Artificial Intelligence and Law (2023) 31:829–863
https://doi.org/10.1007/s10506-022-09339-2

Legal document assembly system for introducing law students with legal drafting

Marko Marković[1] · **Stevan Gostojić**[1]

Accepted: 23 October 2022 / Published online: 16 November 2022

Abstract
In this paper, we present a method for introducing law students to the writing of legal documents. The method uses a machine-readable representation of the legal knowledge to support document assembly and to help the students to understand how the assembly is performed. The knowledge base consists of enacted legislation, document templates, and assembly instructions. We propose a system called LEDAS (LEgal Document Assembly System) for the interactive assembly of legal documents. It guides users through the assembly process and provides explanations of the interconnection between input data and claims stated in the document. The system acts as a platform for practicing drafting skills and has great potential as an education tool. It allows teachers to configure the system for the assembly of some particular type of legal document and then enables students to draft the documents by investigating which information is relevant for these documents and how the input data shape the final document. The generated legal document is complemented by a graphical representation of legal arguments expressed in the document. The system is based on existing legal standards to facilitate its introduction in the legal domain. Applicability of the system in the education of future lawyers is positively evaluated by the group of law students and their TA.

Keywords Legal education · Legal documents · Document assembly · Document drafting · Knowledge representation

1 Introduction

Document drafting is one of the most frequent tasks in the legal profession. Lawyers use these documents to communicate some information to other parties in legal proceedings. There are a variety of legal document types due to the broadness of the legal system. Some of them are contracts, wills, judgments, indictments, claims,

✉ Stevan Gostojić
gostojic@uns.ac.rs

[1] Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia

appeals, bills, and orders. The legal documents need to meet some standards usually defined by legislation or by the legal profession. Although legal documents are written in the natural language it uses its special type sometimes referred to as the Language of the Law (Mellinkoff 1963).

The drafting of legal documents is taught at the law schools, but for mastering drafting skills lawyers still have to practice document drafting at the beginning of their career. To bridge that gap young lawyers usually do not write the documents from scratch, but rather rely on documents created by their senior colleagues. Branting and Lester (1996) emphasize that lawyers usually search for a document that is written before and that shares some similarities with the document they should compose. These old documents serve as some kind of document templates where some corrections have to be made to produce the needed document.

The above method for document drafting is also useful for experienced lawyers as a shortcut that decreases the time for document drafting in comparison to writing an entire document. Unfortunately, this approach sometimes leads to errors if a lawyer misses to make all necessary changes and some old data remains in the new document (Petro 2015; Allee and Elsig 2019; Henley 2001). Furthermore, choosing an old document bears a risk of using the style that is outdated or citing legislation that has ceased to have an effect.

The process of teaching legal writing has evolved through the years. Barnett (2006) noticed that it has moved from a product-oriented approach, focused on the final results in the students' assignments, to the process-oriented approach that encourages students to develop their ideas. Process-oriented teaching enables teachers to provide students with guidance during the writing process. This helps students to refine their writing skills before completing the assignment which in the end improves the quality of their work. In classrooms with a large number of students, it may be difficult for teachers to provide feedback to every student. Keene (2015) discusses some additional improvements that are beneficial for legal writing courses. Teachers with the help of their teaching assistants could guide students more efficiently through the writing process or even peers could give feedback to each other. However, one of the most important steps would be introducing more courses dealing with legal writing in the law schools' curriculum.

Although legal writing courses are being taught at law schools, Winek (2021) reports that young lawyers often have difficulties with legal writing in their workplaces. Some of the deficiencies of their writing skills are wordiness and unclear writing, with incomplete and poor document organization. Similarly, law students also have an impression that their writing skills are insufficient for application in legal practice. Winek suggests that more activities that involve practicing skills related to legal writing should be included in the law school's curriculum.

Pollman (2014) points out that students are usually taught legal writing by analyzing some legal documents with their teacher and then writing similar documents themselves. It requires students to learn how an exemplary document is organized and to understand how the same type of legal document should be written for some other legal issue. It can produce cognitive overload if these two steps are taught

simultaneously. Pollman proposes model-based learning letting students separately learn principles for writing a legal document for a single type of legal issue and how to generalize those principles. This approach helps them to first focus only on important rules that play a role in composing some document. After that, students can be taught how to generalize these rules and how to apply them to the next legal issue or even another type of legal document.

During the last few years pandemic of COVID-19 heavily affected the educational process in the world. It forced teachers to move from classroom teaching to online teaching changing the way of interaction with students. Both teachers and students had to find a way to cope with the new digital environment increasing the demand for educational software tools. In these circumstances, interactive tools for self-learning could be beneficial for both sides enabling students to prepare for the next class at their own pace and reducing the teacher's effort in explaining a new lesson to students.

This paper discusses a method to support teaching law students legal document drafting. The method involves a knowledge base consisting of legal knowledge relevant to document assembly (legal rules, document templates, assembly configuration). The legal rules represent both substantive and procedural legal norms that affect the content of assembled documents. The document templates represent the generalized design of legal documents. We aim to avoid the black box approach for document assembly that outputs legal documents with no explanations for their genesis. Instead, we put focus on the formal representation of legal knowledge to support automatic reasoning over the given facts, provide explanations for produced inferences, and construct the document on the basis of inferred data. We intend to design a document assembly method for creating legal acts in a transparent manner revealing the reasoning process that occurs behind claims stated in the document. This approach should not replace traditional legal writing courses but it rather enables students to make the first steps in learning legal writing by themselves, and allows once prepared system configuration to be run multiple times.

The rest of the paper is organized as follows. The next section describes research related to legal document drafting and its application in education. The Sect. 3 proposes a method for teaching the drafting of legal documents. The Sect. 4 presents a prototype system for teaching legal document drafting and brings evaluation results. The Sect. 5 discusses obtained results. Finally, the last section gives conclusions and directions for future work.

2 Related work

This section reviews some previous research related to the legal document drafting.

Canick (2014) noticed the gap between skills taught at law schools and the expertise that a legal professional should have. Document drafting is recognized as a legal communication skill that all lawyers need to perform no matter what legal area they

work in. Students need to be taught the importance of precise language and that every word they put in the document matters in contrast to communications they use in informal contexts. The author suggests the introduction of technology into legal education to prepare students for efficient work. Online document drafting tools are proposed to give the students an insight into how to benefit from technology in their everyday work.

There are numerous commercial solutions in the legal tools market that perform document assembly including Contract Express (Thomson Reuters 2022), HotDocs (AbacusNext 2022), ClauseBase (2022), Afterpattern (2022), Outlaw (2022), Legito (2022), Woodpecker (2022), and Draftomat (2022). Users enter case data by filling out forms and these tools put this data in predefined templates to create a legal document. Most of these tools are either web-based systems or provide integration with document editing software allowing users to work in an environment they are familiar with. Some of these tools provide advanced features, like collaborative work, libraries of clauses, and the use of variables and logic expressions to control if some text part (e.g. contract clause) should be included in the generated document. Aiming to support lawyers in the legal document drafting tasks and to save their time, these tools are suitable for use by experienced lawyers.

In (Lauritsen 1993) several types of knowledge involved in legal document drafting are identified. It includes knowledge that helps in presenting information the document is intended to communicate, instantiation of document class for a particular case, selecting components to build the document, applying supplementary non-legal-specific knowledge (e.g. language rules and writing styles), fulfilling the purpose of the document, and arranging document sections and its sentences. However, expressing all these knowledge types in a way that supports automated document drafting may require serious undertaking. Markup languages are recognized as a powerful formalism for representing some drafting knowledge types. Using these formalisms can help to produce so-called "knowledgable documents" i.e. documents that include extra information about the content. This information facilitates the management of legal documents by legal information systems.

In (Lauritsen and Gordon 2009) a method for modeling legal documents is proposed to construct a theoretical framework for document drafting. The method uses modeling statements to define requirements that a class of legal documents should satisfy. Ideally, these statements bring completeness and consistency to the document model. Nevertheless, modeling statements may originate from different sources of drafting knowledge resulting in incompleteness or some inconsistencies in document models. Furthermore, if a document meets the requirements defined by the model, the final decision regarding its quality should be left to the lawyers. Even then, they may come to different conclusions due to subjectivity in assessment. Although the proposed method is general and domain-independent, it is likely to produce domain-specific models at some point in modeling.

Branting et al. (1998) use discourse structure as the basis for generating judicial documents. This structure consists of two types of operators, namely illocutionary and rhetorical. Illocutionary operators represent goals that the author

intends to reach and rhetorical operators represent conventions regarding discourse and writing style of the document's genre. As a proof of concept, the authors have developed a prototype application called Docu Planner. The creation of judicial documents consists of two phases, the document planning phase which adjusts document structure to the specifics of given case facts and the document drafting phase which produces the final document. The application provides two modes for document creation, text mode and web mode. The text mode enables generating documents for printing purposes and the web mode uses hypertext for document formatting. In the next release, the Docu Planner 2.0 (Branting et al. 1999) brings explanations to text parts of the generated document. In that way, the user can get additional information on reasons for including that part of the text in the document.

Visualization of information stated in the legal documents can be useful for readers to help them better understand the legal text. In (Passera et al. 2014) the authors present a set of tools for visualization of contract clauses. One of these tools shows a graphical representation of the contract's timeline from contract formation to termination. Another tool visually presents a payment plan agreed upon in the contract as a sequence of installments. The third tool shows a diagram of liquidated damages depending on the delay in delivery of the goods specified in the contract. The authors emphasize the benefits of these tools for lawyers in designing contracts and also for non-lawyers as an instrument that helps them understand the meaning of contract clauses. Integration of visualization tools in document assembly systems is suggested.

Ashley (2009) investigates how the visual representation of arguments helps law students to understand legal reasoning in oral arguments presented in the courtroom. Students analysed the transcriptions of argumentation to identify argument parts. One group of students was allowed to use a tool for visual designing argumentation LARGO while the other group was making the text notes. The results show that analysis of argumentation using the visual tool is beneficial for students' understanding of argumentation, mostly for the students with lower LSAT (Law School Admissions Test) scores.

Marković and Gostojić (2020) propose a knowledge-based method for the assembly of documents in enterprise settings. The authors present the system that guides users through the assembly process for business documents whose content depends on legislation in force. It speeds up the writing of legal documents such as service contracts and also helps less experienced lawyers to create these documents. Generated documents are semantically annotated improving interoperability between enterprise information systems. This method shares some similarities with the approach presented in this paper.

The previous research on automated legal drafting shows different approaches for producing legal documents from input data. There are varieties of methods for formal representation of the drafting knowledge. It mainly depends on the purpose of the system that uses the knowledge and target users' profiles. Theoretical research on legal drafting identifies some important types of drafting knowledge and suggests

creating a knowledge base using formal representations of drafting knowledge. Most of the legal drafting systems available on the market are created for legal professionals to simplify their everyday drafting activities and help them to create legal documents faster. Drafting tools developed specifically for legal education are less common. Some of these tools provide students with textual or visual explanations of the generated content to help them understand the drafting results. Such tools show positive effects on the education of future lawyers and the potential for introduction in legal writing courses.

3 Methodology

This paper proposes a method for legal document assembly that helps law students to learn and exercise how legal documents can be constructed.

This method uses a knowledge base representing the legal knowledge needed for legal document writing. The complexity of the legal system brings some difficulties to the acquisition of legal knowledge involved in legal writing. Lawyers gain this knowledge at law schools and later while practicing law. It involves legal theory, legislative texts, teachers' instructions, senior lawyers' advice, reusable knowledge embedded in legal precedents, etc. Furthermore, there is a broad range of legal areas and a diversity of legal document types that lawyers work with.

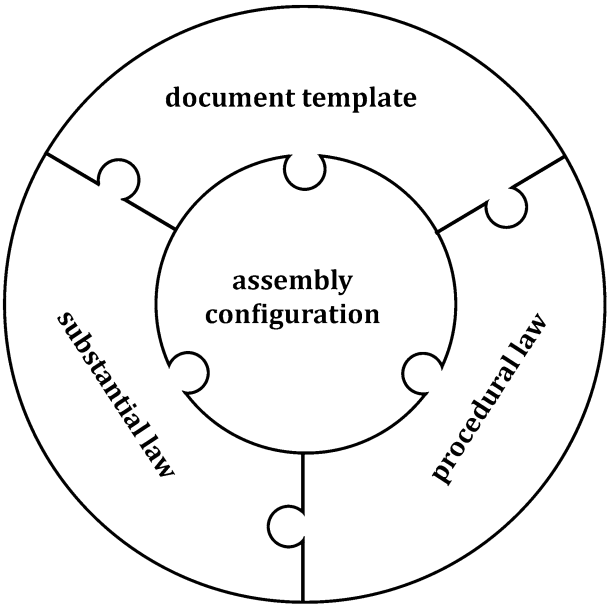

Fig. 1 The structure of legal document assembly knowledge

Thus, this knowledge can be seen as a two-dimensional space, where one dimension represents a legal area, and another dimension is related to a particular document type. In this two-dimensional system of legal knowledge, an intersection of the document type and the legal area determines the knowledge needed for the document assembly. Therefore, to support legal document assembly the knowledge base should model legal knowledge related to the target document type in a chosen legal area.

Our method distinguishes four parts of the legal document assembly knowledge (Fig. 1). The first part of this knowledge is legislation that regulates a relevant legal area. This part of knowledge represents legal norms from substantial sources of law governing the given legal area.

The second part of the knowledge is regulations prescribing constraints regarding the content and the structure of a specific legal document type. It represents legal norms originating from procedural sources of law relevant to the concrete document type.

The third part is tacit knowledge of legal writing for a particular type of legal documents. This part of knowledge represents the writing skills of experienced lawyers and can be modeled using document templates.

The fourth part interconnects the previous three knowledge parts and defines a procedure for document formation. It combines knowledge needed for document assembly with instructions for students and is organized as a sequence of assembly steps.

We adopt the modular design for the knowledge base allowing changes in any of the knowledge base parts to extend support for some new legal area, document type, or the flow of document assembly.

Additionally, document assembly could involve other knowledge representations. In this paper we consider the assembly of legal documents in civil law legal systems. For the assembly of documents in common law jurisdictions, where precedents are legally binding sources of law, knowledge from prior decisions should be formally represented and supported by the assembly method. For example, machine-readable representation of judicial opinions could be performed using an ontology as proposed in (Ceci and Gangemi 2016).

We seek to capture and formally represent all parts of legal document drafting knowledge, as well as to enable the application of this knowledge in document assembly. The objective of this method is to introduce the legal drafting process to law students and help them to explore how changes in input data cause variations in assembled documents.

We aim to collect the knowledge relevant for legal document assembly on several legal issues due to the complexity of the legal system. A broader range of legal document types could bring some differences in the creation of legal acts specific to that document types. We focused on the assembly of legal acts in criminal proceedings because claims in these documents are mostly supported by physical evidence making it easier for students to understand the logical structure of the documents. In particular, we chose to analyze indictments as legal acts formed in the early phases

of criminal proceedings resulting in less dependence on claims in other documents. Although indictments are one type of legal documents the method proposed in this paper can be applied to other judicial documents because they are strictly based on substantiative and procedural law. For the same reason, this approach is also applicable in drafting contracts. However, legislative documents as one of the most complex document types are not suitable for creation using this method due to their general and less constrained content.

For explanations of our method, we give examples for knowledge representation. Without losing generality we focus on the legal knowledge that belongs to the legal system that we are the most familiar with i.e. legislation of the Republic of Serbia. The laws used in these examples include The Criminal Code (2005), The Criminal Procedure Code (2011), The Law on Road Traffic Safety (2009), and The Law on Organization of Courts (2008).

In the next subsections, we briefly explain how every part of the knowledge base is constructed and how legal document assembly is performed.

3.1 Legal rules

The legal rules should formally represent the legal knowledge given by legal norms in legislation. Considering assembly of indictments most of the legal norms involved in the assembly of this document type are substantial and procedural criminal law. While substantial criminal law defines criminal offenses and penalties, procedural criminal law determines types of criminal proceedings and courts' jurisdiction. To transform these sources of law into a machine-readable format we start by representing relevant legal norms in predicate logic.

Legal norms are governing human behavior and consist of two parts, the disposition that describes a mandatory behavior, and the sanction that prescribes the consequence for violation of that behavior. Both disposition and sanction can be represented as legal rules. The applicability of these provisions is usually determined by the exact circumstances that have to occur. We find predicate logic appropriate formalism for the representation of these circumstances.

For example, the substantial provision defining that excessive driving speed in a populated area is when a person drives a vehicle at a speed greater than 50 km/h, can be represented using predicate logic as:

$$\forall x \forall y \, (S(x, y) \wedge y > 50 \rightarrow E(x)) \tag{1}$$

where x represents a person, y represents the speed, the predicate S is the person's driving speed and the predicate E implies that the person x is driving over the speed limit.

Furthermore, if a person violating the traffic rules causes bodily injuries to some other person or produces material damage greater than 200.000 RSD, this act can be qualified as a criminal offense represented by the formula:

$$\forall x \forall y \forall q \, (E(x) \wedge (I(x, l) \vee (D(x, q) \wedge q > 200000)) \rightarrow C(x, a)) \tag{2}$$

where E denotes the same predicate from the previous formula, the predicate I asserts that person x caused injuries to others while l stands for light injuries, the predicate D represents that person x caused material damage in the amount of q, and the predicate C denotes that the person x have committed the criminal offense identified by the constant value a.

The penalty for this criminal offense is imprisonment of up to 3 years. We can define this sanction as follows:

$$P(a, 3) \tag{3}$$

where the predicate P represents the maximum penalty for the criminal offense a, as imprisonment for the duration of 3 years.

An example of procedural provision is a determination of jurisdiction of basic courts for criminal offenses with a penalty less than or equal to 10 years of imprisonment. It can be represented by the following formula:

$$\forall x \forall y \forall z \, (C(x, y) \wedge P(y, z) \wedge z \leq 10 \rightarrow B(x, y)) \tag{4}$$

where predicate C represents that defendant x has committed the criminal offense y, predicate P states that for the criminal offense y the penalty is z years of imprisonment, and predicate B represents that a basic court has jurisdiction in the case for the defendant x and the criminal offense y.

The jurisdiction of higher courts for criminal offenses with a penalty greater than 10 years of imprisonment is determined by the formula:

$$\forall x \forall y \forall z \, (C(x, y) \wedge P(y, z) \wedge z > 10 \rightarrow H(x, y)) \tag{5}$$

where C and P are the same predicates used in the previous formula, while the predicate H represents the jurisdiction of a higher court in the case against defendant x for the criminal offense y.

An example of another procedural provision that grants jurisdiction to higher courts in juvenile criminal cases can be represented as:

$$\forall x \forall y \forall z \, (C(x, y) \wedge A(x, z) \wedge z < 18 \rightarrow H(x, y)) \tag{6}$$

where predicate C represents that the defendant x has committed the criminal offense y, the predicate A represents that age of the defendant x is z, and the predicate H states that the case will be judged by the higher court.

When the defendant is a minor and the penalty for a committed criminal offense is less than 10 years of imprisonment, the formulas (4) and (6) may lead to contradictory conclusions because the former formula (4) supports the jurisdiction of a basic court for the criminal case while the latter formula (6) supports jurisdiction of a higher court. In this case, the legislator grants the jurisdiction to the higher court because the provision represented by formula (6), due to its specificity in terms of

the defendant's age, represents an exception to the provision represented by formula (4).

The important characteristic of legal rules is their non-monotonic nature that allows the retraction of inferences when new information appears. For example, if some legal rule makes an exception to the other legal rule it also retracts its conclusion when conditions for that exception are met. In the same way, the legal principles known as *lex specialis*, *lex superiori*, and *lex posteriori* prioritizes one legal rule over another. To support nonmonotonicity in our knowledge base, we use defeasible logic.

Using the defeasible logic to represent rules determining the jurisdiction of basic and higher courts the formulas (4), (5), and (6) becomes:

$$\begin{aligned} &r_1\colon commited\_offence(Defendant, Offence),\\ &max\_imprisonment(Offence, Years),\ Years <= 10\\ &\Rightarrow basic\_court\_jurisdiction(Defendant) \end{aligned} \tag{7}$$

$$r_2\colon commited\_offence(Defendant, Offence),\ max\_imprisonment(Offence, Years),\ Years > 10 \Rightarrow higher\_court\_jurisdiction(Defendant) \tag{8}$$

$$r_3\ :\ commited\_offence(Defendant, Offence),\ age(Defendant, Years),\ Years < 18 \Rightarrow higher\_court\_jurisdiction(Defendant) \tag{9}$$

Defeasible logic provides superiority relations for resolving the contradictory conclusions between rules. These relations give priority to one rule over another. For rules on court jurisdictions the superiority relation should grant priority to the rule r_3 over the rule r_1:

$$r_3 > r_1 \tag{10}$$

3.1.1 Substantial legal norms

Having the logical representation of relevant sources of law, we formed the rule base as a set of machine-readable legal norms that defines criminal offenses related to a traffic violation and drug abuse i.e. substantial law, along with legal norms defining procedural constraints related to form and content of indictments i.e. procedural law. We decided to use the OASIS LegalRuleML standard (OASIS 2021) for modeling the rule base. This standard is designed as an interchange format for legal rules. It is an open standard specified as an XML-based language that facilitates the representation of semantics and logic properties of legal norms. The LegalRuleML supports the defeasibility of legal rules and its syntax is based on the RuleML language making it easier to find a compatible reasoner.

The legal rules can be represented by the LegalRuleML language as a <PrescriptiveStatement> element based on the RuleML <Rule> element. The rule is given in

the *if–then* form consisting of logical expressions. The <if> part of the rule represents a condition that is needed for the rule to be applied and the <then> part contains a consequence that occurs for the given condition. Both, the <if> part and the <then> part are formed of <Atom> elements i.e. expressions representing predicates. Atoms use <Rel> elements to name predicates, <Var> elements to denote variables and <Ind> elements to reference an individual or a constant value.

The legal rule that qualifies a driving speed as excessive speed (Formula 1) can be represented using a <PrescriptiveStatement> element (Listing 1). Representation of the legal rule in its <if> part contains the conjunction of two atoms, one atom for the predicate on the person's driving speed, and another atom that tests if the driving speed is greater than the speed limit. The <then> part of the rule specifies a consequence as an atom built on the predicate indicating that the driver exceeds the speed limit.

```
<lrml:PrescriptiveStatement key="ps_lorts_art43b">
  <ruleml:Rule key=":lorts_art43b" closure="universal" strength="defeasible">
    <ruleml:if>
      <ruleml:And>
        <ruleml:Atom>
          <ruleml:Rel iri=":speed"/>
          <ruleml:Var>Defendant</ruleml:Var>
          <ruleml:Var>Speed</ruleml:Var>
        </ruleml:Atom>
        <ruleml:Atom>
          <ruleml:Expr>
            <ruleml:Fun>&gt;</ruleml:Fun>
            <ruleml:Var>Speed</ruleml:Var>
            <ruleml:Ind>50</ruleml:Ind>
          </ruleml:Expr>
        </ruleml:Atom>
      </ruleml:And>
    </ruleml:if>
    <ruleml:then>
      <ruleml:Atom>
        <ruleml:Rel>excessive_speed_in_populated_area</ruleml:Rel>
        <ruleml:Var>Defendant</ruleml:Var>
      </ruleml:Atom>
    </ruleml:then>
  </ruleml:Rule>
</lrml:PrescriptiveStatement>
```

Listing 1 - Legal rule on excessive speed in a populated area

The qualification of excessive driving speed that causes bodily injuries or property damage, as a criminal offence (expressed by the Formula 2), is represented by LegalRuleML as shown in Listing 2.

```
<lrml:PrescriptiveStatement key="ps_cc_art289para1">
  <ruleml:Rule key=":cc_art289para1" closure="universal" strength="defeasible">
    <ruleml:if>
      <ruleml:And>
        <ruleml:Atom>
          <ruleml:Rel iri="excessive_speed_in_populated_area"/>
          <ruleml:Var>Defendant</ruleml:Var>
        </ruleml:Atom>
        <ruleml:Or>
          <ruleml:Atom>
            <ruleml:Rel iri="caused_injuries"/>
            <ruleml:Var>Defendant</ruleml:Var>
            <ruleml:Ind>light</ruleml:Ind>
          </ruleml:Atom>
          <ruleml:And>
            <ruleml:Atom>
              <ruleml:Rel iri="caused_damage"/>
              <ruleml:Var>Defendant</ruleml:Var>
              <ruleml:Var>Damage</ruleml:Var>
            </ruleml:Atom>
            <ruleml:Atom>
              <ruleml:Expr>
                <ruleml:Fun>&amp;gt;</ruleml:Fun>
                <ruleml:Var>Damage</ruleml:Var>
                <ruleml:Ind>200000</ruleml:Ind>
              </ruleml:Expr>
            </ruleml:Atom>
          </ruleml:And>
        </ruleml:Or>
      </ruleml:And>
    </ruleml:if>
    <ruleml:then>
      <ruleml:Atom>
        <ruleml:Rel>commited_offence</ruleml:Rel>
        <ruleml:Var>Defendant</ruleml:Var>
        <ruleml:Ind>art289para1</ruleml:Ind>
      </ruleml:Atom>
    </ruleml:then>
  </ruleml:Rule>
</lrml:PrescriptiveStatement>
```

Listing 2 - Legal rule on the criminal offence committed by a traffic violation

In this rule, the severity of the injuries is denoted by the constant value ‘light’. The criminal offence is identified by the constant value ‘art289para1’ representing the article and the paragraph of the Criminal Code which defines this criminal act.

To represent penalty for a criminal offence, LegalRuleML provides <PenaltyStatement> and <ReparationStatement> elements, where former defines the sanction and the latter connects that sanction with the corresponding disposition i.e. <PrescriptiveStatement> element.

The penalty defined by Formula 3, which represents maximum imprisonment of 3 years for criminal offences denoted by the value 'art289para1', can be represented in LegalRuleML as defined in Listing 3.

```
<lrml:PenaltyStatement key="pen_cc_art289para1">
  <lrml:SuborderList>
    <lrml:Obligation>
      <ruleml:Atom>
        <ruleml:Rel iri=":max_imprisonment"/>
        <ruleml:Var>art289para1</ruleml:Var>
        <ruleml:Ind>3</ruleml:Ind>
      </ruleml:Atom>
    </lrml:Obligation>
  </lrml:SuborderList>
</lrml:PenaltyStatement>
```

Listing 3 - Definition of a sanction in the LegalRuleML

Now, the penalty given by Listing 3 can be connected to the disposition given by Listing 2, as presented in Listing 4 of Online supplementary material.

```
<lrml:ReparationStatement>
  <lrml:Reparation>
    <lrml:appliesPenalty keyref="#pen_cc_art289para1"/>
    <lrml:toPrescriptiveStatement keyref="#ps_cc_art289para3"/>
  </lrml:Reparation>
</lrml:ReparationStatement>
```

Listing 4 - Connection between the sanction and the disposition in the LegalRuleML

3.1.2 Procedural legal norms

Procedural legal norms can be similarly transformed into the LegalRuleML format as substantial legal norms. To represent court jurisdictions in the judicial hierarchy as determined by procedural law, we use <PrescriptiveStatement> elements. Thus, the jurisdiction of basic courts defined by Formula 4 can be represented as shown in Listing 5 of Online supplementary material

```
<lrml:PrescriptiveStatement key="ps_lorts_art43b">
  <ruleml:Rule key=":lorts_art43b" closure="universal" strength="defeasible">
    <ruleml:if>
      <ruleml:And>
        <ruleml:Atom>
          <ruleml:Rel iri="commited_offence"/>
          <ruleml:Var>Defendant</ruleml:Var>
          <ruleml:Var>Offence</ruleml:Var>
        </ruleml:Atom>
        <ruleml:Atom>
          <ruleml:Rel iri="max_imprisonment"/>
          <ruleml:Var>Offence</ruleml:Var>
          <ruleml:Var>Years</ruleml:Var>
        </ruleml:Atom>
        <ruleml:Atom>
          <ruleml:Expr>
            <ruleml:Fun>&lt;=</ruleml:Fun>
            <ruleml:Var>Years</ruleml:Var>
            <ruleml:Ind>10</ruleml:Ind>
          </ruleml:Expr>
        </ruleml:Atom>
      </ruleml:And>
    </ruleml:if>
    <ruleml:then>
      <ruleml:Atom>
        <ruleml:Rel>basic_court_jurisdiction</ruleml:Rel>
        <ruleml:Var>Defendant</ruleml:Var>
        <ruleml:Var>Offence</ruleml:Var>
      </ruleml:Atom>
    </ruleml:then>
  </ruleml:Rule>
</lrml:PrescriptiveStatement>
```

Listing 5 - Legal rule on jurisdiction of basic courts

Determination of jurisdiction of higher courts expressed in Formula 5 is also transformed into the LegalRuleML format, but its listing is not given separately due to the minor differences compared to Listing 5.

The jurisdiction of higher courts for criminal offenses committed by juvenile defendants (Formula 6) can be represented by LegalRuleML as shown in Listing 6 of Online supplementary material.

```
<lrml:PrescriptiveStatement key="ps_lorts_art43b">
  <ruleml:Rule key=":lorts_art43b" closure="universal" strength="defeasible">
    <ruleml:if>
      <ruleml:And>
        <ruleml:Atom>
          <ruleml:Rel iri="commited_offence"/>
          <ruleml:Var>Defendant</ruleml:Var>
          <ruleml:Var>Offence</ruleml:Var>
        </ruleml:Atom>
        <ruleml:Atom>
          <ruleml:Rel iri=":age"/>
          <ruleml:Var>Defendant</ruleml:Var>
          <ruleml:Var>Years</ruleml:Var>
        </ruleml:Atom>
        <ruleml:Atom>
          <ruleml:Expr>
            <ruleml:Fun>&amp;lt;</ruleml:Fun>
            <ruleml:Var>Years</ruleml:Var>
            <ruleml:Ind>18</ruleml:Ind>
          </ruleml:Expr>
        </ruleml:Atom>
      </ruleml:And>
    </ruleml:if>
    <ruleml:then>
      <ruleml:Atom>
        <ruleml:Rel>higher_court_jurisdiction</ruleml:Rel>
        <ruleml:Var>Defendant</ruleml:Var>
        <ruleml:Var>Offence</ruleml:Var>
      </ruleml:Atom>
    </ruleml:then>
  </ruleml:Rule>
</lrml:PrescriptiveStatement>
```

Listing 6 - Legal rule on jurisdiction of higher courts by defendant's age

A potential contradiction between the conclusions of legal rules given by Listing 5 and Listing 6 can occur in juvenile cases for crimes whose prescribed imprisonment is not more than 10 years. The LegalRuleML format supports defeasible logic and is capable of establishing superiority relations between rules. The <OverrideStatement> element can be used for setting a priority of one legal rule over another. Because juvenile criminal cases are exempt from the jurisdiction of basic courts, priority should be given to the legal rule shown in Listing 6 over the legal rule shown in Listing 5, as presented in Listing 7 of Online supplementary material.

```
<lrml:OverrideStatement>
  <lrml:Override under="#ps_cc_art246para1f" over="#ps_cc_art246apara1"/>
</lrml:OverrideStatement>
```

Listing 7 - Giving a priority to one rule over another in LegalRuleML

Similarly, we transformed other legal norms related to indictments and to selected criminal offenses into the LegalRuleML format.

3.2 Legal document assembly knowledge

To enable computer-guided legal document assembly we aim to represent the drafting skills of experienced legal professionals. We call this knowledge the document assembly knowledge and it represents a tacit knowledge gained through experience in legal writing. We seek to formally represent a procedure for creating these documents rather than simply producing the legal document. It is an important feature for educational purposes that enables tutoring the students on how they should perform the document drafting by themselves. Thus, we build the assembly knowledge by combining the method for generating the document from case facts with instructions on how to conduct the whole document assembly procedure. We intend to represent this knowledge in a machine-readable format.

Our research of assembly knowledge responsible for generating legal documents includes analysis of real-world examples of legal documents. By the courtesy of the Higher Public Prosecutor's Office in Novi Sad, we obtained a sample of indictments for our research. The sample contains 21 fully anonymized documents involving indictments charging drug and traffic offenses. After examining these documents we noticed some consistency in document style although the indictments are created by different deputy public prosecutors. All these documents follow procedural norms that regulate the structure and mandatory data in indictments while substantial norms are included in explanations of committed criminal acts.

The sample documents contain a variety of data e.g. defendant's personal data, crime description, competent court, qualification of crime, and proposed sentence. Some of these data are simple facts about the crime, while others can be derived on the basis of these facts and legal norms. For example, the qualification of crime and the proposed sentence depends on the crime description. Similarly, the court jurisdiction can be determined by the crime severity, the location where the crime is committed and also by age of the defendant. Therefore, we identified a set of base facts that is sufficient to derive the rest of the data needed for document assembly. Detailed analysis of the indictments also shows that some common phrases are used to include case data in the document.

To formally represent the assembly knowledge that models document formation procedure spanning the entire assembly process, from obtaining basic case facts to the completely generated legal document, we are separately modeling the text composing knowledge and the assembly procedure knowledge. The text composing knowledge defines how the text in the document should be put together and it can be represented by document templates. The assembly procedure knowledge defines how document assembly and interaction with the student should be conducted. The next two subsections briefly explain machine-readable representation for both parts of the assembly knowledge.

3.2.1 Document template

Before designing the document templates to represent the knowledge for composing text in the legal documents, the target format for the generated documents should be selected. There are a variety of formats for representing legal documents. We intend

to generate legal documents in an open and machine-readable document standard to allow students to analyze the anatomy of the semantic documents and potentially reuse these documents for other educational purposes. We choose the Akoma Ntoso standard (Palmirani and Vitali 2011) which enables the representation of legal documents in XML format. It is a technology-neutral standard that supports the structural and semantic markup of legal documents.

To build legal documents complying with the Akoma Ntoso schema, we need a template modeling language providing support for generating XML documents. For this purpose, we use ToXgene (Barbosa et al. 2002) which is both a document template language and a document generator. We chose ToXgene because its template language is based on the well known XML Schema standard and enables using variables and data values to control if some document part should be generated.

Our observations regarding the style of the sample documents, such as the layout of the document, used phrases, the order of presented facts, text formatting, etc. helped us to construct machine-readable document templates using the ToXgene language. These templates support loading the input data from provided XML files. The template initiates the data loading using the <tox-list> element by specifying the collection name (attribute 'name') and the name of the XML file that contains input data (attribute 'readFrom'). The structure of loaded data should be modeled according to the structure of the XML file containing the input data. When the input data consists of key-value pairs, it could be represented as elements with two child elements, one for the data name and the other for the data value (Listing 8).

```
<tox-list name="fact_list" readFrom="facts.xml">
  <element name="fact">
    <complexType>
      <element name="name" type="string"/>
      <element name="value" type="string"/>
    </complexType>
  </element>
</tox-list>
```

Listing 8 - Loading input data by ToXgene document template

When the list of case facts is constructed, these data can be embedded in the generated document. The data can also be included in logical expressions for determining if some text fragments should be displayed in the document or should be omitted.

Presenting the case facts in the document can be performed by putting their values in some of the structural elements. Because AkomaNtoso uses grammar similar to HTML for markup of paragraphs, we use <p> elements to embed the data. The part of the ToXgene document template representing some facts related to a traffic accident is shown in Listing 9.

```
<element name="p">
  <complexType>
    <tox-value>Indictment</tox-value>
  </complexType>
</element>
<element name="p">
  <complexType mixed="true">
    <tox-value>against </tox-value>
    <element name="party">
      <attribute name="id">
        <tox-expr value="'party1'"/>
      </attribute>
      <attribute name="refersTo">
        <tox-expr value="'#defendant'"/>
      </attribute>
      <complexType>
        <tox-sample path="[fact_list/fact]" where="EQ([name],'defendant')">
          <tox-expr value="[value]"/>
        </tox-sample>
      </complexType>
    </element>
  </complexType>
</element>
<element name="p">
  <complexType>
    <tox-value>because of justified suspicion that:</tox-value>
  </complexType>
</element>
<element name="p">
  <complexType>
    <tox-value>on </tox-value>
    <tox-sample path="[fact_list/fact]" where="EQ([name],'accident_date')">
      <tox-expr value="[value]"/>
    </tox-sample>
    <tox-value>caused traffic accident, driving the vehicle </tox-value>
    <tox-sample path="[fact_list/fact]" where="EQ([name],'vehicle_model')">
      <tox-expr value="[value]"/>
    </tox-sample>
    <tox-value> plates number </tox-value>
    <tox-sample path="[fact_list/fact]" where="EQ([name],'vehicle_plates')">
      <tox-expr value="[value]"/>
    </tox-sample>
    <tox-value> and caused </tox-value>
    <tox-sample path="[fact_list/fact]" where="EQ([name],'victim_injuries')">
      <tox-expr value="[value]"/>
    </tox-sample>
    <tox-value> bodily injuries.</tox-value>
  </complexType>
</element>
```

Listing 9 - Fragment of document template

In general, the ToXgene document template defines XML elements, similar to XML Schema language, and these definitions are used to generate the content of

the output document. For every paragraph in the document, we define a <p> element that has a complex type to enable child elements and mixed content. To embed constant strings in the content of a paragraph we use the ToXgene element <tox-value>. Retrieval of data from the collection of loaded case facts is performed by <tox-sample> elements, specifying the collection name (attribute 'path') and criteria that the fact name must meet (attribute 'where'). Retrieved fact is included in the document using <tox-expr> element and by specifying that the corresponding data value should be embedded in the document (attribute 'value'). Additionally, Akoma Ntoso annotation using the <party> element is applied to the defendant's name to improve the machine-readability of the document data.

3.2.2 Assembly configuration

The knowledge base represents general document assembly knowledge. It consists of the rule base and the document templates. To create an instance of the legal document the data about a particular legal case is needed. Using the rule base to reason over the case facts produces legal conclusions relevant for the document genesis. Both the case facts and the reasoning results, shape the final document on the basis of the document template.

To define relevant facts and to set up the assembly process we designed an assembly configuration. The assembly configuration determines the flow of the document assembly for a particular legal document type. It puts together the knowledge base and instructions on how the document assembly process should proceed. The assembly configuration is stored as an XML document and its structure is shown in Listing 10.

```
<Exercise name="Indictment for traffic violation" templateFilename="indictment_art297para2.tsl">
  <RuleFacts>...</RuleFacts>
  <TemplateFacts>...</TemplateFacts>
  <Steps>...</Steps>
</Exercise>
```

Listing 10 - The assembly configuration structure

The assembly configuration consists of its name, the filename of the document template, definitions of case facts relevant for reasoning using the rule base, definitions of case facts that appear in the document, and definitions of assembly steps. The filename of the rule base is not explicitly specified because a single rule base is used for all assembly configurations. We intend to facilitate the construction and maintenance of the rule base since numerous legal rules can be used in various assembly configurations, in contrast to document templates that are specially designed for the particular document type and the legal area.

The connection between assembly configuration and the legal rules both, substantial and procedural, is established by definitions of all case facts the assembly

process should obtain to draw conclusions relevant for the document assembly. These facts are defined as illustrated in Listing 11 of Online supplementary material.

```
<RuleFacts>
  <RuleFact name="defendant" type="string"/>
  <RuleFact name="speed" type="int"/>
  <RuleFact name="property_damage" type="float"/>
  <RuleFact name="caused_injuries" type="enum(minor,grievous,death)"/>
  <RuleFact name="defendant_age" type="int"/>
  …
</RuleFacts>
```

Listing 11 - Definition of facts needed for the rulebase

The case facts are defined by their names and data types. The fact names must match the corresponding variable names used in LegalRuleML statements. The fact type determines if some format conversion is needed before reasoning over the fact. Most common data types are supported including the enumerations that consist of constant values allowed as the fact's input values.

The assembly configuration also contains definitions of case facts needed by the document template. These definitions specify case facts that appear in the legal document as shown in Listing 12.

```
<TemplateFacts>
  <TemplateFact name="defendant" type="string"/>
  <TemplateFact name="speed" type="int"/>
  <TemplateFact name="property_damage" type="float"/>
  <TemplateFact name="accident_date" type="date"/>
  <TemplateFact name="vehicle_model" type="string"/>
  <TemplateFact name="jurisdiction:court" type="reasoningResult"/>
  …
</TemplateFacts>
```

Listing 12 - Definition of facts needed for the document template

The case facts needed for populating the document template are specified by their name and their value, similarly to the definitions of facts needed for legal reasoning. The type 'reasoningResult' is reserved for inferred case facts determined by the reasoning over the input data. The name of these facts consists of a relation name and variable name, separated by a colon, that should match the corresponding relation and the variable name in reasoning conclusions.

Our plan is to organize the document assembly process as a set of assembly steps which gradually build the final document. This way students can track how a single fact reflects on the generated document. Every assembly step defines interaction with the student for gathering a case fact. Therefore, an assembly step consists of a question the student should be asked, a type of data the student is expected to enter, and explanatory material for the current step. Currently, the

assembly configuration does not support loops of assembly steps for multiple entries. An example of a step definition is shown in Listing 13.

```
<Steps>
  <Step ruleFact="defendant" templateFact="defendant" answerType="string"
        text="Defendant's name" hint="e.g. John Doe">
    <StepExplanations ref="explanatory_cpc332.xml"/>
  </Step>
  …
</Steps>
```

Listing 13 - Definition of an assembly step

The assembly step is defined using <Step> element with attributes determining the fact name used for reasoning by legal rules (attribute 'ruleFact'), the fact name passed to the document template (attribute 'templateFact'), a type of data the user is expected to enter (attribute 'answerType'), the text of the question asked to the user (attribute 'text'), and the example of expected answer format (attribute 'hint'). The child element <StepExplanations> refers to a document containing explanations for the current assembly step. The configuration allows multiple explanatory documents. Explanatory materials are stored in XML files with a simple structure presented in the example in Listing 14.

```
<ExplanatoryMaterial>
  <title>Criminal Procedure Code (Article 332)</title>
  <content>Indictment contains the following: first name and surname of the defendant with personal data, brief
description of the offence, …</content>
</ExplanatoryMaterial>
```

Listing 14 - Structure of an explanatory material

The structure of explanatory material consists of title and content. The title indicates the origin of the explanation (e.g. title of textbook or legislation) and the content provides its excerpt. Storing explanatory material separately from assembly configuration enables the reuse of the materials by multiple assembly configurations.

3.3 Document assembly method

The document assembly method employs the knowledge base previously described in this section to assemble the legal documents. It is intended to support the education of law students in legal writing by guiding them through the assembly steps.

The assembly configuration is designed to enable successive addition of facts to the case description enabling building of legal documents iteratively.

The assembly method should enable data input and document assembly in cycles and can be decomposed into several processes, namely the assembly process, the reasoning process, the document generating process, and the argument graph generating process.

Our legal document assembly method consists of the steps defined by document assembly configuration. The document assembly process is performed in cycles, providing the explanations to the student, asking for the case fact, and obtaining the given answer. At every step, the legal document and the argument graph are generated using the data collected so far.

The student is expected to provide only basic facts that cannot be automatically inferred. Application of the rule base consisting of substantial and procedural legal norms to the input facts should obtain inferred facts. A reasoning process is responsible for drawing these conclusions about the case. The inferred facts are used in addition to the input facts for creating output documents. It is mandatory for the reasoning process to support defeasible reasoning.

Assembly of the legal document is performed using the case data and the document template to populate the content of the document. The document generating process performs the document assembly task supporting the complexity of the document structure and relations between its elements.

Application of the document assembly method in the education of law students should reveal the chains of reasoning about the case, exposing how the input data and the rule base lead to the conclusions. The argument graph generating process should build an argument graph on the top of these inferences by creating the arguments to connect facts and rules i.e. premises with their conclusions in deductive reasoning. Representation of the argument graph should help students to better understand how legal norms apply to the case.

An illustration of the document assembly method is shown in Fig. 2.

The processes participating in the document assembly method are depicted as the gear shapes. The rectangles with gray color represent components of the knowledge base. Other rectangles with white background represent the case-specific data, collected or produced during the assembly process.

The document assembly is performed in cycles i.e., steps enabling the student to enter an answer to the given question and providing him with explanations on the purpose of that step. Input facts are a collection of answers entered by the student. These data are used for document assembly as they represent key facts in a particular legal case. The reasoning component uses the input facts and legal rules to obtain inferred facts. Drawing conclusions by the reasoning component is based on defeasible logic. The document generator component produces legal documents from the document template, input facts and inferred facts. The generated document is displayed to the user reflecting the case description collected so far. The argument graph is constructed by the argument graph generator using legal rules to identify relations between input facts and inferred facts. The argument graph shows the student how the law applies to the case facts and what it implies. After the assembly

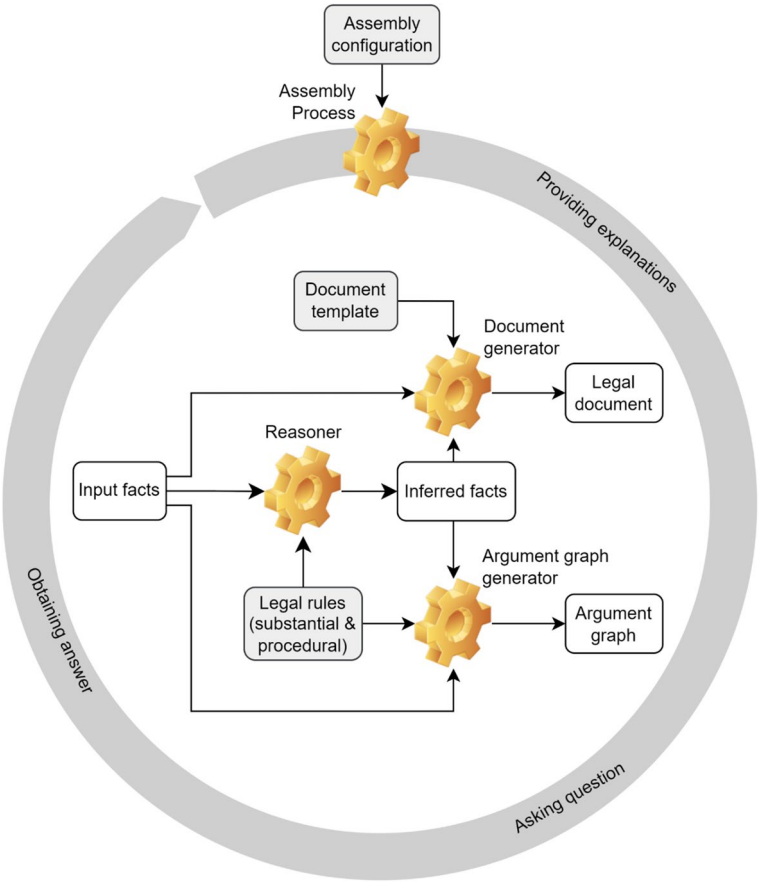

Fig. 2 The document assembly method

step is completed, the whole cycle is repeated for the next assembly step until the last step is reached.

4 Results

As a proof of concept for the method proposed in this paper, we constructed a legal document assembly system called LEDAS. The source code of the system is available at Marković and Gostojić 2022. The system is developed as a web application to support simultaneous access of multiple students (e.g. in the classroom) using various client platforms.

The system structure is illustrated using the UML deployment diagram shown in Fig. 3. The system consists of the backend and the frontend application. The

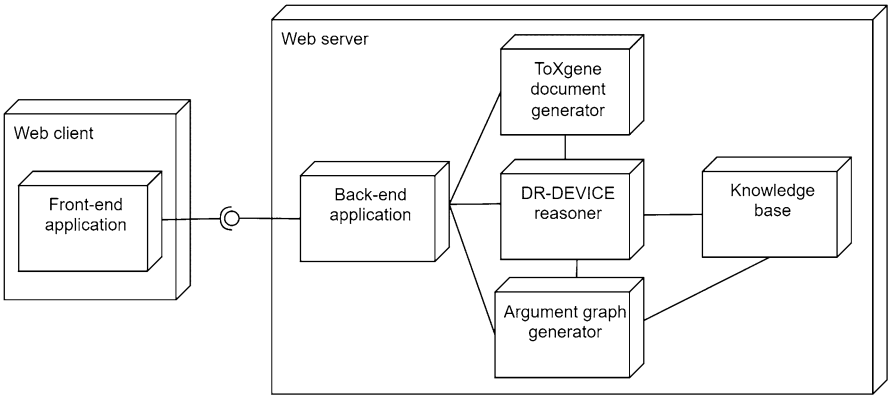

Fig. 3 UML deployment diagram

backend application is developed as a Java application that handles requests initiated by the frontend application. For communication with the frontend application, it implements RESTful web services by JAX-RS specification using the Jersey framework (Eclipse 2022). The backend application also generates and serves documents and other content that should be presented to the student. It implements the processes used by the document assembly method i.e., the assembly process, the reasoning process, and the processes for generating the legal document and the argument graph. Implementation of the reasoning process is a vital component of our assembly system. It uses the input data and the rule base to produce inferred data required for generating the legal document and the argument graph.

Defeasible reasoning over the case facts and the rule base is enabled using the DR-DEVICE engine (Bassiliades et al. 2004). We chose this reasoning engine because it uses similar formats for data representation as our method. The engine supports reasoning over rule base in DR-RuleML format and input facts in RDF format (Kontopoulos et al. 2011). DR-RuleML refers to an extension of RuleML language enabling the representation of defeasible rules. To enable reasoning over the rule base given in LegalRuleML format our application performs XSLT transformation of the rule base into the DR-RuleML format.

The major difference between LegalRuleML and DR-RuleML formats is the representation of arguments in relations. The LegalRuleML use positional representation while DR-RuleML uses slotted argument representation. Additionally, rules defined in LegalRuleML by <PrescriptiveStatement> elements use <if> and <then> elements as wrapper elements for the body and the head part of the rule, while rules in DR-RuleML format are given by <Implies> elements and consists of <body> and <head> elements.

For instance, the transformation of the rule shown in Listing 1 into the DR-RuleML syntax using results in the content given by Listing 15.

```
<Implies ruletype="defeasiblerule">
  <oid>
    <Ind uri="lorts_art43b">lorts_art43b</Ind>
  </oid>
  <body>
    <And>
      <Atom>
        <op>
          <Rel uri="dd:case"/>
        </op>
        <slot>
          <Ind uri="dd:defendant"/>
          <Var>Defendant</Var>
        </slot>
        <slot>
          <Ind uri="dd:speed"/>
          <Var>Speed</Var>
        </slot>
      </Atom>
      <Equal>
        <Expr>
          <Fun in="yes">&amp;gt;</Fun>
          <Var>Speed</Var>
          <Ind>50</Ind>
        </Expr>
      </Equal>
    </And>
  </body>
  <head>
    <Atom>
      <op>
        <Rel>excessive_speed_in_populated_area</Rel>
      </op>
      <slot>
        <Ind uri="defendant"/>
        <Var>Defendant</Var>
      </slot>
    </Atom>
  </head>
</Implies>
```

Listing 15 – The rule from Listing 1 transformed into the DR-RuleML format

The document generation process is implemented using the ToXgene engine. The engine uses the document template and the case data to generate the legal document. The case data consisting of input facts and the inferred facts needs to be exported into an XML before generating the document. The output of the ToXgene engine is Akoma Ntoso documents because in our method we designed the document templates for producing the documents of this type. The download of generated

documents in RTF and PDF format is supported using the Apache FOP library (Apache 2022).

Implementation of the argument graph generation process is performed on the basis of input facts, the rule base, and the reasoning results. As the legal rules are represented in the *if–then* form, we search through the rule base to find which rules have their *then* part confirmed by the reasoner. The arguments are constructed using atoms from the *if* part as premises and the atom from the *then* part as the conclusion. The argument graph is constructed by establishing connections between them when two or more arguments are sharing the same relations in their atoms. Representation of the argument graph is exported in JSON format.

Pseudo-code for generating the argument graph is given in Algorithm 1. It relies on proofs generated by the DR-DEVICE reasoner. The proofs are a collection of inferences obtained during the reasoning process indicating which atoms are proven, either strictly or defeasibly. This information is helpful for identifying rules whose head and body are both confirmed. We take identifiers of these rules and use them as vertices in the argument graph. Also, relation names found in atoms of these rules are used as vertices. Connections between rules and their atoms become edges in the argument graph.

Algorithm 1 : Generating argument graph

input 1: R, the rulebase
input 2: P, the proofs
output: $G(V,E)$, the argument graph
$V \leftarrow \emptyset$; $E \leftarrow \emptyset$
for all $rule \in R$ **do**
    $ruleconfirmed \leftarrow rule.head.atom.rel \in P$;
    for all $atom \in rule.body$ **do**
        if $atom.rel \notin P$ **then**
            $ruleconfirmed \leftarrow$ false;
        end if
    end for
    if $ruleconfirmed$ **then**
        $V \leftarrow V \cup \{rule.oid\} \cup \{rule.head.atom.rel\}$;
        $E \leftarrow E \cup \{(rule.oid, rule.head.atom.rel)\}$
        for all $atom \in rule.body$ **do**
            $V \leftarrow V \cup \{atom.rel\}$;
            $E \leftarrow E \cup \{(atom.rel, rule.oid)\}$
        end for
    end if
end for
Output $G(V,E)$

Additionally, for the visual representation of the argument graph, we translate rule identifiers and relation names into meaningful labels. Also, we use different shapes to indicate statement nodes (rectangle) and argument nodes (ellipse).

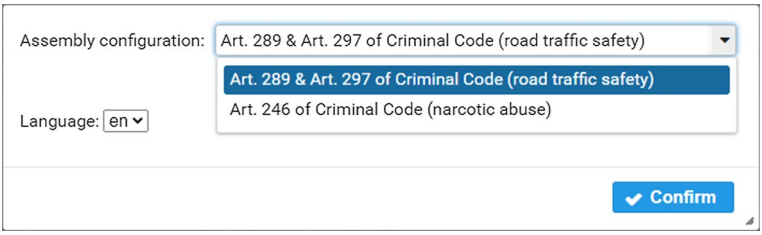

Fig. 4 The dialog for choosing assembly configuration

The assembly process is implemented by the frontend application because it controls interaction with the student. Once the frontend application gets the assembly configuration from the backend application it can run the assembly process as specified by the configuration. Sequentially, going through the assembly steps, the explanatory materials are provided to the student, and the question is being asked. The student's answer is sent to the backend application resulting in a generated legal document and argument graph. These data are presented to the student and the process moves to the next question.

The frontend application is developed using the Angular framework (Google 2022) and PrimeNG library (PrimeTek 2021) of user interface components. For a visual representation of the argument graph, we use the vis.js library (Almende 2022). Due to the complexity of the user interface it is designed for use on a desktop computer.

When the student starts using the system the dialog box is shown with the list of available assembly configurations created for particular document types and criminal offenses (Fig. 4). Currently, assembly configurations are available for indictments on traffic violations and drug abuse.

The dialog also allows the user to select the language of the user interface offering English and Serbian at this point. After choosing some of the displayed assembly configurations, the dialog box disappears showing the user interface as illustrated in Fig. 5.

The user interface consists of four panels. In the "Assembly progress" panel displayed at the top of the page, the chosen assembly configuration is specified and the progress of the assembly procedure is indicated. Every assembly step is illustrated with a button. The buttons enable the student to return to some of the prior steps and alter the entered data. Just below the button of the current step a dialog box is displayed asking the student to enter the needed information. The dialog box contains the question, input field and action buttons. The first button cancels the data input by closing the dialog, but it can be shown again if the student clicks on the button of the current assembly step. The second button reverts the document assembly to the previous step, while the third button forwards the assembly to the next step. Also, if the student just confirms the input by pressing the 'enter' key on the keyboard the document assembly advances to the next step.

The "Document" panel displayed on the left side shows the content and layout of the generated document. It is updated on every assembly step reflecting the changes

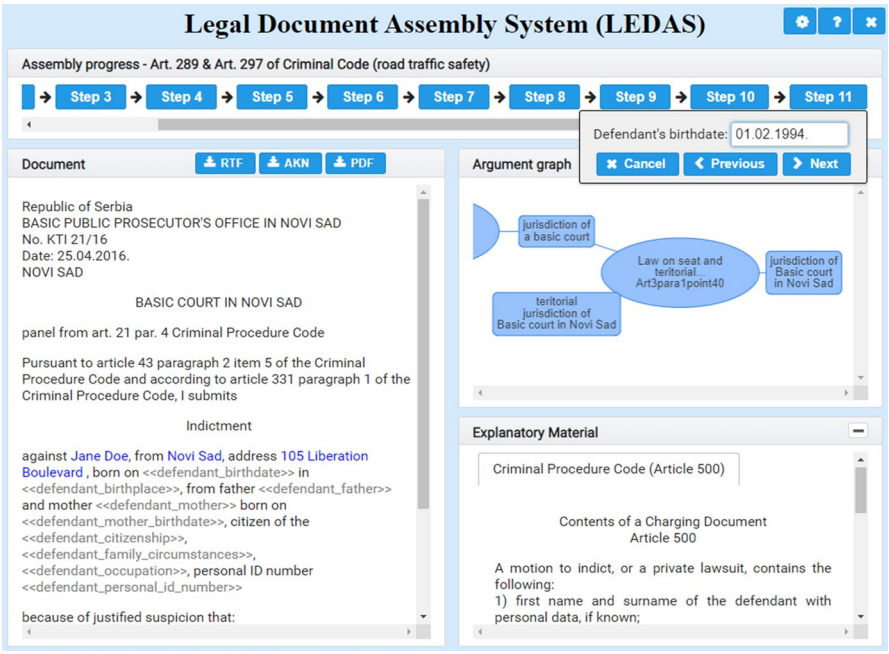

Fig. 5 The user interface of the document assembly system

caused by the student's input. The document content is read-only but the panel contains the three buttons for exporting the document in Akoma Ntoso, RTF or PDF format. The placeholders showing the name of a case fact surrounded by symbols '≪' and '≫' mark positions in the document where input data will be embedded. After the assembly procedure reaches the step asking for some of these facts and the student enters the value, it will replace the corresponding placeholder. Some of these values also become hyperlinks that, if clicked, enable the student to return to the assembly step in which the value is entered. Hyperlinks are not provided for inferred case facts because they usually depend on multiple input data.

On the right side of the application window, there are two panels. The "Argument graph" panel is the top panel showing the argument graph built on the results of reasoning over the input data. With each assembly step, the argument graph is updated showing the inferences that can be drawn using the input data and the knowledge base. The statements i.e. premises and conclusions are represented by rectangles and arguments are represented by ellipses. Arguments are connected by lines with their premises and the conclusion. Orientation of the arguments is from left to the right, meaning the premises of an argument are on its left side and the conclusion is on its right side. The rectangles that represent input facts perform as hyperlinks enabling the user to directly access the assembly step where the fact was entered.

The "Explanatory material" is the bottom panel on the right side of the page that displays explanatory material for the current assembly step. It supports multiple

documents organized in tabs. These materials help students to understand why the case fact they are asked to enter is relevant to the legal document. Using teachers' instructions, excerpts from textbooks, legal norms, etc. these materials provide assistance to the student enabling them to practice document assembly on their own.

Formation of the knowledge base for legal document assembly requires the rule base consisting of substantial and procedural legal norms, the document template and the assembly configuration. To make it easier for teachers to create a formal representation of the assembly configuration, we developed a simple tool for generating this configuration based on the rule base and the document template. The tool uses a command-line interface and works in text mode. The source code of the tool is available at Marković and Gostojić 2016.

Assuming that the user has legal norms represented in LegalRuleML format and the document template in ToXgene format, this tool first asks the user to enter the name of the assembly configuration and names of files containing the rule base and the document template. The tool then detects all input data needed by the rule base and the document template. The names of the variables found in the body of LegalRuleML prescriptive statements are extracted as case facts needed by the rule base. The names of the facts referenced by 'where' attributes in <tox-sample> elements of ToXgene template are extracted as case facts needed by the document template. These names of the facts are then listed to the user as two numbered sequences (Listing 16).

```
Rulebase facts:
0:speed, 1:defendant, 2:property_damage, 3:alcohol_level,
---
Template facts:
0:accident_date, 1:defendant_vehicle, 2:victim_injuries, 3:defendant,
---
```

Listing 16 - Lists of detected case facts

The user defines the assembly configuration, step by step, by choosing the rule base fact and the template fact by entering their numbers and typing the text of the question that should be displayed to the student when asked to enter a value for these facts (Listing 17 of Online supplementary material).

```
[step=0] rulebase fact: 1
[step=0] template fact: 3
[step=0] question: Defendant's name
```

Listing 17 - Lists of detected case facts

When all detected facts become referred by assembly steps the teacher is asked to choose a filename for the assembly configuration and the data is saved. The tool assigns string type as the default data type for all fact and step definitions.

Table 1 The survey results

Question	Average score	Median score
Q1: Getting started with the software is easy	5.00	5
Q2: The software is easy to use	5.00	5
Q3: The user interface is intuitive	4.79	5
Q4: It is always clear what am I expected to do	4.83	5
Q5: I understand how to fix potential errors	4.29	5
Q6: The software helps me to understand the content of the indictment	4.58	5
Q7: The software helps me to assemble an indictment	4.88	5
Q8: I would recommend this software to others	5.00	5
Q9: I would like to use similar software in professional work	4.83	5

Further improvements to assembly configuration in terms of reorganizing assembly steps or linking assembly steps to explanatory materials should be made manually using an XML editor.

5 Evaluation

To evaluate the assembly system we asked 24 law students to use the software and to participate in a survey. The evaluation was conducted in a computer laboratory with a computer for every student. We prepared two assembly configurations, one for indictments on traffic offenses and the second for indictments for drug offenses. We printed anonymized descriptions of two real prosecuting cases and distributed them to each student, one description of a traffic accident and one of narcotic abuse. To introduce students to the evaluation process, we first explained the purpose of the system and how the system works. Then, we demonstrated the document assembly process using the system to assemble an indictment on a traffic violation based on the description of the traffic accident. During the assembly, we gave instructions to the students on how to perform each step on the computer in front of them. When we finished document assembly, we let students examine by themselves how different case facts reflect on the generated document.

In the second part of the evaluation, we let students to assemble by themselves an indictment for drug abuse. While the students were engaged in assembling we monitored their progress, and provided support in interaction with the system.

When all students finished their work on indictments, we distributed a survey to them to evaluate their experience and satisfaction with the system. The survey aims to determine how students perceive help they can get from the system while being introduced to indictments creation. The questionnaire consists of nine Likert scale questions and four open-ended questions. Five-point Likert-scale questions measure students' first impression of ease of use and usefulness of the system. The open-ended questions allow students to enter their grade point average, give their opinion on the advantages and disadvantages of the system, and also suggest some improvements.

The analysis of the students' answers by questionnaire is summed in Table 1.

The grade point average is reported by 18 students and their overall grade point average is approximately 8.56 (ranging from 6 to 10) with a median value of 8.595.

Most of the students' answers on the advantages of the assembly system are related to time-saving i.e. increased speed of document assembly, easiness of use, helpful explanatory material, clarity of assembly procedure, automatic reasoning on court jurisdiction and its explanation by the argument graph. Disadvantages identified by the students include the small number of supported criminal offenses, inability to edit generated documents, grammatical incorrectness in sentence construction, and inadequate flexibility of case description to support the complexity of real-life cases. Suggestions the students gave us for the system improvement are: extending the set of supported crimes, involving more municipalities for determining the territorial jurisdiction, providing support for the assembly of contracts, and showing excerpts from legislation as hints on the argument graph when the user places the pointer over an argument.

During the evaluation the students were asked to write down their email addresses if they would like to receive installation of the document assembly system. 13 students out of 24 took this opportunity and applied for a free copy of the software.

6 Discussion

All the students completed the evaluation process successfully.

The highest score (5.00) is achieved for Q1, Q2, and Q8. The scores for Q1 and Q2 suggest that the students had no difficulties understanding how to use the system. The score for Q8 and the slightly lower score for Q9 (4.83) indicates that the students find the system valuable for them and their colleagues.

Question Q7, measuring how much the software is helpful, achieved a score of 4.88 suggesting that the students perceive the system as a useful tool for document assembly.

The score for Q4 (4.83) and the score for Q3 (4.79) show that the students had no major difficulties interacting with the system. Considering that law students rarely have a technical background, a higher score could be achieved by improving the user interface. During the survey, several students asked for the meaning of the term "user interface". It is possible that the Q3 is not clearly understood by students.

The lowest score on the questionnaire is given to Q5 (4.29). We assume this score reflects a minor software bug the students experienced during the evaluation when they were unable to deal with the error by themselves.

The ability of the system to improve students' understanding of the document content measured by the question Q6 achieved a score of 4.58 suggests that the features of the system that supports learning of the document drafting are helpful for the students. It is a very promising result for introducing the assembly system in the lawyers' education.

The overall grade point average suggests the students are above average academic performance. The feedback they provided is predominantly positive, also indicating

the urge for extending the knowledge base, providing new features to the user interface and improving the grammatical correctness of the generated document.

The students evaluated the system in the presence of their teaching assistant who commended the assembly system and positively assessed its potential application both as an educational tool in the teaching process and as a document assembly tool in the legal practice. This opinion is in accordance with the scores obtained from the survey and confirms the urge for introducing this kind of software in the lawyers' education. The application of the LEDAS system in education as an alternative to the traditional teaching of legal writing should require a more complex evaluation process. It should include pre-test and post-test of students' legal writing skills and the assessment of these skills should be performed by their teachers. Students should be divided into two groups, the first one taught by the LEDAS system, while the second one would be the control group having their lessons organized in the traditional way. Besides the results of such evaluation, the final decision on the effects of using the LEDAS system in education should also consider the evaluation results presented in this paper to estimate the quality of the proposed approach from both perspectives, the students' and the teachers'.

7 Conclusion

In this paper, we present a legal document assembly method for teaching legal writing to law students. The method is supported by legal knowledge represented as a machine-readable knowledge base. To demonstrate the application of this method in the education of law students, we developed LEDAS, a legal document assembly system. The system uses an interactive approach to gather case facts from students and to gradually create the legal document and the argument graph representing the influence of individual facts on claims stated in the document. The system is positively evaluated by a group of law students and their teachers.

The system enables students to exercise document assembly by themselves and demonstrates how an instance of chosen document type could be composed on the basis of facts entered by the student. Thoroughly prepared assembly configurations including explanatory material allow students to investigate numerous combinations of factual circumstances and examine the results.

The knowledge base of the proposed solution is developed on the basis of several criminal offenses. However, other legal norms could also be supported as the rule base relies on the promising LegalRuleML standard. Besides indictments considered in this research, the proposed solution could also support other types of initial documents (e.g. motions to indict, private prosecution, motion to pronounce a security measure, order to conduct an investigation, etc.). Once identified elements of the legal norm (disposition and sanction) can be relatively easy represented in LegalRuleML format as it follows the logical structure of legal rules. It makes the transformation of substantial and procedural law into machine-readable format a straightforward operation. Also, the simplicity of assembly configuration structure and similarity of document template language with XML Schema standard facilitates modeling of document assembly knowledge. The logging capabilities of the

LEDAS system, the reasoning engine, and the document generating engine minimize effort in testing and debugging the knowledge base making it easier for users to find and fix potential errors.

Extending support of the LEDAS system to other legal areas and document types requires the preparation of formal representation of the relevant document assembly knowledge. Although the complexity of the legal system brings numerous possibilities of legal areas and document types, some of their intersections might never appear in legal practice, for instance, the environment law and wills. Furthermore, substantial legal norms can be applied to several document types. For example, besides indictments, the legal norms about traffic violations are applicable to motions to indict in criminal cases and motions in misdemeanour cases. Although formal representation of legal knowledge may seem a complex and demanding task it is still feasible due to the fact that sets of legal norms and document types are finite sets. Moreover, the reusability of legal norms for multiple legal areas and document types reduces the overall effort to codify legal knowledge.

Broad use of the proposed approach in the classrooms would require simplification of the language used to model assembly knowledge (the assembly steps, the rules, and the document templates). It would enable law professors, in cooperation with knowledge engineers or even by themselves, to customize the application more easily to a particular legal domain. A solution could be the development of a domain-specific language that can be automatically translated to native knowledge formats.

According to the legal tradition, public prosecutors write whole indictments or motions to indict in a single sentence. This brings additional complexity to the assembling process. Natural language generation mechanisms should be used in the document assembly process to improve the quality of generated documents.

Further evaluation of the proposed system as an educational tool should be conducted in the classroom in some of the legal writing courses. It could be used to teach one group of law students to draft legal documents. The control group should be taught using the traditional approach. The legal drafting skills of both groups should be assessed by their teachers, before and after the legal writing course. A comparison of obtained results will reflect how the LEDAS is successful as an alternative method for teaching legal writing.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10506-022-09339-2.

References

AbacusNext (2022) Document automation software: generation & assembly. https://www.hotdocs.com/. Accessed 5 June 2022

Afterpattern (2022) Afterpattern: build a better way to practice law. https://afterpattern.com/. Accessed 5 June 2022

Allee T, Elsig M (2019) Are the contents of international treaties copied and pasted? Evidence from preferential trade agreements. Int Stud Quart 63(3):603–613

Almende B V (2022) vis.js: A dynamic, browser based visualization library. http://visjs.org/. Accessed 5 June 2022

Marković M, Gostojić S (2016) Configuration tool for the legal document assembly system. Available at: https://github.com/legal-informatics/knowledge-based-document-assembly/tree/master/src/legal/documentassembly/cli. Accessed 5 June 2022
Marković M, Gostojić S (2020) A knowledge-based document assembly method to support semantic interoperability of enterprise information systems. Enterprise Information Systems. https://doi.org/10.1080/17517575.2020.1793389
Marković M, Gostojić S (2022) The legal document assembly system. Available at: https://github.com/legal-informatics/document-assembly-webapp. Accessed 5 June 2022
Apache (2022) The Apache FOP Project. https://xmlgraphics.apache.org/fop/. Accessed 5 June 2022
Ashley KD (2009) Teaching a process model of legal argument with hypotheticals. Artif Intell Law 17(4):321–370
Barbosa D, Mendelzon A, Keenleyside J, Lyons K (2002) ToXgene: a template-based data generator for XML. In: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, pp. 616–616.
Barnett D (2006) Triage in the trenches of the legal writing course: the theory and methodology of analytical critique. Univ Toledo Law Rev 38:651–704
Bassiliades N, Antoniou G, Vlahavas I (2004) DR-DEVICE: a defeasible logic system for the Semantic Web. In: international workshop on principles and practice of semantic web reasoning (pp. 134–148). Springer, Berlin.
Branting LK, Lester J, Callaway C (1998) Automating judicial document drafting: a discourse-based approach. Artif Intell Law 6(2–4):111–149
Branting L K, Lester J C (1996) A framework for self-explaining legal documents. In Tilburg University, the Netherlands.
Branting L K, Callaway C B, Mott B W, Lester J C (1999) Integrating discourse and domain knowledge for document drafting. In: Proceedings of the 7th international conference on Artificial intelligence and law, pp 214–220.
Canick S (2014) Infusing technology skills into the law school curriculum. Cap UL Rev 42:663
Ceci M, Gangemi A (2016) An OWL ontology library representing judicial interpretations. Semantic Web 7(3):229–253
ClauseBase (2022) Contract Drafting Software. https://www.clausebase.com/. Accessed 5 June 2022
Draftomat (2022) Welcome to Draftomat. https://www.draftomat.app/. Accessed 5 June 2022
Eclipse (2022) Eclipse Jersey. https://eclipse-ee4j.github.io/jersey/. Accessed 5 June 2022
Google (2022) Angular. https://angular.io/. Accessed 5 June 2022
Henley BK (2001) Avoid malpractice traps with practice management software. Ohio Lawyer 15(6):26–30
Keene SL (2015) Are we there yet: aligning the expectations and realities of gaining competency in legal writing. Duquesne Law Rev 53(1):99–132
Kontopoulos E, Bassiliades N, Antoniou G (2011) Visualizing Semantic Web proofs of defeasible logic in the DR-DEVICE system. Knowl-Based Syst 24(3):406–419
Lauritsen M, Gordon T F (2009) Toward a general theory of document modeling. In: Proceedings of the 12th international conference on artificial intelligence and law, pp. 202–211.
Lauritsen M (1993) Knowing documents. In: Proceedings of the 4th international conference on Artificial intelligence and law, pp. 184–191.
The Law on Road Traffic Safety (2009) Official Gazette of the Republic of Serbia, no. 41/2009
Legito (2022) Document Automation—Legito. Available at: https://www.legito.com/US/en/document-automation (accessed 5 June 2022)
Mellinkoff D (1963) The language of the law. Little, Brown & Company, Boston
OASIS (2021) OASIS LegalRuleML TC. https://www.oasis-open.org/committees/legalruleml. Accessed 5 June 2022
Outlaw (2022) Draft—Outlaw. https://getoutlaw.com/platform/draft/. Accessed 5 June 2022
Palmirani M, Vitali F (2011) Akoma-Ntoso for legal documents. In Legislative XML for the semantic Web (pp. 75–100). Springer, Dordrecht.
Passera S, Haapio H, Curtotti M (2014) Making the meaning of contracts visible–Automating contract visualization. In: Transparency. Proceedings of the 17th international legal informatics symposium IRIS (pp. 443–450).
Petro N (2015) Document automation: using technology to improve your practice. Gpsolo 32(5):56–63
Pollman T (2014) The sincerest form of flattery: examples and model-based learning in the classroom. J Leg Educ 64(2):298–333
PrimeTek (2021) PrimeNG. https://www.primefaces.org/primeng/. Accessed 5 June 2022

Thomson Reuters (2022) Document automation. https://mena.thomsonreuters.com/en/products-services/legal/highq/document-automation.html. Accessed 5 June 2022
The Criminal Code (2005) Official Gazette of the Republic of Serbia, no. 85/2005
The Criminal Procedure Code (2011) Official gazette of the Republic of Serbia, no. 72/2011
The Law on Organization of Courts (2008) Official Gazette of the Republic of Serbia, no. 116/2008
Winek KM (2021) Writing like a lawyer: how law student involvement affects self-reported gains in writing skills in law school. J Leg Educ 69(2):568–598
Woodpecker (2022) Legal document automation. https://www.woodpeckerweb.com/. Accessed 5 June 2022