Approach and a Study of Luxembourg's Tax Law

Ghanem Soltana, Elizabeta Fourneret, Morayo Adedjouma,
Mehrdad Sabetzadeh, and Lionel Briand

SnT Centre for Security, Reliability and Trust, University of Luxembourg
Email: {firstname.lastname}@uni.lu

Abstract. Many laws, e.g., those concerning taxes and social benefits, need to be operationalized and implemented into public administration procedures and eGovernment applications. Where such operationalization is warranted, the legal frameworks that interpret the underlying laws are typically *prescriptive*, providing *procedural rules* for ensuring legal compliance. We propose a UML-based approach for modeling procedural legal rules. With help from legal experts, we investigate actual legal texts, identifying both the information needs and sources of complexity in the formalization of procedural legal rules. Building on this study, we develop a UML profile that enables more precise modeling of such legal rules. To be able to use logic-based tools for compliance analysis, we automatically transform models of procedural legal rules into the Object Constraint Language (OCL). We report on an application of our approach to Luxembourg's Income Tax Law providing initial evidence for the feasibility and usefulness of our approach.

1 Introduction

Legal compliance is a major concern for governments. In domains such as taxation and social benefits, laws need to be operationalized so that they can be implemented into administrative procedures and software systems. Such operationalization is typically performed by putting in place a legal framework, comprised of legislation, regulations, and circulars, aimed at providing a detailed interpretation of the underlying laws. These frameworks are often *prescriptive*: they provide step-by-step guidance in the form of *procedural rules* as to what needs to be done for compliance. Procedural legal rules are closely linked to the behavior of eGovernment applications. To illustrate, consider Article 2 from Luxembourg's Income Tax Law [14], describing how taxpayers are classified as resident and non-resident:

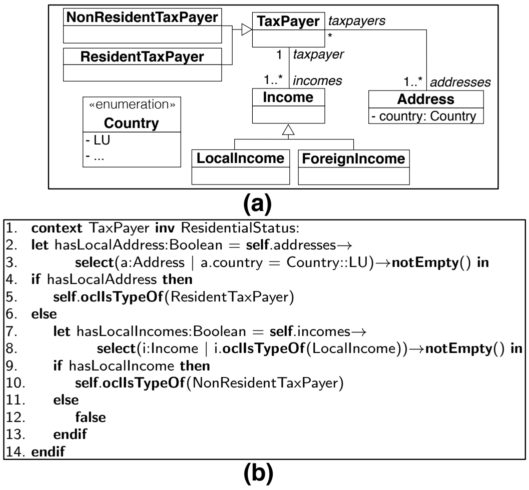
> ***Article 2.***[1] *Individuals are considered resident taxpayers if they have their address in the Grand Duchy. Individuals are considered non-resident taxpayers if they do not reside in the Grand Duchy but have a local income within the definition of Article 156.*

To be able to analyze whether a software system complies with the taxpayer classification described in the law, one could develop a UML model like the one in Fig. 1: The domain model in Fig. 1(a) captures the main concepts and associations in Article 2 of the Income Tax Law; and the OCL expression in Fig. 1(b),

[1] The article has been translated from the original French text and simplified.

written in the context of `TaxPayer`, provides a procedural rule for distinguishing between resident and non-resident taxpayers (L. 2-5 and 7-13, respectively).

To be a resident taxpayer, one must have a Luxembourgish address (L. 2-3). If such an address exists (L. 4), the taxpayer is deemed resident (L. 5). To be a non-resident taxpayer, one must have a local income (L. 7-8) but no local address. If these requirements are met (L. 9), the taxpayer is deemed non-resident (L. 10). A model like that in Fig. 1 makes the underlying legal article amenable to automated analysis. In particular, one can use such a model to check whether the outcome produced by a software system is consistent with the law. For example, using existing OCL evaluators such as Eclipse OCL [11], one can verify if a system correctly classifies taxpayers (instances of the model in Fig. 1(a)) into resident and non-resident.



(a)

```
1.  context TaxPayer inv ResidentialStatus:
2.  let hasLocalAddress:Boolean = self.addresses→
3.          select(a:Address | a.country = Country::LU)→notEmpty() in
4.  if hasLocalAddress then
5.      self.oclIsTypeOf(ResidentTaxPayer)
6.  else
7.      let hasLocalIncomes:Boolean = self.incomes→
8.              select(i:Income | i.oclIsTypeOf(LocalIncome))→notEmpty() in
9.      if hasLocalIncome then
10.         self.oclIsTypeOf(NonResidentTaxPayer)
11.     else
12.         false
13.     endif
14. endif
```

(b)

Fig. 1. (a) domain model for a legal article, (b) procedural rule for the article (expressed as OCL)

Before a model such as the one in Fig. 1 can be used for automated analysis, it needs to be reviewed and validated by legal experts. To aid with validation, it is helpful to express procedural rules such as that in Fig. 1(b) in a visual manner.

This paper develops a visual and at the same time semantically-precise way to model procedural legal rules. Our approach follows the Domain-Specific Modeling (DSM) paradigm; but rather than building a new language, we use UML's built-in customization mechanism, namely *profiles* [22], to adapt UML for use in our context. Using UML is motivated by its widespread use, commercial tool support, and the availability of standard extension mechanisms in the language.

Our work addresses a real need observed during our collaboration with our public service partner, CTIE (Centre des Technologies de l'Information de l'Etat). CTIE is Luxembourg's national center for information technologies and responsible for developing eGovernment services for the state. CTIE already applies Model Driven Engineering (MDE), including UML and its extensions, for system development and is interested in enhancing its development methods with means for modeling legal rules. An important consideration for CTIE is for the models to be palatable to governmental stakeholders without IT background, but who have familiarity with simple conceptual models and business process models from earlier exposure and training.

The approach we propose in this paper is not meant as a general solution for modeling all types of legal rules. In particular, we focus on prescriptive legal frameworks where legal rules are procedural. This situation is typical of highlyregulated domains such as taxation and social benefits. In general, however, many legal frameworks, e.g. privacy laws, are declarative, with rules defined using deontic notions, i.e., permissions, obligations, and prohibitions [24]. Our current solution does not extend to declarative legal rules. In the rest of this paper, we therefore take legal rule to mean "procedural" legal rule.

The starting point for our work is a field study, where we interacted with legal experts and analyzed several legal statutes, to identify both the information needs and the sources of complexity in the formalization of (procedural) legal rules (Section 2). Drawing on our field study, we define a UML-based methodology for modeling legal rules (Section 3). The core component of the methodology is a customization of UML Activity Diagrams, defined through a UML profile (Section 4). To use for analysis purposes the models resulting from our approach, we provide an algorithm for automatic transformation of the models into OCL (Section 5). We report on a case study, providing initial evidence for the feasibility and usefulness of our approach (Section 6). Finally, we compare our approach with related work and suggest avenues for future work (Sections 7–8). The paper is accompanied by a technical report [26] where we provides additional details about our UML profile and automated transformation to OCL.

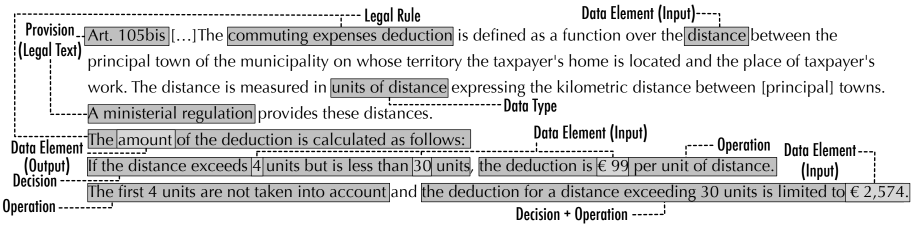
2 Field Study of Legal Rules

Our field study applies a Grounded Theory (GT) process [8], whereby observations and analysis of collected data are used for defining the problems to be addressed. In our context, we apply GT to define (1) what needs to be expressed in models of legal rules, i.e., the information requirements that such models should meet; and (2) factors that lead to complexity in models of legal rules and thus need special consideration in an approach targeted at building such models.

We began our field study with a series of meetings with legal experts, totaling ≈ 15 hours. The purpose of these meetings was (a) for the researchers to develop familiarity with legal concepts; (b) to define a suitable scope for the laws to consider; and (c) to identify representative legal rules for further investigation. Taxation was selected as the scope for the study, partly because of the priorities of the legal experts in our study, and partly because of the tax law's large societal impact. Our field study resulted in several observations, outlined below.

Information Requirements. To identify the information needs in the specification of legal rules, we analyzed selected legal texts concerned with personal income taxes. While personal income taxes are only one facet of the tax law, the experts deemed the scope to be largely representative within the tax domain and the closely related domain of social benefits delivery. With help from legal experts, we identified, read, and interpreted the legal provisions relevant to personal income taxes. Our analysis covered a summary of direct taxes levied by the Government of Luxembourg, 16 articles from Luxembourg's Income Tax Law (for brevity, referred to as *LITL* in the remainder of this paper), three regulations, one tax scale, and several official web pages and circular letters.

While reading the above material, we applied a standard technique from qualitative data analysis [8, 21, 9] for analyzing text, and classifying, describing, and



Art. 105bis [...]The commuting expenses deduction is defined as a function over the distance between the principal town of the municipality on whose territory the taxpayer's home is located and the place of taxpayer's work. The distance is measured in units of distance expressing the kilometric distance between [principal] towns. A ministerial regulation provides these distances.

The amount of the deduction is calculated as follows:

If the distance exceeds 4 units but is less than 30 units, the deduction is € 99 per unit of distance.

The first 4 units are not taken into account and the deduction for a distance exceeding 30 units is limited to € 2,574.

Fig. 2. Excerpt of Article 105bis from *LITL* (translated from French)

connecting the information presented in it. We annotated each important concept with a label denoting the nature of the concept, i.e., a meta-concept. Each time that a new meta-concept was encountered, we defined it in a glossary. As we proceeded through the text, we either created new labels or reused previous ones based on the definitions we had. We illustrate our analysis over an excerpt, shown in Fig. 2, of Art. 105bis of *LITL*. The excerpt, a simplified translation of the original French text, covers many of the information requirements identified by our study. The meta-concepts gleaned from the excerpt are shaded and labeled.

The `(Legal) Rule` the excerpt is concerned with is calculating the deduction a taxpayer is eligible for in relation to their commuting expenses. A rule may depend on several `Provisions`. It is important to maintain traceability from rules to the provisions they depend on. This is necessary both for reasoning about compliance and also for managing change in a predictable way. For the commuting expenses deduction, these provisions are: Art. 105bis of *LITL*, and an abstract reference to a ministerial regulation. *LITL* does not cite any regulation explicitly, as regulations may vary from year to year. The regulation that was in effect for commuting distances at the time our study was conducted is the ministerial regulation of February 6, 2012 ("règlement ministériel du 6 février 2012").

Each rule is made up of a set of `Decisions` and `Operations`, describing the (procedural) flow of the rule. An example decision from the excerpt is: *"If the distance exceeds 4 units but is less than 30 units"*; an example operation is setting "*the deduction [amount to be] €99 per unit of distance*".

There are several `Data Elements` in the excerpt, denoting inputs to, outputs from, or intermediate values computed within the rule. For example, *distance* is an input to and *amount* is the output from the rule. The constants in the text, e.g., €2,574, are marked as input. This choice is motivated by the fact that constants may change over time and thus need to be treated explicitly.

Data elements are typed. The types are sometimes specified in the text, e.g., the excerpt states that *distance* is measured in certain units; but most often, the types are implicit, e.g., for monetary values and dates. One of the goals of our analysis was to identify and restrict the data types associated with the inputs and outputs of legal rules. Doing so is important for improving consistency. For example, all mathematical operations, e.g., summation and multiplication, over monetary values have to be consistent in how they round decimal values with precision points. A uniform treatment requires a specific data type to be defined for monetary values and used consistently in all legal rules.

For data elements that represent inputs, it is important to maintain traceability to the sources where the inputs come from. Some inputs are obtained directly from legal texts, e.g., the constants in the excerpt of Fig. 2. Alternatively, an input may be provided based on expert judgment by a legal agent. For example, to decide whether a company is eligible for certain deductions, a tax officer may need to determine whether the accounting performed by the company is adequate. Finally, an input may be derived from a physical or electronic data record, e.g., the *distance* input mentioned in the excerpt.

We distinguish different sources for inputs. The distinctions are important for better elaboration and validation of legal rules. For example, elaborating the inputs derived from electronic records often requires consultation with both legal experts and IT staff; whereas, inputs based on expert judgment or legal texts typically only concern legal experts.

Based on our analysis above, we have developed an abstract information model, shown in Fig. 3, for legal rules. The model is organized into three packages, in line with the three main observations from the analysis, namely: (1) capturing legal rules as decisions and operations, (2) maintaining traceability, (3) restricting data types to what is essential.

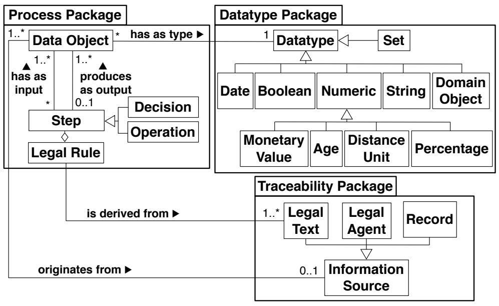


Fig. 3. Information model for legal rules

The `Process` package in Fig. 3 defines the concepts related to the flow of a rule. Each `Legal Rule` is made up of a set of `Steps`, which can be either `Decisions` or `Operations`. Each step has `Data Objects` as input and output. The `Traceability` package groups the information sources to which traceability needs to be maintained from the elements in the `Process` package. Rules need to be traceable to `Legal Texts`. Inputs need to be linked to the `Legal Text`, `Legal Agent`, or `Record` where they originate from. The `Datatype` package contains a partial list of data types identified in our study. There is a special data type, named `Domain Object`, to enable handling instances of domain concepts, e.g., `TaxPayer` (see Fig. 1(a)). In addition, the data types include a composite type, `Set`, to enable handling sets of objects. Note that the data types in Fig. 3 are specific to the tax law and may require tailoring if the approach is applied to other laws and regulations.

Complexity factors. We considered nine legal rules from *LITL* in our analysis of complexity factors. Six of these concern requirements on taxpayers' records (e.g, the taxpayer classification in Fig. 1). The rest concern the calculation of income tax credits. We captured these rules in OCL (total of 108 OCL lines, excluding comments and blanks). Our investigation of the resulting OCL constraints alongside our interactions with legal experts led to the following observations:

– *Navigation*: Navigation expressions in OCL tend to be lengthy for legal rules. For example, Art. 127 of *LITL* sets a cap on the costs a taxpayer can claim for the care of dependents. Calculating this cap requires identifying the dependents who live in the same household as the taxpayer but are not taxpayers themselves, and for whom the taxpayer receives some allowance. The corresponding OCL navigation expression (the OCL context being `TaxPayer`) is as follows:

```
self.taxPayerDependents→select(dependent:Person| not dependent.oclIsTypeOf(TaxPayer) and
        dependent.addresses→intersection(self.addresses)→notEmpty() and
        dependent.allowances.amount→sum()>0)
```

The complexity of navigation expressions is caused in part by the expressive (and thus long) labels of domain model elements in legal contexts, and in part by the richness of legal rules and the need for multiple navigation levels.

– *Branching*: Legal rules often have numerous decision branches, capturing the different cases where they apply and the corresponding actions to take. To illustrate, we recall the example of Fig. 1. Even for the simple task of classifying taxpayers into resident and non-resident, we need two if-then-else statements (or similarly complex propositional logic equivalents of if-then-else). This number rises to six or seven for more complex rules. Feedback from legal experts indicate that branching statements negatively impact comprehension.

– *Iteration*: OCL iterator operations (e.g., `select`, `exists`, `forAll`, `iterate`) are often inevitable in legal rules. For instance, in the navigation expression given earlier for identifying eligible dependents, one has to iterate over the dependents to determine which ones satisfy the desired criteria. Our interaction with legal experts suggests that iterations, specially nested ones, reduce comprehensibility.

Our approach, described next, take steps to address the observed information requirements and complexity factors.

3 Modeling Methodology

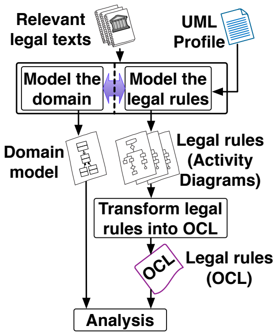


Fig. 4. Methodology

An overview of our modeling methodology is shown in Fig. 4. The legal texts and the specific provisions within them that are relevant to the legal rules of interest are provided as input by legal experts. The modeling step in the methodology includes two parallel but interrelated tasks: (1) modeling the domain and (2) modeling the legal rules. Both tasks require close interaction with legal experts to ensure a sound understanding of the underlying legal notions. The first task results in a domain model, providing a precise representation of the concepts and relationships in the input legal texts. As is common in object-oriented analysis, we use UML class diagrams for representing domain models [18]. A domain model excerpt for Article 2 of *LITL* was shown in Fig. 1(a). We follow standard practices for domain modeling and thus do not elaborate this task further. For guidelines, see [18].

The second modeling task, i.e., modeling of the legal rules, is performed using a customization of UML Activity Diagrams (ADs). ADs have long been used for modeling procedural aspects of systems and organizations [17]. The procedural nature of legal rules makes ADs a good match for our needs. Our customization of AD's is based on UML profiles [22].

The domain model in our methodology is an instrument for elaborating the information that legal rules use as input. It is thus best to conduct tasks (1) and

Table 1. UML profile stereotypes

Stereotype	Description	UML Metaclass(es)	Concept & Package	
«rule»	Defines an activity as a legal rule	`Activity`	`Legal Rule`	*Process*
«iterative»	Defines an iterative region	`ExpansionRegion`		
«context»	Defines the OCL context in which a legal rule is being specified	`Activity`	*Auxiliary*	
«decision»	Defines a decision step	`DecisionNode`	`Operation`	
«calculate»	Defines an operation that calculates a value	`OpaqueAction`		
«assert»	Defines an operation that checks an assertion	`OpaqueAction`	*Auxiliary*	

«in»	Defines an input to a legal rule	`ActivityParameterNode,`
		`InputPin`
			`Data Object`	

«out»	Defines an output from a region	`OutputPin`		
«intermediate»	Defines an intermediate value resulting from a calculation	`CentralBufferNode`	*Auxiliary*	
«formula»	Defines the formula for a calculation	`Constraint`	*Auxiliary*	
«statement»	Defines the logical expression for an assertion	`Constraint`	*Auxiliary*	

«fromlaw»	Declares a (constant) input as originating from a legal text	`ActivityParameterNode,`
		`InputPin`
			`Legal Text`	*Traceability*

«fromagent»	Declares an input as being provided by a legal expert	`ActivityParameterNode,`
		`InputPin`
			`Legal Agent`	

«fromrecord»	Declares an input as being retrieved from a record (e.g., a database)	`ActivityParameterNode,`
		`InputPin`
			`Record`	

«query»	Defines the query for obtaining an input from its respective source	`Comment`	*Auxiliary*	

(2) *in tandem* and not sequentially. Doing these tasks in parallel ensures that the domain model is aligned with the legal rules in terms of data needs, and further narrows the scope of domain modeling to what is necessary for supporting the legal rules of interest. Once the legal rules have been modeled using our tailored AD notation, the models are automatically translated into OCL. The resulting OCL expressions along with the domain model can then be used for automated analysis using OCL evaluators [11] and OCL solvers [7, 1].

Our main technical goal in this paper is to present the profile we have developed to customize ADs for expressing legal rules, and to describe how ADs built using our profile are transformed into OCL. The profile and the OCL transformation are respectively tackled in Sections 4 and 5.

4 UML Profile for Legal Rules

Our profile's stereotypes are shown in the first column of Table 1, followed by a description in the second column. The third column shows the UML metaclass(es) that each stereotype extends. We distinguish two kinds of stereotypes: (1) those that directly represent concepts from the information model of Fig. 3, and (2) those that are auxiliary, providing additional information about model elements. The fourth column in Table 1 shows the mapping between the stereotypes and the concepts and packages of our information model. Auxiliary stereotypes are marked as *Auxiliary* in the column. The *Datatype* package of the information model is not represented through stereotypes. Instead, typing information is attached directly to the input and output nodes of ADs. The profile diagram, the relevant fragment of the UML metamodel, and our datatype library are provided in the supplementary material [26].

We illustrate our profile over the Commuting Expenses Deduction rule from the excerpt of Article 105bis given in Fig. 2. The French term for this deduction

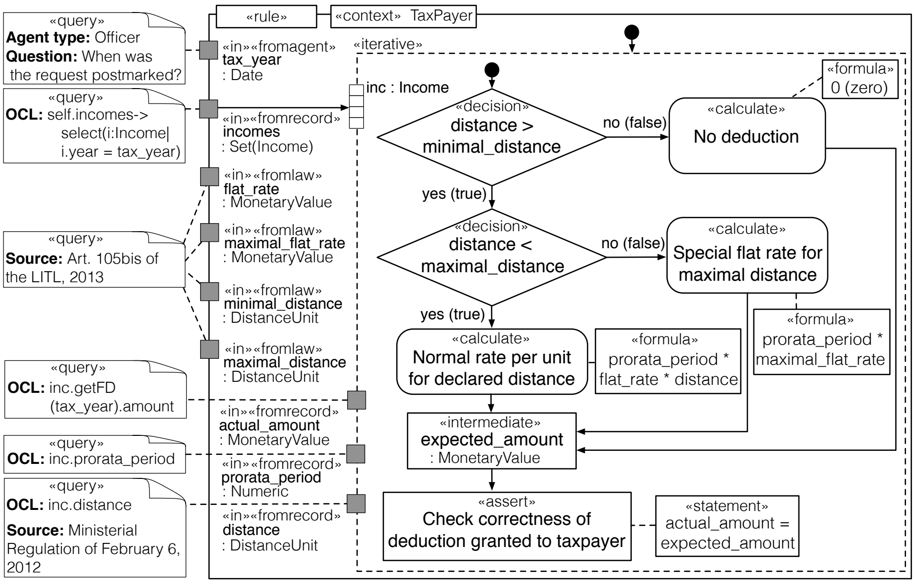

«query»
Agent type: Officer
Question: When was the request postmarked?
«query»
OCL: self.incomes-> select(i:Income| i.year = tax_year)
«query»
Source: Art. 105bis of the LITL, 2013
«query»
OCL: inc.getFD (tax_year).amount
«query»
OCL: inc.prorata_period
«query»
OCL: inc.distance
Source: Ministerial Regulation of February 6, 2012
«rule»
«context» TaxPayer
«in»«fromagent» tax_year : Date
«in»«fromrecord» incomes : Set(Income)
«in»«fromlaw» flat_rate : MonetaryValue
«in»«fromlaw» maximal_flat_rate : MonetaryValue
«in»«fromlaw» minimal_distance : DistanceUnit
«in»«fromlaw» maximal_distance : DistanceUnit
«in»«fromrecord» actual_amount : MonetaryValue
«in»«fromrecord» prorata_period : Numeric
«in»«fromrecord» distance : DistanceUnit
«iterative»
inc : Income
«decision» distance > minimal_distance
no (false)
«calculate» No deduction
«formula» 0 (zero)
yes (true)
«decision» distance < maximal_distance
no (false)
«calculate» Special flat rate for maximal distance
«formula» prorata_period * maximal_flat_rate
yes (true)
«calculate» Normal rate per unit for declared distance
«formula» prorata_period * flat_rate * distance
«intermediate» expected_amount : MonetaryValue
«assert» Check correctness of deduction granted to taxpayer
«statement» actual_amount = expected_amount

Fig. 5. Activity Diagram for Commuting Expenses Deduction (FD)

is “Frais de Déplacement”. We refer to this deduction as *FD*. In Fig. 5, we show how the FD rule is modeled using an AD. The «rule» stereotype applied to the AD in this figure indicates that the AD is a legal rule. The AD is further annotated with a «context» stereotype denoting the OCL context in which the AD is being specified. The context is always an instance of a class from the underlying domain model. For the AD in Fig. 5, the context is an instance of the `TaxPayer` class from the domain model.

The core of the AD in Fig. 5 is a calculation procedure. The procedure yields a value of 0 (zero) when a taxpayer is deemed not eligible for FD, i.e., when `distance` $>$ `minimal_distance` is false. For an eligible taxpayer, the procedure yields the result of multiplying three quantities: (1) a flat rate (constant) from the law, denoted `flat_rate`, (2) the distance between a taxpayer’s work and home addresses, denoted `distance`, and (3) a prorated ratio representing the full-time equivalent period during which the taxpayer has been employed over the course of the tax year, denoted `prorata_period`. The formula applies up to a maximum home-to-work distance threshold, denoted `maximal_distance` and specified in the law. Beyond this threshold, a nominal rate, denoted `maximal_flat_rate`, is applied irrespective of distance but prorated as discussed above.

As illustrated in Fig. 5, the decisions and calculations are marked respectively with the «decision» and «calculate» stereotypes. Each calculation has a «formula» constraint attached, providing the formula for the calculation. The result of a calculation is always stored in a (typed) intermediate variable marked by the «intermediate» stereotype, e.g., `expected_amount` in Fig. 5.

Each legal rule concludes with an assertion: an operation marked by the «assert» stereotype and providing an implicit Boolean output for the rule. Specifically, an assertion is used to ascertain that the outcome produced by a system or a human agent matches the outcome envisaged by the legal rule. Associated with an assertion is a constraint with the «statement» stereotype, defining the Boolean claim that needs to be checked. For example, the assertion in Fig. 5 checks whether the value of FD on a taxpayer's file, denoted `actual_amount`, matches the value computed by the rule, denoted `expected_amount`.

Inputs to decisions and operations are represented by small rectangles with a gray shade and an «in» stereotype. The origin of each input is captured through one of the following stereotypes: «fromlaw», «fromrecord» or «fromagent». Each input has a query attached to it, represented as a comment with a «query» stereotype. A query provides details on how an input is obtained from its source. The «fromlaw» stereotype is used for inputs that are constants and specified in a legal text, e.g., `flat_rate`. For these inputs, the query provides a traceability link to the legal provision where the constant is defined. The «fromrecord» stereotype is used for inputs derived from a record, e.g., `incomes`. For these, the query is an OCL expression over the underlying domain model. Additional information may be provided along the OCL expression such as the legal text that describes the input, e.g., `distance`. Finally, «fromagent» stereotype is applied to inputs that originate from a legal agent, e.g., `tax_year`. For these, the query provides information about the agent type (role) authorized to provide the input as well as the question that the agent needs to answer.

The legal rule in Fig. 5 takes into account the fact that a taxpayer may have multiple (simultaneous or sequential) employment activities, and thus multiple incomes and work addresses. To correctly compute the FD for a taxpayer, one needs to iterate over *all* incomes gained by the taxpayer and ensure that the computation of the FD portion for each income is consistent with the law. To capture this iterative behavior, we use expansion regions from the AD notation. An expansion region is an activity region that executes multiple times over the elements of an input collection [22]. The legal rule of Fig. 5 has one expansion region with `incomes` as its input collection. UML provides three execution modes for expansion regions: *iterative*, *parallel*, and *stream* [22]. Of these, our profile uses only the iterative mode, marked by the «iterative» stereotype. In this mode, executions are performed sequentially and according to the order of elements in the input collection. The name of the region's expansion node, `inc` in our example, serves as an alias for the iterator element in an individual execution. This alias can be used in the OCL expressions associated with the inputs of the expansion region, e.g., the OCL expression in the query attached to `prorata_period`.

The expansion region in the legal rule of Fig. 5 does not require an explicit output because the assert operation occurs within the expansion region. Our profile allows expansion regions to have an explicit output. This is useful for capturing complex iterative calculations, e.g., computing the total amount of benefits received by the dependents of a taxpayer. We use the «out» stereotype to denote the (explicit) output of an expansion region, if one exists.

Consistency Constraints. To apply our profile in a sound manner, a number of consistency constraints need to hold. We provide a complete list of these constraints in the supplementary material [26]. The consistency constraints are aimed at enforcing the following: (1) Completeness of the information in models of legal rules, e.g., to ensure that the source for each input has been specified through the application of an appropriate stereotype and a query; (2) Mutually exclusive application of certain stereotypes, e.g., to ensure that each input has one and only one source stereotype («fromlaw», «fromrecord» or «fromagent») applied to it; and (3) Restrictions on the structure of ADs. Most notably, these structural restrictions ensure that the flows do not give rise to cyclic paths, and further that only the notational elements allowed by our methodology are being used. All consistency constraints can be enforced as the models are being built.

We next describe how ADs built using our profile are transformed into OCL.

5 Transforming Legal Rules into OCL

In this section, we provide an algorithm to automatically transform ADs to OCL, and illustrate this model-to-text transformation over the Commuting Expenses Deduction (FD) rule discussed in Section 4. The choice of OCL as the target language for the transformation is motivated by OCL being part of the UML and further to benefit from existing testing and simulation frameworks, e.g., [1], that are built around OCL. This section does not cover all the implementation details of our transformation. See the supplementary material for full details [26].

The algorithm for the transformation, named **ADToOCL** and shown in Alg. 1, takes as input an Activity Diagram, $\mathcal{AD}$, and an element $\in \mathcal{AD}$. Specifically, $\mathcal{AD}$ is an instantiation of the UML metamodel fragment for activity modeling, and element is an object within this instantiation. We assume that $\mathcal{AD}$ satisfies the consistency constraints of our profile (Section 4). To ensure consistency between the semantics of activity diagrams and that of the OCL constraints generated from them, we further assume that $\mathcal{AD}$ uses only deterministic decisions.

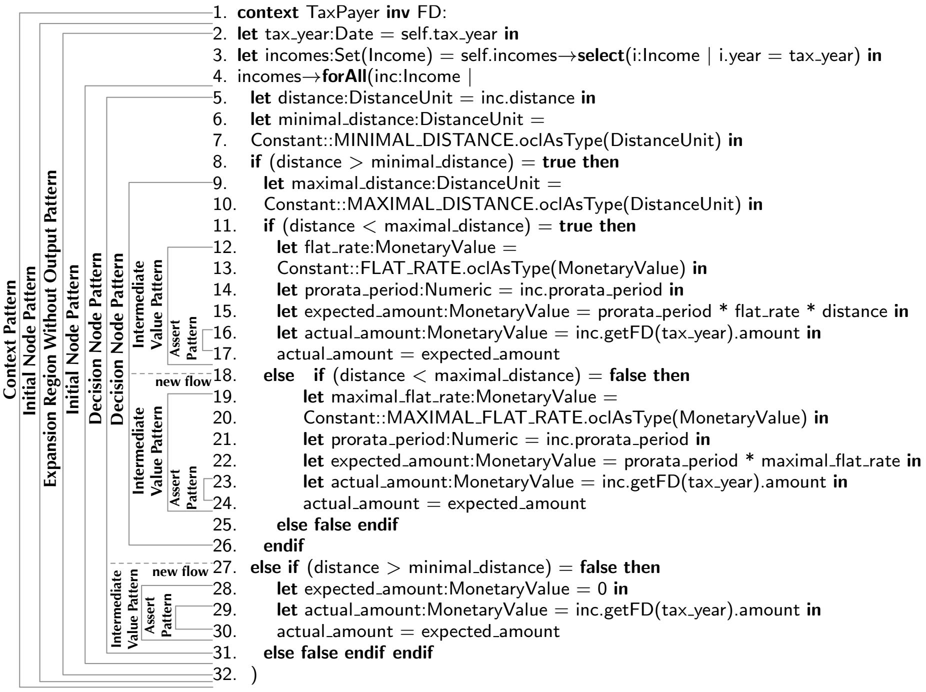
Initially, the algorithm is called over $\mathcal{AD}$ with element pointing to the root `Activity` instance in $\mathcal{AD}$. In Fig. 6, we show the OCL constraint resulting from the application of Alg. 1 to the FD rule of Fig. 5. Note that when our approach is applied, analysts work over the ADs and are not exposed to such complex OCL constraints. The generated constraints are meant for use by OCL engines.

The transformation is based on a set of predefined patterns. These patterns are detailed in the supplementary material [26]. Each pattern is defined as a graph P. If P is matched to a subgraph of $\mathcal{AD}$ rooted at element, the appropriate OCL fragment for P is generated. For example, consider the intermediate value expected_amount in FD. There is a pattern, *Intermediate Value Pattern*, that deals with such values. This pattern has the following shape: an action with the «calculate» stereotype connected by a flow to an intermediate value with the «intermediate» stereotype. The application of this pattern generates a **let** expression which defines an intermediate value based on a given calculation. This pattern is applied three times during the transformation of FD for the three calculations that lead to expected_amount. The OCL fragments for these three applications are shown on L. 15, 22, and 28 of the constraint in Fig. 6.

The transformation process is recursive and mimics a depth-first traversal of the underlying graph of $\mathcal{AD}$. There are three main parts to this process: (1)

Alg. 1: ADToOCL

```
Inputs : (1) An Activity Diagram, $\mathcal{AD}$. (2) An element $\in \mathcal{AD}$.
Output: An OCL string, result.
if (element is NULL) then
    return '' /* Return empty string */
end if
Input declarations
Let $P$ be the transformation pattern applicable to element.
Let $input_1, \ldots, input_n$ be the non-declared inputs required by $P$.
foreach $input_i$ do
    result ← result+ADToOCL($\mathcal{AD}$, $input_i$)
end foreach
if (element is not a DecisionNode) then
    Transforming non-decisions
    Let $st_1$, $st_2$ respectively be the opening and closing OCL fragments obtained from applying $P$.
    Let $next$ be the next element to visit. /* ... chosen based on $P$ and its outgoing flow */
    result ← result+$st_1$+ADToOCL($\mathcal{AD}$, $next$)+$st_2$
    if (element is an ExpansionRegion with an output) then
        Let out denote the output element.
        result ← result+ADToOCL($\mathcal{AD}$, $out$)
    end if
else
    Transforming decisions
    Let $f_1, \ldots, f_m$ be the outgoing flows from element. /* $m \geq 1$ */
    foreach $f_i$ do
        if ($i = 1$) then
            result ← result+'if('+element.name+')='+$f_i$.name+'then'+ADToOCL($\mathcal{AD}$, $f_i$.target)
        else
            result ← result+'else if('+element.name+')='+$f_i$.name+'then'+ADToOCL($\mathcal{AD}$, $f_i$.target)
        end if
    end foreach
    result ← result+'else false'+$\overbrace{\texttt{'endif'+...+'endif'}}^{m \text{ times}}$
end if
return result
```



```
context TaxPayer inv FD:
let tax_year:Date = self.tax_year in
let incomes:Set(Income) = self.incomes→select(i:Income | i.year = tax_year) in
incomes→forAll(inc:Income |
  let distance:DistanceUnit = inc.distance in
  let minimal_distance:DistanceUnit =
  Constant::MINIMAL_DISTANCE.oclAsType(DistanceUnit) in
  if (distance > minimal_distance) = true then
    let maximal_distance:DistanceUnit =
    Constant::MAXIMAL_DISTANCE.oclAsType(DistanceUnit) in
    if (distance < maximal_distance) = true then
      let flat_rate:MonetaryValue =
      Constant::FLAT_RATE.oclAsType(MonetaryValue) in
      let prorata_period:Numeric = inc.prorata_period in
      let expected_amount:MonetaryValue = prorata_period * flat_rate * distance in
      let actual_amount:MonetaryValue = inc.getFD(tax_year).amount in
      actual_amount = expected_amount
    else   if (distance < maximal_distance) = false then
          let maximal_flat_rate:MonetaryValue =
          Constant::MAXIMAL_FLAT_RATE.oclAsType(MonetaryValue) in
          let prorata_period:Numeric = inc.prorata_period in
          let expected_amount:MonetaryValue = prorata_period * maximal_flat_rate in
          let actual_amount:MonetaryValue = inc.getFD(tax_year).amount in
          actual_amount = expected_amount
      else false endif
    endif
  else if (distance > minimal_distance) = false then
      let expected_amount:MonetaryValue = 0 in
      let actual_amount:MonetaryValue = inc.getFD(tax_year).amount in
      actual_amount = expected_amount
    else false endif endif
)
```

Fig. 6. Generated OCL expression for the example of Fig. 5 (FD)

input declarations (Alg. 1, L. 4-8); (2) transformation of all elements other than decisions (Alg. 1, L. 10-16). Within this class of elements, additional processing is necessary for expansion regions to propagate their output if they have one (Alg. 1, L. 13-16); and (3) transformation of decision nodes (Alg. 1, L. 18-26).

The first part of the transformation process concerns identifying all inputs to be declared before transforming a given element (Alg. 1, L. 5). Each such input is transformed into a **let** expression (Alg. 1, L. 6-8). To illustrate, consider the decision `distance` > `minimal_distance` in FD. The inputs to this decision are transformed into L. 5-7 of the constraint in Fig. 6. This is performed before the transformation of the decision itself (Fig. 6, L. 8). An input may have dependencies to other inputs, e.g., `incomes` (Fig. 6, L. 3) depends on `tax_year` (Fig. 6, L. 2). Such dependencies are handled through the recursive call of L. 7 in Alg. 1.

The second part of the process handles non-decisions. This is where the initial call to Alg. 1 begins to unwind. The initial call is handled by the *Context Pattern*, which transforms the context information attached to an `Activity` instance via the «`context`» stereotype. There are no inputs associated with the *Context Pattern*. Handling the pattern thus reduces to executing L. 10-12 of Alg. 1. The opening OCL fragment ($\mathtt{st}_1$) resulting from the application of this pattern is L. 1 of Fig. 6; the closing fragment ($\mathtt{st}_2$) is empty. Then, on L. 12 of Alg. 1 a recursive call is made with *next* set to the initial node of FD. The unwinding of this recursive call generates the remainder of the OCL constraint (Fig. 6, L. 2-32). On the left side of Fig. 6, we mark the scope of each recursive call and the respective pattern. To avoid clutter, calls that handle input declarations are not marked.

The third and final part of the process transforms decisions into if-then-else statements. This part is analogous to what we previously described.

We have implemented our transformation using Acceleo [10] – a model-to-text transformation tool for Eclipse. Our Acceleo implementation is closely aligned with the way we present the transformation in Alg. 1. While we currently support only OCL as the target language for the transformation, it is possible to modify our text generation rules to support other languages, e.g., Alloy [16].

6 Evaluation

We report on an industrial case study where we apply our approach to *LITL*. The case study is an initial step towards answering the following Research Questions (RQs): ***RQ1.*** *Is the approach expressive enough to model complex legal rules?* ***RQ2.*** *Is the level of effort required by our approach reasonable?* And, ***RQ3.*** *Are the ADs built using our approach structurally less complex than OCL constraints written directly?* In the longer term, we plan to perform more extensive user studies to evaluate the approach in a more thorough manner.

Our case study builds on an initiative by the Government of Luxembourg to improve its eGovernment services in the area of taxation. One of the main objectives of the initiative is to ensure that these services remain *verifiably compliant* with the tax law as the law evolves. A key prerequisite for verification of compliance is to have analyzable models of the tax law. Our case study develops such models for a substantial fragment of the income tax law. The case study was conducted in collaboration with our public service partner, CTIE.

Study selection and execution. Our study concerns a set of legal rules from *LITL*. Luxembourg has two complementary schemes for income taxes: (1) withholding taxes from salaries, and (2) assessing taxes based on a declaration. Our study focuses on the former scheme. The basis for withholding is a *tax card*, detailing the tax deductions and credits that apply to an income. Deductions are expense items subtracted from the gross income before taxes. Credits are items applied either against the taxes due or paid to the taxpayer in cash. A tax card provides information about five deductions and three credits. The deductions are for commuting expenses (FD), miscellaneous expenses (FO), spousal expenses (AC), extraordinary expenses (CE), and special expenses (DS). CE is decomposed into three sub-categories and DS into six. The credits are for salaried workers (CIS), pensioners (CIP), and single parents (CIM).

The above deductions and credits give rise to 15 legal rules. We applied our methodology described in Section 3 for expressing these rules. This resulted in a domain model and 15 ADs built using our profile. The domain model has 7 packages, 61 classes, 15 enumerations, 106 attributes, and 24 operations. The distribution for the number of elements in the ADs is given in the box plot of Fig. 7. The element count for each AD is the sum of the number of inputs, outputs, decisions, actions, flows, intermediate variables, expansion regions, and constraint/comment boxes.

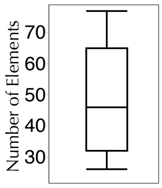


Fig. 7. # of AD elements (distribution)

Discussion. We next discuss the RQs that motivated our study. The three tax credits in our study (CIS, CIP, and CIM) were used previously in our investigation of OCL complexity factors (Section 2) along with six other rules that are unrelated to the case study. Since we had a priori knowledge about the tax credits, the AD models for the tax credits are uninteresting for RQ1. To mitigate learning effects, we further exclude these three models when discussing RQ2.

RQ1. Our profile provided enough expressiveness to conveniently capture the legal rules in our study. One of the factors we considered in our models was to avoid nested structures, particularly nested expansion regions. Although our profile and OCL transformation can handle nesting, models containing nested structures can be hard to comprehend. In our study, we could avoid nesting in all models by choosing a suitable OCL context for each of the legal rules.

RQ2. We are interested in measuring the level of effort as an indicator for whether the approach has a realistic chance of adoption in practice. The ADs were built by the first author, who has 6 years of formal training in computer science and 3 years of experience in MDE. The models were built following a half-day tutorial on personal income taxes by legal experts. The domain model was developed simultaneously with the ADs, as suggested in Section 3. Developing the 12 ADs for tax deductions took $\approx$ 40 person-hours (ph) including the effort spent on the domain model. This is an average of 3.3 ph per AD. The 3 ADs for tax credits took $\approx$ 7 ph to build, i.e., an average of 2.3 ph per AD. Only the tax deductions are representative in terms of effort, due to reasons discussed earlier. Once built, the ADs were presented to a group of six legal experts in a half-day training and walkthrough session. We received positive feedback from the legal

Table 2. Comparison of complexity: direct use of OCL vs. OCL fragments in ADs

		FD		FO		AC		CE1		CE2		CE3		DS1		DS2		DS3		DS4		DS5		DS6		CIS		CIP		CIM	
		M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A
Complexity metrics	C_1 *(# navigations)*	11	5	12	5	11	10	10	5	6	2	12	6	6	6	8	3	17	11	9	3	4	3	9	6	6	2	6	3	13	7
	C_2 *(# if statements)*	3	0	5	0	6	0	7	0	4	0	4	0	4	0	3	0	6	0	4	0	3	0	3	0	2	0	2	0	3	0
	C_3 *(# collection_op's)*	3	0	6	0	3	3	17	2	8	0	10	1	8	3	4	1	12	3	8	0	3	1	5	1	6	4	6	4	5	1
	C_4 *(# iterative_op's)*	3	1	5	1	8	6	9	2	5	1	7	1	6	4	4	2	7	6	5	1	3	2	4	2	2	1	2	1	5	4

experts involved in our study; however, we have not yet conducted a detailed user study to thoroughly assess our approach. We consider the overall effort to be worthwhile as the resulting models provide a complete characterization of the tax card, which applies to a large majority of the taxpayers.

RQ3. Our profile limits the use of OCL to the inputs, formulas, and statements of ADs. In this way, the profile to a large extent shields users from OCL and the structural complexity of OCL expressions. Reynoso et al. [23] argue that reductions in OCL structural complexity bring about reductions in cognitive complexity and improvements in understandability. The aim of RQ3 is to measure the value of our profile in terms of structural complexity reduction when compared to the situation where legal rules are directly written in OCL. This comparison provides preliminary insights as to whether our profile can result in more intuitive and understandable specifications of legal rules.

To answer RQ3, the second author manually wrote constraints for the tax deductions, in a similar manner to the tax credits (Section 2). We then compared these constraints to the OCL expressions used in the ADs, i.e., the OCL expressions to which the users of our profile are exposed. For the comparison, we selected a subset of the OCL structural complexity metrics proposed by Reynoso et al. [23]. Our selection was driven by what we deemed relevant to the complexity factors observed in our field study (Section 2). Specifically, we consider the following metrics: number of navigations (C_1), number of if-then-else statements (C_2), number of operations on collections, e.g., `any`, `sum`, `excludes` (C_3), and number of iterative operations, e.g., `select`, `forAll` (C_4). C_1 and C_2 respectively reflect the *navigation* and *branching* complexity factors; C_3 and C_4 both relate to the *iteration* complexity factor.

Table 2 shows the metrics, across all the deductions and credits, for manually-written OCL constraints (denoted, **M**) vs. the OCL fragments used in the ADs (denoted, **A**). As the table suggests, the ADs built using our approach lead to reductions in structural complexity. In particular, the AD's reduce on average: C_1 by 45%, C_2 by 100%[2], C_3 by 72%, and C_4 53%. The structural complexity that carries over to the ADs is primarily caused by the OCL expressions that define the inputs to the ADs. To validate the ADs with non-software engineers, one can replace these expressions with intuitive descriptions without any impact on the ADs. Finally, we need to emphasize that the complexity reductions seen are only suggestive of benefits, but not definitive evidence for them. Further empirical validation remains essential to determine whether the complexity reductions indeed translate into improved understandability.

[2] The 100% reduction is due to the fact that the ADs in our study do not contain if-then-else statements, as all branching behaviors are captured using decision nodes.

7 Related Work

In this section, we compare our approach with several areas of related work.

Legal rules. van Engers et al. [28] express legal rules via OCL; however, they use OCL directly in their specifications. Our approach provides a model-based solution for expressing legal rules. Breaux et al. [5] describe a rule-based framework for legal requirements. Nevertheless, they do not operationalize these requirements. The legal rules in our approach are in contrast executable.

Verification of legal compliance. Compliance verification has long been studied for *business processes* in domains such as healthcare [12, 13] and finance [15]. Few strands however address compliance for *software systems*. Notable among these is work by Maxwell et al. [19], where they derive system compliance rules from legal texts, and by Breaux [4] where he extracts finite state machines from legal texts to guide system compliance checking. These earlier strands focus on capturing the functional requirements of software systems. Our approach instead focuses on modeling software systems in terms of inputs and expected outputs (as envisaged by the law), and irrespectively of specific system functions.

Visualization of logical languages. Bottoni et al. [3] and Stein et al. [27] propose visualizations for OCL, and Amàlio et al. [2] – for the Z language [25]. These approaches are not tailored to legal rules and lack means for addressing the information requirements and complexity factors discussed in Section 2.

Model-to-OCL transformation. Cabot et al. [6] construct OCL transformations of domain-specific language rules, and Milanović et al. [20] derive OCL constraints from integrity rule models. These approaches neither address legal rules nor tackle the transformation of activity diagrams, as done in our approach.

8 Conclusion

We proposed a UML-based approach for modeling procedural legal rules. The key component of the approach is a profile for activity diagrams. To enable automated compliance analysis, we defined a transformation that produces OCL specifications from activity diagrams built using our profile. We presented a preliminary evaluation of our approach.

Our approach focuses on prescriptive legal frameworks. In the future, we would like to investigate how and to what extent our approach can accommodate declarative frameworks and notions such as permissions and obligations. Another topic for future work is to conduct more field studies and generalize our UML profile to a larger set of legal domains. Further, a more thorough evaluation of our approach is essential. In particular, the legal experts in our study underwent training before they were able to understand our models. Legal experts trained in other approaches, e.g., mathematical logic, may have done equally well. User studies are necessary to determine what advantages and disadvantages our approach offers compared to the direct use of logic. Finally, we plan to study how our models can support automated analysis tasks such as simulation.

Acknowledgments. Financial support was provided by CTIE and FNR under grant number FNR/P10/03. We are grateful to members of Luxembourg Inland Revenue Office and CTIE for sharing their valuable insights with us.

References

1. Ali, S., Zohaib Iqbal, M., Arcuri, A., Briand, L.: Generating test data from OCL constraints with search techniques. IEEE Transactions on Software Engineering 39(10), 1376–1402 (2013)
2. Amàlio, N., Kelsen, P., Ma, Q., Glodt, C.: Using VCL as an Aspect-Oriented Approach to Requirements Modelling. Transactions on Aspect-Oriented Software Development 7, 151–199 (2010)
3. Bottoni, P., Koch, M., Parisi-Presicce, F., Taentzer, G.: Consistency checking and visualization of OCL constraints. In: Proc. of 3rd Intl. Conf. on The Unified Modeling Language: Advancing the Standard (UML'00). pp. 294–308 (2000)
4. Breaux, T.: A method to acquire compliance monitors from regulations. In: Proc. of 3rd Intl. Wrkshp. on RE and Law (RELAW'10). pp. 17–26 (2010)
5. Breaux, T.: Exercising due diligence in legal requirements acquisition: A tool-supported, frame-based approach. In: Proc. of 17th IEEE Intl. Requirements Engineering Conf. (RE'09). pp. 225–230 (2009)
6. Cabot, J., Clarisó, R., Guerra, E., Lara, J.: A UML/OCL framework for the analysis of graph transformation rules. Software and Systems Modeling 9(3), 335–357 (2010)
7. Cabot, J., Clariso, R., Riera, D.: Verification of UML/OCL class diagrams using constraint programming. In: Proc. of 2008 IEEE Conf. on Software Testing Verification and Validation Wrkshp. (ICST'08). pp. 73–80 (2008)
8. Corbin, J., Strauss, A.: Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. SAGE Publications, 3rd edn. (2008)
9. Dey, I.: Qualitative data analysis - A user-friendly guide for social scientists. Routledge (1993)
10. Eclipse Foundation: Acceleo - transforming models into code. `http://www.eclipse.org/acceleo/`, last accessed: March 2014
11. Eclipse Foundation: Ecore tools. `http://www.eclipse.org/ecoretools/`, last accessed: March 2014
12. Ghanavati, S., Amyot, D., Peyton, L.: Towards a framework for tracking legal compliance in healthcare. In: Proc. of 19th Intl. Conf. on Advanced Information Systems Engineering (CAiSE 2007). pp. 218–232 (2007)
13. Goedertier, S., Vanthienen, J.: Designing compliant business processes with obligations and permissions. In: Proc. of 7th Wrkshp. on Business Process Management (BPM'06). pp. 5–14 (2006)
14. Gov. of Luxembourg: Modified income tax law of december 4, 1967 (2013)
15. Hassan, W., Logrippo, L.: Requirements and compliance in legal systems: a logic approach. In: Proc. of 1st Intl. Wrkshp. on RE and Law (RELAW'08). pp. 40–44 (2008)
16. Jackson, D.: Software Abstractions Logic, Language, and Analysis. The MIT Press (2006)
17. Korherr, B., List, B.: Extending the UML 2 activity diagram with business process goals and performance measures and the mapping to BPEL. In: Proc. of 2nd Intl. Wrkshp. on Best Practices of UML (ER BP-UML'06). pp. 7–18 (2006)
18. Larman, C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition). Prentice Hall (2004)
19. Maxwell, J., Anton, A.: Checking existing requirements for compliance with law using a production rule model. In: Proc. of 2nd Intl. Wrkshp. on RE and Law (RELAW'09). pp. 1–6 (2009)

20. Milanovic, M., Gasevic, D., Giurca, A., Gerd, W., Devedzic, V.: Towards sharing rules between OWL/SWRL and UML/OCL. Electronic Communications of European Association of Software Science and Technology 5, 2–19 (2007)
21. Miles, M., Huberman, A.: Qualitative data analysis: an expanded sourcebook. SAGE (1994)
22. Object Management Group: UML 2.2 superstructure specification (2009)
23. Reynoso, L., Genero, M., Piattini, M.: Towards a metric suite for OCL expressions expressed within UML/OCL models. Journal of Computer Science and Technology 4(1), 38–44 (2004)
24. Ruiter, D.: Institutional Legal Facts: Legal Powers and Their Effects. Kluwer Academic Publishers (1993)
25. Smith, G.: The Object-Z specification language. Kluwer (2000)
26. Soltana, G., Fourneret, E., Adedjouma, M., Sabetzadeh, M., Briand, L.: Using UML for modeling legal rules. Tech. Rep. TR-SnT-2014-3, Interdisciplinary Centre for Security, Reliability and Trust (SnT) (March 2014), http://people.svv.lu/soltana/Models14.pdf
27. Stein, D., Hanenberg, S., Unland, R.: A graphical notation to specify model queries for MDA transformations on UML models. In: Proc. of 2nd Conf. on Model Driven Architecture: Foundations and Applications (MDAFA'04). pp. 60–74 (2004)
28. van Engers, T., Gerrits, R., Boekenoogen, M., Glassée, E., Kordelaar, P.: POWER: using UML/OCL for modeling legislation - an application report. In: Proc. of 8th Intl. Conf. on Artificial Intelligence and Law (ICAIL'08). pp. 157–167 (2001)