SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY – INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Information Systems

Facilitation of GDPR Compliance and Automation of Enforcement

Maximilian Josef Frank

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY – INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Information Systems

Facilitation of GDPR Compliance and Automation of Enforcement

Erleichterung von DSGVO-Konformität und Automation der DSGVO Durchsetzung

Author: Maximilian Josef Frank

Supervisor: Prof. Dr. Jens Grossklags

Advisor: M.Sc. Emmanuel Syrmoudis

Submission Date: 17. April 2023

I confirm that this bachelor’s thesis is my own work and I have documented all sources and materials used.

Munich,

_____

Maximilian Josef Frank

Abstract

The GDPR lacks of consistent, fast and wide enforcement. In this thesis it is elaborated whether parts of the GDPR enforcement workflow of supervising authorities can be automated or not. For example determining whether a data controller complies with GDPR or not can be automated by formalising the GDPR as a ruleset. A rule engine can then use the ruleset and compares it with machine-readable privacy practices of the data controller. Based upon the knowledge base DAPRECO, which is a formalisation of the GDPR, a SHACL ruleset is created. For this a converter from reified I/O logic in LegalRuleML to SHACL has been developed. The SHACL ruleset can then be used by supervising authorities to automate GDPR compliance checking, as well as data controllers for self-assessment. As it is very difficult to convert human-readable language into machine-readable formalisations, the thesis also focuses on preconditions, which have to be met for building an accurate ruleset. This also includes conditions and improvements for specifying and creating law (i.e. for legislatives). The methodology follows a universal approach, so that it can also be adapted to other laws.

Contents

1. Introduction....4
1.1. GDPR....4
1.2. Compliance with GDPR and Enforcement....5
1.3. Legal Expert Systems....6
1.4. Structure....7
2. Related Work....8
2.1. Approaches to Compliance Checking....8
2.2. Machine-Readability of Rulesets....8
2.3. Machine-Readability of Privacy Practices....10
2.3.1. Source of Privacy Practices....10
3. Scope of Automation....12
3.1. System Requirements....14
4. DAPRECO Knowledge Base....16
4.1. Container Structure....16
4.2. Rule Structure....17
5. Conversion to SHACL....20
5.1. Rule Design....20
5.1.1. Evaluation Order....22
5.1.2. Quantifiers....22
5.2. Rule Conversion....23
5.2.1. Constitutive Rules....23
5.2.2. Regulative Rules....24
5.3. Translation to SPARQL Patterns....24
5.3.1. Predicate names....24
5.3.2. Triples....25
5.3.3. Functions....25
5.3.4. Optionals....26
5.3.5. Multidirectionality of Predicates....27
5.4. Solving Problems of DAPRECO....28
5.4.1. Undefined Ontologies....28
5.4.2. Inconsistent Predicate Types....29
5.5. Conditions for Use....30
6. Discussion....31
6.1. Suitability of DAPRECO....31
6.2. Legal Preconditions and Improvements....31
6.3. Practical Relevance....33
7. Conclusion....34
Appendix A. Extraction Code Snippets....41
Appendix B. Predicate Typification....42

1. Introduction

1.1. GDPR

In May 2018 the General Data Protection Regulation (GDPR) came into effect [1]. This is a regulation to strengthen the rights of users (i.e. data subjects), who entrust companies (i.e. data controllers) with their personal data. Rights granted by this regulation are for example the right to data erasure, portability or the restriction of processing (s. GDPR Chapter 3 [1]). They give data subjects control over their personal data, even though the data have been shared with or given away to data controllers. Data controllers are by definition in GDPR Art. 4 ("Definitions") [1] those bodies (often companies), which determine the purposes and means of processing the personal data of a person (i.e. the data subject). The GDPR is valid whenever personal data are processed inside the EU or the data subject is a EU citizen – disregarding where the personal data are actually processed. Furthermore data controllers are required to ensure the privacy rights across sub-service providers. This imposes the need for awareness at cloud or infrastructure sub-service providers, which may not be used to the GDPR due to e.g. being in a different jurisdiction. Exactly this compliance of sub-service providers is one large problem for European companies, as other jurisdictions may be not compatible with the GDPR (s. reasoning of the Schrems II ruling [2] by the European Court of Justice).

Even though there has been a privacy law in Germany before, which gives a similar level of privacy (s. previous versions of BDSG [3]), the main advantage of the GDPR was to include all EU member states into having the same privacy standard, which German citizens were used to. Some conditions given with the GDPR are also reason for the export of ethical values (i.e. privacy standards) to jurisdictions without the GDPR. This is due to the facts that (a) the GDPR is also valid for data processing of EU citizens, even if the data controllers are not under EU jurisdiction [1], (b) the EU is economically stronger than Germany alone, and (c) both facts cannot be ignored in a multinational environment like the internet. Because of the economical importance of the EU, data controllers outside of the EU won't likely ignore the EU market or limit their services to non-EU users – for example we can still use many US services from the big five (i.e. Google, Amazon, Meta being the former Facebook, Apple, Microsoft) as a EU citizen. So many non-EU controllers will have to adapt and comply with the GDPR as well – at least in theory (s. Section 1.2).

One additional obligation for data controllers is the creation and maintenance of a records of processing activities (s. GDPR Art. 30 [1]). These records contain all privacy relevant information of any data processing of the data controller and its sub-service providers. The main purpose of these records is the documentation for company internal reasons, as well as for controls by supervising authorities. In contrast to those records, privacy policies are only transparency information for the data subject, and thus may not contain internal information or business-secrets about processes (s. GDPR Art. 13 and 14 [1]). Therefore the records of processing activities instead of privacy policies should be taken into account for assessing the privacy practices of a data controller.

Besides obligations and permissions imposed on data controllers, as well as rights granted to data subjects, the GDPR further regulates the duties and capabilities of public authorities,

mainly supervising authorities. One of these duties is to process complaints against unlawful data processing, as well as the actual GDPR enforcement (s. GDPR Art. 57 "Tasks" [1]). The workflow of processing a complaint (or other indications to non-compliance) includes checking the compliance of a company, as seen in Section 3.

1.2. Compliance with GDPR and Enforcement

Because of the main change in jurisdiction by the GDPR for data controllers both inside and outside the EU and their need to adaption, not only data controllers but also supervising authorities were overwhelmed [4][5][6] – despite the legal adoption period of two years, being stated before in the GDPR [1]. As the main punishing instrument of the GDPR are fines (s. GDPR Art. 58 "Powers" [1]), the lack of enforcement led to a classification of the GDPR as a calculated risk in some companies – not an obligation for compliance. For example Google ignored two consecutive rulings by the European Court of Justice (CJEU) regarding EU-US data transfer [7].

After five years of GDPR there are still many companies not complying with the GDPR. In an online survey among EU and US companies by the International Association of Privacy Professionals (IAPP) only 7% of all EU companies were fully compliant [8]. You can find a list of imposed GPDR fines in detail on [9]. As almost all of these non-compliant companies target for profit (in contrast to companies of a purpose economy), there may be only two options to force companies in complying with the GDPR: (a) improving customers' awareness, in hope that they will use only GDPR compliant companies, and thus forcing profit-oriented companies to improve their GDPR compliance, or (b) to enforce GDPR by law in an efficient manner (meaning law enforcement becomes scalable).

Unfortunately it is very inefficient to ensure privacy awareness with every citizen. This is because privacy awareness needs both motivation to act idealistic in this topic, and some basic understanding of human rights concepts. Edward Snowden [10] gave the following example to this correlation: The right to free speech would not be rejected just because someone has nothing to say. But this also means that the right to privacy must not be rejected just because someone has nothing to hide. Especially the motivation of an average person cannot be guaranteed, when it comes to peer pressure e.g. by vendor lock-ins. So the first option by counting on user behaviour seems very unlikely.

As you can see from recent trials or complaints [8][9], but also from uncertainties in law due to different judgements of supervising authorities [4], as well as the statements on the European Data Protection Supervisor Conference 2022 [4], the GDPR doesn't enjoy a consistent and broad enforcement – mainly due to missing resources at the supervising authorities [4]. This may also be due to the fact, that enforcement models are not yet fully developed. Furthermore filing a complaint to the supervisory authority often takes months to get a response and years to get a judgement [5]. This also may be due to the large number of cases being filed. From May 2018 to February 2019 there have been 206,326 cases in the EU, of which only 52% have been closed during that period [6].

For efficient enforcement there could be a system which automatically analyses compliance with data protection law, gives a result whether a data controller complies with law, and a reasoning for this result. By this the work of supervising authorities could be accelerated rapidly by using a scalable system as an aid. They could even use trade registers to automatically

monitor all companies falling under their jurisdiction, and thus achieve a 100% enforcement rate. To build such a system, I imposed the first research question, asking for which parts of a GPDR enforcement workflow can be automated.

On the other hand such a system also has advantages for companies (i.e. data controllers), because they can use that system for a self-check. If the system (or its rules) is approved by the supervising authority, the vacuum of legal knowledge and different interpretations of law in the field of privacy [4] could be filled with reliability and certainty. This would facilitate compliance with privacy laws significantly. As this also means that there is an immediate advantage of such a system, I included the next step into this thesis: How to implement such a compliance check or enforcement system?

1.3. Legal Expert Systems

Legal expert systems are systems in a specific legal domain (e.g. the GDPR), which support in legal argumentation. Legal argumentation is especially needed in trials – or transferred to the domain of the GDPR – to determine whether a data controller is compliant or not. A legal expert system is thus often designed to support (or even replace) a lawyer to the most extent possible [11].

Legal expert systems are based on knowledge bases. The latter is often a large set of rules or other statutes and decided cases (i.e. case law) [11]. Legal expert systems expect a set of facts as input [11], which are often either queries on the knowledge base or scenarios, for which legal reasonings have to be found.

This is very similar to a rule engine, which infers new facts from a set of rules and previously given facts. As proposed in [12], rule engines in the legal domain should first infer new circumstances, which are needed for compliance check, from given facts, and then infer facts of validation i.e. if certain obligations or permissions have been violated. Those obligations and permissions are also formalised as rules.

Nevertheless there are multiple approaches to legal expert systems. Some are based on neural networks and yield fuzzy results, some are case-based and take previous judgements into account, and others are strictly based on rules [11][13]. The latter ones are capable of yielding exact decisions if the input (i.e. rules and facts) is deterministic.

For legal expert systems to work, their ruleset (i.e. knowledge base) has to be a formalised version of law. This is difficult, as the legislative sometimes uses open definitions and interpretability on purpose e.g. to allow for developments in a certain field. The problem of non-interpretability imposes the third research question, asking for which preconditions have to be met for automated compliance checking or enforcement.

To convert law into machine-readable legal rules, several semi-formalising languages have evolved, starting with Akoma Ntoso [14]. Akoma Ntoso is a first way of adding machine-readable tags to natural language legal text. An Akoma Ntoso document can be seen as the rich text version of a legal document i.e. enables the use of links, references, and others.

A more fine-grained way of formalising rules is RuleML, or its for legal domains specialised counterpart LegalRuleML [15]. In contrast to Akoma Ntoso, LegalRuleML enables a complete formalisation of rules. For example the statement "If someone is guilty, then the person has to be punished." can be logically converted into "If x is a person and x is guilty, x has to be punished." and ultimately represented in `(typeof x == person) ∧ (x.guilty ==`

`true) → punish(x)`. The last representation is called first-order-logic [16], because it formalises natural language into the first (lowest) level of logic. LegalRuleML enables the encoding of such logic levels [16]. This is important to build a knowledge base directly from the law itself, which can then be used for a legal expert system.

1.4. Structure

Summarising from above, the arising research questions are …

1. Which parts of GDPR enforcement can be automated?
2. How to implement the enforcement system?
3. Which legal preconditions have to be met for efficient automated enforcement?

Research question 1 is to be answered in Section 3. The whole process of enforcement until the possible final trial in front of court is first formalised, and then visualised as a BPMN model. From this model all parts of the enforcement workflow, which can be automated, are determined.

For research question 2 regarding the implementation of such automation, the container structure and rule structure of DAPRECO is examined. DAPRECO serves as the main knowledge base for the GDPR, so that the GDPR has not to be translated from human-readable to machine-readable again. Later on in Section 5 a converter is being written, which transforms the GDPR from the DAPRECO format into SHACL. This enables a platform independent and standardised use of the GDPR rules with SHACL-SPARQL validators. The converter and its output SHACL document is one of the main deliverables of this thesis, because for automatic assessment you only need the rule engine (i.e. any SHACL-SPARQL rule engine), the privacy practices (which depend on each company and thus cannot be determined in advance), and the GDPR ruleset (i.e. the second deliverable of this thesis). The whole conversion logic is based on the specifications for reified I/O logic [17], LegalRuleML [15] and SHACL [18] – and not only the DAPRECO implementations. So the converter can also be used for different laws formalised as reified I/O logic in LegalRuleML, and is not limited to DAPRECO.

Research question 3 is then answered in Section 6, containing legal or other preconditions, which enable a complete automation of enforcement. This is based on the findings and the problems being experienced before, because building the converter to SHACL also deals a lot with machine-readable formalisation of law.

2. Related Work

2.1. Approaches to Compliance Checking

As being elaborated later in Section 3 the main work of automation of enforcement is with checking a data controller for compliance.

One approach for checking the compliance of business processes is the use of event-based compliance language, which assesses possibilities of execution (i.e. possible events in a process). This is used by bpCMon [19] to enable compliance rules from multiple perspectives. This approach is most suited to business processes, but a large part of the GDPR not only adheres to companies as data controllers, but also contains regulative norms for public authorities.

At the Center of Identity, University of Texas at Austin, the research group for PrivacyCheck [20] created a browser plugin tool, which can be run on a privacy policy page and returns basic information about the contents of the privacy policy. The tool uses natural language processing to answer predefined questions on its users' privacy protection. Even though many questions are based on principles from the GDPR, it is rather for use by data subjects than for compliance checking a data controller – not only because privacy policies are not as rich in information as records of processing activities.

icomplâi [21] uses artificial intelligence without a black-box phenomenon to check privacy policies, business processes and other documents for legal compliance with the GDPR. As the project is in its early-access phase it is not publicly available yet. It is created at the same chair, where LegAi Editor [22] was built – a knowledge base generator tool for laws and legal texts.

Other approaches are based on a concept called rule engines: a set of rules (i.e. GDPR) and a set of facts to test against (i.e. privacy practices) yield a set of results (i.e. compliance violations). Ultimately this approach is used, because it fulfils the criteria in Section 3.1 for automation of enforcement. To enable the use of the rule engine concept, both privacy practices as well as rules have to be in a machine-readable format. The following sections deal with the machine-readability of both the facts (i.e. privacy practices) as well as the rules (i.e. GDPR laws).

2.2. Machine-Readability of Rulesets

There are already multiple approaches for machine-readable law. Akoma Ntoso [14] for example is an XML format, from which you can auto-generate natural language text for law books. The approach of first creating machine-readable law and then convert it to natural language would be optimal regarding non-interpretability. Unfortunately Akoma Ntoso is designed for the other way round (i.e. human to machine language) and allows for interpretable expressions, and most laws don't exist in Akoma Ntoso. So the reverse method has to be used for now: Creating machine-readable and non-interpretable representations of law being in natural language.

Another format is LegalRuleML, a specialised RuleML format based on XML [15]. One disadvantage of LegalRuleML is that by default it doesn't contain any semantics, which makes

it unsuitable for direct evaluation [23]. Semantics are important, because different parties can use different terms for the same meaning (e.g. "data controller" and "company"). Because of the lack for direct evaluation, LegalRuleML is often converted into a format called modal defeasible logic [23]. There are some tools to process modal defeasible logic, e.g. SPINdle [24]. Unfortunately both the SPINdle engine and a SPINdle plugin transforming a subset of LegalRuleML (s. Section 5.3 of [23]) to modal defeasible logic [23] don't seem to be available anymore[1].

The W3C[2] proposed a technology stack for semantic data called "Semantic Web" [25]. Semantic Web supports creating, storing, linking and handling data, and includes the formalisation of rules or constraints for validation. Part of the Semantic Web stack is RDF (a representation format for facts i.e. privacy practices as a directed graph) [26], SHACL (the correlated constraints language for defining rules) [18], OWL (a specification for defining ontologies i.e. semantics) [27] and SPARQL (an RDF query language) [28]. An ontology unifies meanings or semantics and sets them into correlation. One example for a correlation may be, that a data controller cannot be a data subject.

There are several promising approaches to implement legal rules (see [29] and [30]), but many of them formalise rules as RDF/OWL representations instead of the dedicated SHACL (s. Section 3.1 of [32]) – even though SHACL is the W3C standard for constraints or rules [18]. Some of aforementioned tools don't even work for conversion to RDF[3]. RDF is intended for graph formalisation to describe predicates (`rdf:Property`) from a subject (`rdfs:Resource`) to an object (`rdfs:Resource`) – like a directed graph consists of relations from origin nodes to targets.

There is already a representation of the GDPR in LegalRuleML [15]: The DAPRECO knowledge base [31]. DAPRECO has the approach of first translating law into machine-readable XML, then adding semantics to it (i.e. which variables mean what). Interpretability of law is tackled by translating all possible interpretations into separate non-interpretable rules (i.e. variations or alternatives), and if necessary introduce decision variables called like `assumption` e_a (Section 3.3 of [31] on p. 5693) or exceptions.

Even though there is an experimental, incomplete RuleML to RDF converter [33], it again lacks the conversion to SHACL constraints. Conversion could be facilitated by using first-order logic and slotted argument logic directly, as proposed in [16] – which are both special types of logic languages. A very promising approach for transferring rules between rule systems is RIF in RDF [34]. This would bypass the need for a converter between different rule formats. To the author's best knowledge this unfortunately doesn't exist for the DAPRECO knowledge base yet.

To use native standards with their intended purpose, SHACL is used as target rule language. There is already an ontology which allows the use of RuleML rules in SHACL: SWRL [35].

---

1 Tracing via web.archive.org: SPINdle original webpage http://spin.nicta.org.au/spindleOnline (deadlink) redirecting to http://spin.nicta.org.au/spindle, GitHub fork of SPINdle-Editor (no evaluation logic) https://github.com/gb96/SPINdle-Editor from original author Brian Lam (github.com/oleklam), inofficial release of source code on https://sourceforge.net/projects/spindlereasoner/files/2.2.4/

2 The World Wide Web Consortium (W3C) is a leading consortium for developing web standards.

3 Executing the `triplifyMerger-ids.xsl` transformation file given from [15] on the DAPRECO XML document [31] returned compilation errors. Execution was commanded by `xsltproc -o dapreco-rdf.xml triplifyMerger-ids.xsl rioKB_GDPR.xml`

Unfortunately the mapping of RuleML to SHACL in SWRL is also not complete yet (s. Section 5 of [35]). So for automated conversion a custom method has to be found.

For the conversion of reified I/O logic (as DAPRECO uses) to SHACL, [32] is used for a basic understanding of both input and output languages. Even though [32] shows the formalisation of SHACL rules based on DAPRECO, it is incomplete yet, and also uses a custom ontology (i.e. uncommon semantics). SHACL also has implementations for rule engines in different languages, including Java (s. [32]) and JavaScript (s. [36]) to name a few. So cross-platform compatibility is already given.

2.3. Machine-Readability of Privacy Practices

There are already approaches for analysing process compliance with BPMN models (or their privacy-specific extension PE-BPMN [38]). This notation allows for many information and corresponding features. PE-BPMN is focused on privacy enhancing technologies (e.g. security guarantees as required by GDPR Art. 32 "Security of processing" [1]), but doesn't include important BPMN stereotypes about e.g. used legal bases by default. There has already been an XSLT document transforming LegalRuleML into Process Compliance Logic (PCL) to check business process models in BPMN for compliance with the rules stated in the LegalRuleML document (s. Section 4.1 p. 755 [39]) with the REGOROUS [37] business process validator. Unfortunately neither that XSLT document nor REGOROUS seems to be available anymore[4].

During research on querying business process models, multiple BPMN query languages have evolved over the past years including APQL [40], VMQL [41], BP-QL [42], WS-BPEL with temporal logic [43], BPMN-Q [44] and corresponding frameworks (as in [45]) and enhancements (as in [46]). Unfortunately those query languages only have either limited functionality, don't provide evaluation tools/frameworks, provide only partial compliance assessment by rule patterns (as in [47]) and anti-patterns (as in [48]) or are in a completely different structure, compared to reified I/O logic, which DAPRECO [31] is using. Thus the focus will be on the supported structure of the rule engine (i.e. SHACL), so that a transpiler has to be created if necessary (e.g. from BPMN to RDF [26] or SHACL).

2.3.1. Source of Privacy Practices

A privacy policy may not contain every data process or privacy practice. So if natural language processing of privacy policies would be used, there may be problems to perfect accuracy (i.e. accuracy of 100%) but also relevance of some paragraphs in the privacy policy [49][50]. For example the rights of data subjects are to be told in every privacy policy (e.g. instead of just referring to the respective paragraphs in the GDPR), but don't tell anything about the data controller's privacy practices. These rights are also granted though, if they would not be stated in a privacy policy – so the policy is blown up. It is also required by law that every person understands a privacy policy (s. GDPR Art. 5(1) lit. a "Transparency" [1]). Some companies therefore include introductory paragraphs or copy and paste GDPR Art. 4 (Definitions) [1], which reduces readability by adding very long text-blocks. This may be contrary to the original sense of that obligation: transparency by easy reading.

4 The CSIRO Data61 Regorous Archive Page doesn't contain any download links or other references to Regorous [37]. All email addresses of the authors of the XSLT document, partially also participating in Regorous, [39] seemed to be inactive – even those of correspondence authors.

Because of both cluttering with algorithmic irrelevant information and also missing information, it is not assumed to generate the machine-readable privacy practices from a natural language text, but rather directly receiving them by the data controller itself. It is out of scope of this thesis to control whether companies' privacy statements also reflect reality, or to formalise machine-readable privacy practices from a privacy policy or other natural language text.

3. Scope of Automation

In order to build a system for automating the enforcement workflow of a supervising authority, you have to take a look at the existing workflow in the supervising authority itself. It has to be determined which part of the current manual workflow can be automated. Based on correspondence with the DSK president Marit Hansen, the officer of the data protection authority in Schleswig-Holstein, Germany [51], I formalised the following workflow:

1. Entry condition: The supervising authority acts only to achieve tasks specified in GDPR Art. 57 (Tasks) [1] with the means specified in GDPR Art. 58 (Powers) [1]. As Art. 57 covers much more tasks than the target of this workflow, you can narrow the entry condition down to the assessment of a data controller. In general the potential violation, which has to be assessed, is given by a complaint (regardless whether from inside i.e. a check by the supervising authority or outside). For sake of completeness the proposed system in this thesis takes all regulations of the GDPR into consideration (including regulations for public authorities or institutions).
2. The supervising authority conducts a risk assessment, determining the severity of a potential data breach. Depending on the powers specified in GDPR Art. 58 [1] and the risk assessment, the supervising authority allows for different deadlines and post-controls.
    - There may be multiple or no phases of self-correction or rectification. If the violation was already done, there would be only a warning caution or fine and warning or hint for the future. It has to be differentiated between one-time violations and ongoing violations. For example a business process leading to violations is not compliant, so this would be an ongoing violation even though there is no data breach or similar.
    - In practice compulsory fines ("Zwangsgelder") are used to enforce claims of the supervising authority. They are issued until illegal practices are turned off.
3. The controller receiving a fine according to GDPR Art. 83 (General conditions for imposing administrative fines) [1] gets the possibility to file an appeal. This option was used only in 1% of all cases in the EU between May 2018 and February 2019 [6]. In the stage of an appeal there are often self-corrections of processes. Countermeasures given in the appeal are taken into account, whether a fine is issued and how high it will be.
4. After that a notice (administrative act) is sent to the data controller, containing another possibility to appeal and information where to object, as well as a deadline for objection (often a month).
5. On objection a court requires a statement from the supervising authority. This is then included into an oral trial, yielding a judgement. The data controller has again the possibility to file an appeal against the judgement. This repeats through court instances until there is a legally binding judgement. After this the actual enforcement is done, e.g. enforcing a fine.

As both the assessments (see 2) as well as the complaints (see 3-5) are to be evaluated manually, the task of finding violations (see 1 and green-fenced area in Fig. 1) can be automated, if necessary information about a controller's privacy practices, law and its legal interpretations are given. This is similar (but not the same) to the automated lodging of complaints against deceptive cookie banners by noyb [52] (acronym for "none of your business"), which is a consumer protection organisation for privacy. The difference is that noyb narrowed its scope to cookie banners, but on the other hand took real practices into account, instead of information given by companies. The second difference to noyb is in practicability: Legally effective automated decision making is only allowed in accordance to GDPR Art. 22(2) (Automated individual decision-making, including profiling) [1][53].

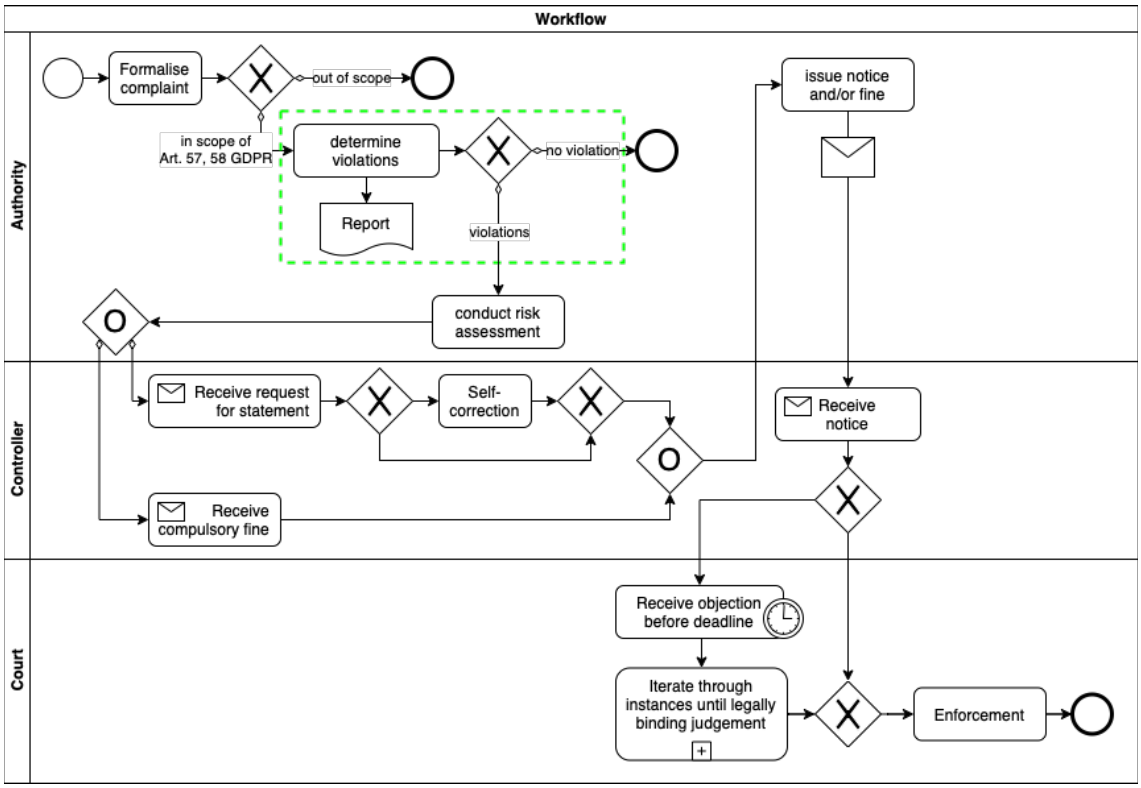

Fig. 1: BPMN model of enforcement workflow

Conformance checking and enforcement takes time, so there might be some arbitrage depending on the order in which different data controllers are assessed. Controllers being assessed at a later time might have more time available until assessment and also can learn from previously assessed controllers. Thus when going through a trade register this has to be taken into account. Therefore countermeasures have to be taken e.g. (a) assessing all controllers in random order, which draws implications when adding companies to the trade register after determining a random order, or (b) assessing companies in ascending order of registry date. Assessing companies in a different order (e.g. in descending order of registry date) might draw implications about ethical reasoning. Thus the order of assessment is left up to the supervising authorities as a one-time setup.

3.1. System Requirements

Most legal expert systems are based on finding arguments for or against a possible judgement. A legal expert system as part of this thesis should be subject to requirements, which ensure consistent judgement according to principles of a constitutional state. To determine requirements for such a system, I took a look at EU guidelines on automated individual decision-making [53].

- *Transparency* [53]: Achieved by publishing the decision rules, or at least how those rules are generated.
- *Accuracy* [53]: A judgement should be without errors, so the result of the systems must be perfect in theoretical sense. Accuracy also includes the absence of any bias. Both criteria exclude holistic approaches like fuzzy-based expert systems, or neural networks.
- *Determination* (inferred from "fairness" [53]): Perfect accuracy means that the system has to yield the same judgement for the same input. If this would not be the case, the system would have to re-decide a previous judgement with the same input, meaning the previous judgement has not been perfect. This assumes that no input can yield more than one perfect judgement at the same time. All possible outputs (i.e. compliant, unknown or non-compliant) are each excluding all other possible outputs, thus pairwise not being equal in perfectness for a specific case. That means that each of them are mutually exclusive, which makes the assumption of perfectness hold.
- *Adaptability* (inferred from "lawful" [53]): The system must be able to reflect changes in law. Thus the decision-tree has to be based on a modifiable privacy ruleset. Case-based legal expert systems are based on previous judgements, and then compare these judgements with the case to be assessed. This makes case-based systems difficult to update for new laws – especially new law contradicting previous case law.

There are multiple different architectures for legal expert systems: rule-based or deontic-logic-based (i.e. judging according to formalised rules of law), case-based (i.e. judging similar to previous cases), neural networks and fuzzy algorithms (i.e. logic being able to deal with impreciseness or missing information/declarations) [13]. The requirement for determination excludes both neural networks and fuzzy algorithms by nature (s. above).

The case-based approach introduces multiple problems. Some of them are, whether the case to be assessed is comparable to other cases [54], and secondly case-based systems interpret new laws similar to previous cases [54]. Maintaining reliability of such a system would be very hard. Therefore case-based systems are rather used for finding arguments for each party [55], than giving absolute judgements. As the system described in this thesis should be capable of the latter (i.e. giving absolute judgements), the rule-based approach is used. This also enables the transfer of liability for legal interpretations away from the system to the creators of rulesets (e.g. legislatives). The downside of rule-based approaches is their precise logical structure, whereas some laws are formulated with moving boundaries on purpose [11] (e.g. GDPR Art. 6(1) lit. f "Legitimate Interest" [1]). These uncertainties in formulation must be either mapped by case law, which is hardcoded manually as rules, or annotated as "in-assessable". To maintain

flexibility, the system should behave non-monotonic, i.e. laws evaluated at a later point in time can alter the overall result of the system [13].

There may be multiple laws in a single location to comply to. For example in Bavaria there are the law books of GDPR, BDSG (Germany), BayDSG (Bavaria), TTDSG (Germany), TMG (Germany), parts of the KunstUrhG ("Recht am eigenen Bild", a personality right), as well as comments by supervising authorities. Because of this, each law book is mapped to its own ruleset, mapping legislative rules to modules instead of a single heavy-weight whole-containing package. The system proposed in this thesis focuses on the GDPR only.

4. DAPRECO Knowledge Base

The DAPRECO knowledge base uses PrOnto [56] for semantics, which is a legal ontology not only for the GDPR (p. 139 of [56]). PrOnto distinguishes between quality (e.g. deletion, destroying and anonymising) as well as using purpose classes (fig. 7 of [56]), which gives baselines for removing interpretability. For example GDPR Art. 6(1) lit. f (Legitimate Interest) [1] depends on a weighting of data controller's interests versus data subject’s interests. To determine if an interest is legitimate in terms of law, prOnto narrows legitimate interest down to purpose classes, which are then weighted, and thus minimise interpretability. This also goes for other semantics, mainly based on case law [56], which further includes the elimination of some legal uncertainties. Some remaining uncertainties in prOnto have been replaced by custom DAPRECO ontologies (as stated in Section 3 of [31]), e.g. replacing lawfulness by norms of GDPR Art. 6(1) (Lawfulness of processing) [1].

DAPRECO uses reified input/output logic [17], which makes use of first-order logic and deontic logic.

With reified I/O logic, complex boolean expressions are mapped to flat structures, i.e. variables connected by conjunctions (i.e. logical "and"s) [17]. By using first-order logic each statement contains variables, which represent correlations to other statements or variables. For example "Companies must be compliant" becomes "For all companies x, company x is compliant" – where "x" builds the correlation between the universal quantifier "all companies" and the fact of being compliant.

Deontic logic follows a typification of rules into the three categories prohibitions, permissions and obligations. Together with defeasible logic, which means the possibility of overriding other rules, this facilitates to find a correct order for assessing facts (i.e. privacy practices) by rules. DAPRECO first evaluates prohibitions and then permissions and obligations. Reified I/O logic is a way how to process deontic rules [31]. At the beginning there is a set of facts, which is given into the first subset of rules (i.e. constitutive rules) to infer new facts. The rules are formalised in an if-then structure having facts as input and generating an output, which is then used as input again for the next subset of rules (i.e. regulative rules, mostly permissions and obligations).

4.1. Container Structure

The hierarchy used in DAPRECO can be displayed as following:

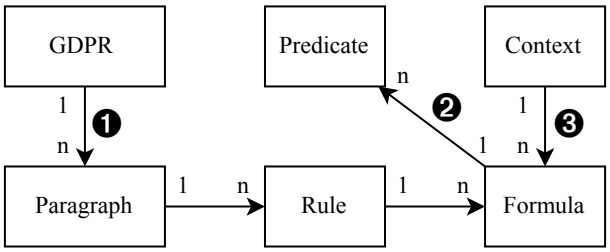

Fig. 2: Hierarchy of Legal Rules

Most legal paragraphs can not be represented by just a single rule, so a one to many relationship is used there.

When having a look into the DAPRECO knowledge base XML file, you can see multiple sections:

- LegalRuleML namespace prefixes to distinguish the ontologies `GDPR` (for legal text references), `rioOnto` (for reified I/O operators), `dapreco` (for DAPRECO ontologies) and `prOnto` (for PrOnto ontologies). This is not to be confused with XML namespaces (e.g. `<lrml:FOOBAR>`).
- Legal References (`<lrml:LegalReferences>`, s. Fig. 2 annotation 1): This contains a mapping of a GDPR legal paragraph identifier (attribute `refID`) to rule / statement identifiers (attribute `refersTo`). The identifier is in Akoma Ntoso Naming Convention format [14].
- Associations (`<lrml:Association>`, s. Fig. 2 annotation 2) occur multiple times. They contain a mapping from above's legal references to multiple rules (`<lrml:Statements>`). There is also one association assigning the modality `rioOnto:reification` to atom identifiers (s. below), indicating that these atoms are unprimed (i.e. reified) predicates in reified I/O logic [17], i.e. their predicates names refer to the existence of a fact, not the fact itself. This will be explained further in Section 5.3.1.
- Contexts (`<lrml:Context>`) occur multiple times. They assign one of the contexts "obligation", "permission", "constitutive rule" and its specialisation "constitutive rule for mapping with PrOnto" to formulas (s. Fig. 2 annotation 3).
- Rules (`<lrml:Statements>`) occur multiple times. They contain multiple …
  - Formulas (`<lrml:ConstitutiveStatement>`). A formula contains the actual rule (`<ruleml:Rule>`) containing each at most one antecedent (`<ruleml:if>`) and one consequent (`<ruleml:then>`) [15]. Both antecedent and consequent contain expressions (i.e. predicates) of atoms (`<ruleml:Atom>`) and operators (multiple tags). Atoms may have an identifier (attribute `key` or `keyref`).

4.2. Rule Structure

Antecedent and consequent have a more complicated structure than the rest of DAPRECO. They are conjunctions (i.e. logical "and"s) of predicates (s. Section 3.1 of [17]). If the antecedent/consequent doesn't start explicitly with the conjunction tag `<ruleml:And>`, then the expression is a conjunction of only one predicate. With the use of conjunctions you can simplify the transformation into other formats. For example SPARQL `WHERE` statements are also a conjunction of SPARQL patterns. SPARQL patterns are either a triple, a SPARQL expression or a SPARQL function [28].

One problem being faced is, that predicates in DAPRECO use multiple different XML structures. A predicate type might be given as a RuleML tag (e.g. `<ruleml:After>`) containing variables `<ruleml:Var>` as children. At the same time the type of a different

predicate may be defined in a `<ruleml:Rel>` tag, with the variables not as children, but as sibling elements.

To get an overview about all possible structures or elements, I took a look into the RuleML XML sheet definitions (XSD files) shipped with LegalRuleML [15]. As XSD files are in XML format, I used the xs3p stylesheet [57] to generate a XHTML document being displayed in a web browser. This enables human readability via inserting the xs3p XSLT document into the XSD file[5].

LegalRuleML ships three versions of XSD files (i.e. basic, normal and compact), each with different RuleML XSD files. For the analysis below I used the basic version, and checked pairwise for differences in both the LegalRuleML and RuleML definitions via the command `diff`. No differences being relevant for the converter could be found. As LegalRuleML XSDs contain relevant references to LegalRuleML definitions from RuleML (e.g. `lrml:PremiseFormula.choice`), you cannot disregard those XSDs for resolving RuleML rules.

xs3p doesn't resolve references between two documents, and also has no equality check for XML scheme constructs. For example `And-node.choice` is the same as `And-fo-node.choice`, if you disregard irrelevant tags for reified I/O logic. The only difference is that the first one is for antecedents and the latter for consequents. Thus all references have to be resolved manually and checked for generalisation. This is done by creating a temporary list of all possible children elements/models and then manually and recursively resolving all references between RuleML and LegalRuleML XSD definitions, until every relevant element and XML structure is mapped. This cannot be done automatically, because there is to the author's best knowledge no list containing only XML elements relevant for reified I/O logic. Elements, which have been determined as irrelevant, are `<lrml:Override>` containing any content of any namespaces, as well as `<ruleml:Skolem>`, `<ruleml:repo>`, `<ruleml:resl>`, `<ruleml:slot>`, `<ruleml:Plex>` used for positional and slotted knowledge formalisation (s. glossary to the XSD file of [15]).

The following gives an abstract description of a tree of only relevant elements from the conjunction of an antecedent/consequent as root – as used in DAPRECO.

A conjunction contains one or more predicates, which each can be a …

- RuleML operation `<ruleml:Naf>` for negation as failure, which means that the operator returns true, if the child predicate (i.e. any elements below) is false or undeterminable [31]. Alternatively there is `<ruleml:Neg>` or its alias `<ruleml:Negation>` which negates an existing expression. Other top-level operators[6] than those two would not occur in reified I/O logic. `<ruleml:Naf>` only occurs in the antecedent.
- `<ruleml:Interval>`, `<ruleml:Spatial>`, `<ruleml:Time>` as well as `<ruleml:Equal>`, `<ruleml:Or>`, `<ruleml:Imp>` containing the same elements as in a `<ruleml:Atom>` (thus being resolved recursively) – except `<ruleml:Rel>`, which is unnecessary due to the determination of the predicate type by the element tag name.

5 `<?xml-stylesheet type="text/xsl" href="path/to/xs3p.xsl"?>`

6 `<ruleml:Operation>` (only in the consequent)

- `<ruleml:Atom>`, containing …
  - Optional `<ruleml:Rel>` determining the predicate type. Alternatively `<ruleml:After>`, `<ruleml:Before>`, `<ruleml:Every>`, `<ruleml:Any>`, `<ruleml:Timer>` as well as `<ruleml:Equal>`, `<ruleml:Or>`, `<ruleml:Imp>` containing the predicate arguments below. As in reified I/O logic there are no nested predicates, but only flat conjunctions [17], no element can be nested into the other – except for `<ruleml:Expr>`. As `<ruleml:Interval>`, `<ruleml:Spatial>`, `<ruleml:Time>`, `<ruleml:Equal>` also occur at top level, they would be arguments, not defining the predicate type of `<ruleml:Atom>`. Because of the nesting prohibition in reified I/O logic they don't occur inside an `<ruleml:Atom>`.
  - Multiple predicate arguments being one or more of `<ruleml:Ind>`, `<ruleml:Data>`, `<ruleml:Var>`, `<ruleml:Expr>`.

A `<ruleml:Expr>` contains a `<ruleml:Fun>` and – because of the restrictions in reified I/O logic – further contains only elements of `<ruleml:Var>`, `<ruleml:Ind>`, `<ruleml:Data>` or `<ruleml:Expr>` (otherwise it could contain all elements in `lrml:AnyTerm.choice`). Expressions still have to be resolved recursively.

RuleML variables (`<ruleml:Var>`) are either of type reference (i.e. they have the attribute `keyref=":`*`varname`*`"`) and point to another variable, or they create a new variable (i.e. they have the attribute `key=":`*`varname`*`"`).

As the `<ruleml:Data>` element may contain any arbitrary value, even XML trees, its children is not resolved, but its text-content used as single atom instead. The optional `xsi:type` attribute is then used as predicate type, defaulting to `xsd:token` (as with `<ruleml:Ind>`).

5. Conversion to SHACL

The converter is implemented in JavaScript, because it is very easy to port JavaScript both to server-side and client-side applications as well as to web environments. To focus on the system itself, no user interface is provided. Before converting from reified I/O logic to SHACL, a basic understanding of SHACL is required. The beginning of a SHACL document [18] is the prefix section for defining namespaces. Both SHACL specific namespaces as well as the LegalRuleML namespaces used in the DAPRECO knowledge base (i.e. `GDPR`, `dapreco`, `rioOnto`, `prOnto`) have to be defined. The prefix section is followed by SHACL constraints [18] separated by the character `"."`. Those constraints are the converted LegalRuleML rules. SHACL constraints start with their name and type identifier, followed by the actual constraint properties (i.e. actual logic) separated by a semicolon. If a target is given to execute the constraint on, this is included as a constraint property (`sh:targetClass …`).

```
@prefix foo: <http://example.org/bar#> .
…

foo:RuleName a sh:NodeShape ;
    foo:constraintContent "bar" ;
    foo:constraintContent2 "bar" .

…
```

Fig. 3: Scheme of SHACL documents

5.1. Rule Design

To the author's best knowledge there is no efficient, convenient and suitable SHACL writing library for JavaScript available at the time of writing. Thus the converter creates SHACL documents from scratch by string concatenation. The resulting SHACL document for the GDPR is used to confirm, that the converter returns syntactically correct SHACL.

Shapes (and thus also constraints or rules) in SHACL need a prefix [18] and prefixes need an identifying Internationalized Resource Identifier (IRI), which is mostly just a URI. As the converter should also be usable for other LegalRuleML datasets, the prefix IRI for the statements must be given as an argument to the converter. In general I defined the custom prefix `stmt` for legal statements (i.e. DAPRECO formulas). In the case of DAPRECO I use the URI of the knowledge base XML file[7]. For other SHACL shapes or SPARQL variables, which are not referring to statements or existing objects, I implemented a custom generator for ephemeral IDs to ensure determinability. By counting the number of generated IDs instead of using e.g. timestamps, you can guarantee that the output is always the same for the same input. This makes it compatible with hash-based file comparison technologies, version control (e.g. backup

7 `https://github.com/dapreco/daprecokb/blob/master/gdpr/rioKB_GDPR.xml#`

systems) and others, even though the exact same input files have been converted at different times.

Formulas/Rules in DAPRECO are used as source elements for transformation into rules, because they allow for more fine-grained reasoning than conversion of complete `<lrml:Statements>`.

Each formula is associated to one of the mutually exclusive (Fig. 9 of [15]) contexts obligation, permission, constitutive rule and constitutive rule for mapping with PrOnto (defined in `<lrml:Context>`). One method to map rules with contexts is the use of `sh:group` as a SHACL constraint property, which is disregarded for optimising the SHACL file size. A more efficient method is the use of inheritance. A context inherits from `sh:NodeShape` (the root element for constraints), while each statement inherits from a context later on. This method both leads to yielding the context of a violated constraint at validation, as well as bypassing the need of resolving machine-readable contexts to their human readable counterpart (e.g. `rioOnto:obligationRule` to `Obligation`).

The latter problem of human readability remains with legal references. With each constraint/rule you also have to encode information about the respective legal reference. Validation information for humans are encoded in the `sh:message` property, which is given as additional information with the respective violated constraint (Section 2.1.5, 3.6.2.7 and 3.6 "Validation Report" of [18]). The parent nodes of rule formulas in DAPRECO (`<lrml:ConstitutiveStatement>`) contain the statement identifier, correlated via an `<lrml:Association>` element to a legal reference. Legal references defined by `<lrml:Prefix>` in [31] use the Akoma Ntoso Naming Convention for legal resources [58]. To provide a generic way of resolving legal references to their human language counterpart[8], one's own legal reference resolver can be given as argument to the converter, defaulting to a non-operative resolver. A non-operative resolver just returns the input i.e. the machine-readable reference. For DAPRECO I implemented a custom Akoma Ntoso GDPR resolver based on regular expressions.

References (e.g. from `<lrml:ConstitutiveStatement>`) to other elements (like contexts) are resolved via XPath expressions. The alternative would be to create a custom context or legal references dictionary with access runtime complexity of *O(1)*, but this approach may have troubles with memory limitations. The drawback of using XPath expressions is the increased need of time due to e.g. parsing XPath expressions and following a multi-dimensional XML hierarchy. In case of DAPRECO a memory limit as given in practice might not cause troubles, but when later on using the converter for other law books (like the German Civil Code [59], which includes itself five law books with over 2380 paragraphs in summary) this might be an issue of feasibility. This is especially the case if the converter is used in browser environments or on low-resource computers. Therefore the dictionary approach is disregarded in favour of the XPath approach.

In DAPRECO every formula (`<lrml:ConstitutiveStatement>`) contains a comment with the complete formula in reified I/O logic representation, being followed by its XML `<ruleml:Rule>` representation. Only the XML representation is converted to SHACL rules. Even though SHACL rules can have non-validating properties like `sh:name`,

8 Machine readable: e.g. `GDPR:art_5__para_1__content__list_1__point_a`
Human readable: e.g. `GDPR Art. 5(1) lit. a`

`sh:description` [18] or `rdfs:label` [60] for such comments, they are not encoded by the converter to mitigate redundancy.

5.1.1. Evaluation Order

SHACL constraints/rules also allow for being executed in a specific order via the property `sh:order` (incrementing from zero) in each constraint property or via the property `sh:entailment` in the SHACL constraint [60]. DAPRECO mentions the recommendation to evaluate constitutive rules before obligations and permissions (Section 2.1 of [31]). Rules in DAPRECO are not ordered according to this scheme (s. order of `lrml:Context` associations), so to optimise the amount of needed iterations of the rule engine, the order should be explicitly defined. As this recommendation is not only valid for DAPRECO but also other legal formalisations [12], the determination of order can be hardcoded into the converter. As the SHACL property `sh:order` is only for properties of SHACL rules (e.g. multiples rules in one SHACL constraint), it defines the order of rule execution *within* `sh:NodeShapes` containing multiple SHACL constraints – but not between constitutive and regulative constraints. SHACL provides the property `sh:entailment` for this, which is given the value `sh:Rules` for constitutive rules. This indicates to a validation engine, that these constraints have to be executed before the other ones. To distinguish between constitutive and regulative rules, only the beginning of the context string is compared, to also include variations of e.g. `rioOnto:constitutiveRule` like `rioOnto:constitutiveRule4MappingWithPrOnto` as seen in DAPRECO [31] – even though no variations of `rioOnto:permissionRule` and `rioOnto:obligationRule` have been used there yet. This facilitates forward-compatibility. To reduce the filesize of the output SHACL document, the `sh:entailment` property will be inherited from the contexts defined before.

There might be also a much more fine grained ordering on per-rule-level possible i.e. which rule or constraint to execute first. Even though defeasible statements indicated with negation-as-failure (`<ruleml:Naf>`) can be postponed in rule execution, it is difficult to map an exact `sh:order` in native RDFs/OWL (s. Section 6 of [32]). Additionally inferred triples of constitutive rules only become visible to rules with higher order, but not same order (s. Section 8.1 of [60]). Thus a rule engine needs multiple iterations of inference [60].

5.1.2. Quantifiers

Reified I/O logic makes uses of quantifiers [17]. Thus antecedent/consequent may be nested into the existential quantifier `<ruleml:Exists>` (∃). Only direct children of this quantifier being a variable `<ruleml:Var>` are then used with this quantifier (s. structure in [31]). All other variables are used with the universal quantifier (∀). This also covers variables inside a `<ruleml:Forall>` quantifier, not being used explicitly in DAPRECO. You can find an explanatory example below.

∀a ∀b: (a ∧ b)

```
<ruleml:And>
  <ruleml:Atom><ruleml:Var key=":a">a</ruleml:Var></ruleml:Atom>
  <ruleml:Atom><ruleml:Var key=":b">b</ruleml:Var></ruleml:Atom>
</ruleml:And>
```

∃a ∀b: (a ∧ b)

```
<ruleml:Exists>
  <ruleml:Atom><ruleml:Var key=":a">a</ruleml:Var></ruleml:Atom>
  <ruleml:And>
    <ruleml:Atom><ruleml:Var keyref=":a" /></ruleml:Atom>
    <ruleml:Atom><ruleml:Var key=":b">b</ruleml:Var></ruleml:Atom>
  </ruleml:And>
</ruleml:Exists>
```

Fig. 4: Encoding of quantifiers in DAPRECO

All variables with a universal quantifier are used to connect variables between antecedent and consequent i.e. they are scoped across the whole `<ruleml:Rule>`. In contrast to that, all variables with an existential quantifier are only scoped within an antecedent or a consequent (s. Section 2 of [32]). As SPARQL doesn't differentiate whether a variable was defined in the antecedent or consequent (s. SPARQL expression definition in [60]), you have to take care of name collisions of existentially quantified variables in the antecedent and consequent. Fortunately DAPRECO was designed to take such collisions into account, because variables occurring both in antecedent and consequent are always bound by a universal quantifier (s. Section 2.3 of [31]).

5.2. Rule Conversion

As DAPRECO is written in reified I/O logic, it uses only a subset of available XML tags and structures of RuleML or LegalRuleML. Thus I narrowed down the input conditions for the converter to a LegalRuleML document in reified I/O logic, as DAPRECO uses it (s. Section 5.5). Rules can be of two types: constitutive statements inferring new facts from other facts and regulative statements asserting facts (derived from Section 2.1 [32]). As both follow a different validation logic, different conversion strategies have to be evolved.

5.2.1. Constitutive Rules

There are two types of native SHACL rules, which can infer triples from facts: `sh:TripleRule` and `sh:SPARQLRule` [60]. Both allow for multiple conditions, of which all of them have to be true for a rule to be applied. `sh:TripleRule` can infer only triples with the same predicate by defining subject, predicate and a `sh:NodeExpression` as objects of an RDF triple. Thus to infer multiple different-predicated triples from a given set of conditions, you would have to use a connecting variable. The connecting variable is true if and only if the conditions are true. Then you can define several rules with the connecting variable as their

condition, each of them inferring a different triple. Despite the use of `sh:SPARQLRule` requires the rule engine to understand SPARQL, it allows for multiple triples to be inferred by a single rule. Rules inferring multiple triples (i.e. facts) with different predicates are needed, because DAPRECO uses different relations (e.g. data processor and data subject correlations) in the same rule. For simplicity reasons the `sh:SPARQLRule` variant of rules is used. The antecedent is mapped inside a SPARQL `WHERE` clause, whereas the consequent fills the `CONSTRUCT` block.

5.2.2. Regulative Rules

Assertion constraints don't infer new triples (or generally speaking "SPARQL patterns" [28]). So a different construct has to be used. To keep a similar structure, SPARQL is used again. The SPARQL expression is then embedded into a SHACL constraint (s. Section 5.1 of [18]). There are two relevant types for SPARQL assertion queries: `ASK` statements, and `SELECT` statements [28]. `ASK` statements return a boolean value, if a set of SPARQL patterns matching the conditions exist. But they don't tell if the whole graph adheres to the conditions. When using `SELECT` statements, only triples are returned, which do not fulfil the validation constraint (s. Section 6.2 of [28]). But in contrast to the `ASK` statement, all triples are assessed – not only a subset of the graph. However the constraints given in reified I/O logic are positive-formulated i.e. they filter for triples fulfilling the condition, not violating them. Thus I had to invert their logic with the following two lemmata: (1) Given a pattern `x` and `c(x)`, where `c(x)` tells whether pattern `x` meets a condition `c` (i.e. the SHACL constraints as SPARQL expression), then `∀x c(x)` is equivalent to `∄ ¬c(x)`; (2) An antecedent `x` and a consequent `y` can be represented as the implication `x imp y = ¬x ∨ y` and converted to the boolean expression `¬(x ∧ ¬y)` using the DeMorgan formula.

As the `SELECT` variant refers to `∄ ¬c(x)`, you can select all triples (i.e. `SELECT *`) which conform to the antecedent (inside the `WHERE` statement) but don't adhere to the corresponding consequent (inside a `FILTER NOT EXISTS` block appended to the `WHERE` statement).

5.3. Translation to SPARQL Patterns

Predicates first have their predicate type and then optionally their name as a variable. After that follows a list of arguments, each being a variable – similar to `(predicateType name arg0 arg1 …)`.

5.3.1. Predicate names

There are two relevant measures for determining the predicate name. The first one is whether a predicate is reified (i.e. unprimed) or not (i.e. primed) [17]. If a predicate is reified, the first argument is always the predicate name [17]. Whether a predicate is reified or not, is indicated in DAPRECO with an `<lrml:Association>` (s. Section 4.1). This doesn't hold by definition for primed predicates [17]. So technically a primed (i.e. not reified) predicate of form `prOnto:DataSubject :w` has by definition no variable name, because `:w` refers to the data subject, but not it's existence. As the hard differentiation between subjects/objects and their existence draws some implications with logic (e.g. you can't say that `:z` is the personal data of the existence of a data subject `:w`), I introduce another differentiation. When having a look into

the DAPRECO XML document, aforementioned problem with logic arises only with descriptive[9] predicates i.e. predicates, that describe e.g. a variable type (like `rioOnto:Controller`). To solve this, a descriptive predicate's name is directly resolved to its subject (i.e. first argument), bypassing the reference to the existence of a subject – given that the predicate is not nameless. Nameless descriptive predicates (e.g. `rioOnto:exception…`), which have no naming variable at all, are used to indicate certain circumstances like exceptions. They are given an ephemeral ID. You can find a list of nameless descriptive predicates in Appendix B, which have been manually determined.

So in summary you can use the first argument of a predicate as its name, if the predicate is descriptive or reified. If none of both conditions are true, an ephemeral identifier will be used as predicate name.

5.3.2. Triples

According to reified I/O logic [17] DAPRECO formulas set multiple existing SHACL objects into a relation to each other (s. `<ruleml:Var>` as direct children of `<ruleml:Exists>`). All of these objects have to have a corresponding variable in the SPARQL expression. Those variables are connected via predicates or functions to SHACL patterns. This conversion from DAPRECO to SHACL was done manually in [32]. But a both custom and incomplete ontology inheriting a subset of PrOnto was introduced there, with predicates not being used in DAPRECO. One example is the use of `shRIOL:has-holder-of-pr` (s. Fig. 2 in [32]), whereas in DAPRECO it is mapped with the predicate name `dapreco:HolderOfPR` (s. XML document [31]). As the ontology and predicates given in DAPRECO are used, the following custom strategy is used for mapping predicates in reified I/O logic to triples.

All predicates having only name and variable are converted to the triple `varname –a→ type` (e.g. `p –a→ prOnto:Processor`). The predicate "`a`" is a synonym to "`rdf:type`" [26], which also respects multiple types e.g. by inheritance. Every argument adds a new triple of the form `varname –type(argument)→ argument`. So the positional notation of arguments (i.e. with indexes) in DAPRECO is converted to a slotted notation (i.e. with argument specifiers like `type(arg)`) for SHACL.

Variables may be defined multiple times and thus may have multiple types. This is possible, because health data are both `prOnto:HealthData` as well as `prOnto:PersonalData`. The use of multiple triples doesn't interfere with SHACL, because multiple types for a variable are possible in SHACL as well (s. "SHACL Types" in Section 1.1 of [18]) e.g. by inheritance – as long as both types are not mutually exclusive, if defined so by an ontology.

5.3.3. Functions

The same strategy goes for logical predicates, with the sole difference that they use a different syntax in SPARQL. Because logical predicates are of functional nature – unlike describing nodes in relation to each other (e.g. `swrlb:add` vs. `prOnto:Processor`) – they are mapped to

9 DAPRECO proposes multiple differentiations of predicates e.g. (non-)declarative (Section 2.2 of [31]) or (non-)semantic (Section 2.3 of [31]). But none of them seem suitable for describing predicates containing no processing logic like operators, expressions or functions. Thus "declarative" and "logical" was chosen as new categories.

SPARQL function bindings i.e. `BIND(`*`expression`* `AS ?`*`var`*`)`. By this `?var` is the reification of `expression`.

The normal representation with SHACL triples, (i.e. `varname –a→ funcName` with arguments `varname –N→ argN` for each argument with index `N`) wouldn't make any sense here, because it doesn't allow the SHACL rule engine to evaluate those functions natively. Also encoding such functions as complete and stand-alone SHACL expression constraints (s. Section 7 of [60]) is discouraged, because if used in regulative scopes the declaration of the SHACL expression is evaluated, not the expression itself.

All functions are accompanied by the SPARQL expression `FILTER(BOUND(?expression) && ?expression != false)`, which allows for asserting that `?expression` is not false or unbound. For the definition of the `FILTER` expression please refer to Section 17.2. of [28].

5.3.4. Optionals

Some of the predicates refer to functions, which accept optional arguments. These predicates might evaluate to true even though their arguments may not exist. For example `rioOnto:or arg0 arg1` (primed form) evaluates to true if one of the optional arguments `arg0` and `arg1` exists. With `rioOnto:and` all of the optional arguments have to exist, but the arguments are syntactically still optional – only the evaluation logic changes.

Every optional predicate (incl. type definition triple and argument triples) is wrapped into its own `OPTIONAL` scope. `OPTIONAL` expressions are not nested, because due to the nature of conjunctions in reified I/O logic some predicates may exist without the others and vice versa.

Functions inside an SPARQL `OPTIONAL` scope remain optional, even though they are accompanied by the `FILTER` expression (s. Section 5.3.3), because the `FILTER` expression itself becomes optional as well. Of course this can be optimised, but this would require a more sophisticated conversion logic.

For determining whether a SPARQL pattern is optional or not, I came up with multiple strategies – given that implemented functions may evaluate to false if an argument doesn't exist. Besides …

1. using the predicate `RexistAtTime var time` as top level non-optional variable, indicating that the `var` evaluates to true at a specific `time` (but not every DAPRECO rule contains the predicate `RexistAtTime`),
2. having only triples as optional expressions (but non-optional functions may be part of that optional triple[10] and some triples might not be wrapped by an expression),
3. using the universal/existential quantification of variables (but it's a completely different concept, because the quantifiers refer only to the reification (i.e. existence) of variables, not their values) or the location of variable definition i.e. XML attribute `key`, instead of `keyref` (but they are defined in first use principle and don't follow a relevant logic, as well as existentially quantified variables are defined at the beginning in the

10 For example `?previous_order` doesn't exist, because the customer orders for the first time.
```
BIND( ?previous_order/date AS ?prev_date)
order –inactive_since→ diff_in_dates( "2023-1-1", ?prev_date )
```
This will fail, because `?prev_date` is unbound. So a difference in dates cannot be calculated.

`<ruleml:Exists>` scope without a special location in the antecedent/consequent itself),

4. creating a dependency tree and using the top-most variable as non-optional variable (but some predicates might have a back-reference to the top-variable as well, creating a circular loop), and
5. manually passing to the converter a list of functions, which require optional arguments,

… I ultimately came up with the following strategy.

If predicates are not reified (i.e. primed) then their evaluation must be true as part of the conjunction. If predicates are reified (i.e. unprimed) then the evaluation refers to the eventuality (i.e. existence) of the predicate (s. Section 2.2. of [31]). Because a reified predicate only defines a variable regarding its eventuality but not its actual value, a reified predicate always evaluates to true. This is analogous to the evaluation of the expression `(variable)` (i.e. primed) and `(variable = …)` (i.e. unprimed). In the first case the variable itself is evaluated, in the second case the assignment to the variable is evaluated, which always yields true. A predicate always yielding true, has no effect in a conjunction. And if a variable doesn't not exist in reified I/O logic, then the variable is unbound in SPARQL. So the use of reification as indicator for optionality circumvents the need for recursively pulling dependent variables into an optional scope (like with a variables dependency tree).

There is one exception to this concept: Whenever negation as failure `<ruleml:Naf>` is used (i.e. evaluate to false if object doesn't exist), then the argument is wrapped into an `OPTIONAL` scope as well. The negation-as-failure expression itself does not become optional (as being proposed as wrapper in [31]). No reification of the argument is needed. This is because negation as failure always takes both the eventuality (i.e. existence) and the value of a variable into account, so the variable itself has to be passed, not it's eventuality.

Apart from this `<ruleml:Naf>` is treated as a function, which cannot be used as an argument. Thus the result value is not assigned a variable name, when implementing `<ruleml:Naf>` into the converter.

5.3.5. Multidirectionality of Predicates

DAPRECO uses different directions for relations in a triple i.e. assumes bidirectionality of predicates. For example in `<lrml:ConstitutiveStatement key="statements124Formula4">` the predicate defining the joint data controller doesn't have arguments for each participating controller (as it should be used consistently with Section 5.3.2 "Triples"), but instead, `rioOnto:partOf` relations are drawn from the controller `:y1` and `:y2` to the joint controller `:y1`. The use of different logic directions here makes it more difficult to map this relation into SPARQL expressions. Therefore a SHACL rule is added, which infers predicates in the counter direction for all triples and thus restores bidirectionality.

5.4. Solving Problems of DAPRECO

Unfortunately DAPRECO [31] has multiple flaws regarding predicates or their prefixes.

5.4.1. Undefined Ontologies

First, some of the prefixes of predicates defined in `<lrml:Prefix>` elements (i.e. `rioOnto`, `dapreco`, `prOnto`) are neither common prefixes[11] nor referring to active URIs. The only exception is the prefix `GDPR`, which refers to a relative IRI (i.e. no URL scheme or host) being an Akoma Ntoso Naming Convention legal reference [58]. `rioOnto` and `dapreco` refer to unavailable locations on the website of one of the DAPRECO authors[12], and `prOnto` refers to an unavailable location at w3id.org, even though this service is used for persistent referrers to changing URIs [61]. The website repository history of w3id.org [61] reveals, that this URI never existed and is neither declared in the issues nor pull requests – assuming that the history of the repository has not been stashed.

Above problem with missing definitions doesn't matter for descriptive predicates. It matters though for logical operations, representing a method or operator (like `rioOnto:imply`), because the interpretation/execution of these predicates may have to be known at validation time. PrOnto [56] in OWL2-DL format seems to be mappable to the predicate scheme with prefix `prOnto` in SHACL format (s. Section 2.3 of [31] and Section 4 of [32]). Nevertheless all non-implemented but used predicates have to be assessed whether they are descriptive or logical. For the prefix `GDPR` this is very easy, as these predicates are only references to legal sources and thus declarative. `swrlb` is a common identifier for an existing ontology and contains by its definition only logical predicates (i.e. methods/operators) [35]. For `rioOnto`, `prOnto`, `dapreco`, `lkif` and `allot` I had to do this typification manually (s. Appendix B). The manual translation from LegalRuleML to SHACL constraints in [32] enables the avoidance of prefixes for undefined ontologies by defining a custom ontology (and prefix) drawn from a subset of PrOnto [56] (s. Section 4 of [32]). But a generic converter as proposed in this thesis has automation as a requirement, so manual conversion like in [32] would be no option. In case of DAPRECO the converter needs to replace logical predicates during conversion. That's why the converter also accepts a section to be prepended at the beginning of the SHACL constraints, which can be used for polyfills. In case of DAPRECO the predicates to be polyfilled are non-implemented logical predicates (i.e. methods/operators).

Another problem of DAPRECO [31] is its use of prefixes, which have not been declared in a `<lrml:Prefix>` statement before. Those prefixes are `allot`, `swrlb` and `lkif`. (s. Appendix A). As those prefixes are commonly known SHACL prefixes, their IRI has been imported via the converter's polyfill section [62][35][63]. `lkif` follows a modular approach. As DAPRECO uses the predicates `lkif:Public_Body` and `lkif:Legal_Person` (from legal-action.owl) as well as `lkif:Role` (from role.owl) from different ontologies, I had to make sure that all predicates are resolved. Fortunately legal-action.owl includes expression.owl, which itself

11 Search on prefix.cc for each respective prefix, additionally manual check if any found prefix matches the use in DAPRECO. Last search: 2023-3-7

12 Last accessed: 2023-3-6

includes role.owl [63]. So only legal-action.owl has to be imported and is used as sole prefix IRI for `lkif`.

5.4.2. Inconsistent Predicate Types

DAPRECO doesn't use `<ruleml:Expr>` for SHACL functions and `<ruleml:Atom>` for predicates consistently. For example a `<ruleml:Atom>` instead of `<ruleml:Expr>` was used for the SHACL formula `swrlb:equal(dapreco:minAgeForConsent(:ep), 16)`. Because of this ambiguity in DAPRECO between SHACL functions and SHACL triples, the converter has to be passed a list of predicates being logical predicates (i.e. SHACL functions). The list may also contain prefixed wildcards (e.g. `swrlb:*`), and is created by assessing manually each uncategorised predicate in DAPRECO with its context. You can find the list in Appendix B.

Indicators (but causally neither sufficient nor necessary) for descriptive predicates are …

- The predicate is not reified (i.e. primed) and contains only one argument
- It describes a subject, object, action or relation as well as attributes of those. All of those have to be named neutrally without the hint of accepting arguments. For example the predicate `Draft` can be mapped to an object. But the predicate `draftOf` requires an argument, of which the draft is. So the predicate returns an argument of the object with a different name (e.g. argument name `Draft`) than the predicate name (i.e. `draftOf`) – thus the latter would be logical. The logic there is the determination of the name of the argument, because it may not be derived from the predicate name directly.
- Some descriptive predicates are given as expressions in DAPRECO. This has no effect on determining whether a predicate is logical or descriptive, because the expression may only refer to an attribute – which is descriptive.
- Nameless descriptive predicates have a variable as first argument, which has been defined by a different predicate and the variable has an incompatible type to the nameless descriptive predicate (e.g. `prOnto:PersonalDataProcessing` and `rioOnto:exception`…). A compatible type would be for example `prOnto:HealthData` and `prOnto:PersonalData`, because health data are also personal data. If a reified (i.e. unprimed) form of a descriptive predicate exist, the predicate must always be nameless, because otherwise it would be sufficient to use the first argument of a non-reified (i.e. primed) predicate.

One indicator for logical predicates is the calculation or evaluation of a value (without just returning it, like with attributes). This also holds for functions which return attributes of a different name than the function name, e.g. `dapreco:ageOf` returns/calculates the `age` attribute (not `ageOf` attribute) of an argument (s. also before). Differentiating between uppercase/lowercase predicates unfortunately doesn't help for distinguishing.

5.5. Conditions for Use

The converter has some conditions on the given LegalRuleML input, which partially have been mentioned before. First of all there must not be name collisions inside each formula (i.e. across antecedent and consequent) for existentially quantified variables. Second the LegalRuleML content must adhere to reified I/O logic as used in DAPRECO [31].

For validation, the RDF graph has to be appended with legal variables of the local jurisdiction e.g. for determining `dapreco:minAgeForConsent`, but also correlations e.g. times `t1` or `t2` of certain actions / processings, and constants e.g. `GDPR` or `GDPRChapter3`. You can find a manually created list of these constants in Appendix A. If during validation regulative rules are not limited in scope to inferred triples only (not RDF graph triples), then the RDF graph has to be sanitised for certain predicates before. For example `prOnto:lawfulness`, `prOnto:fairness`, `prOnto:transparency` is inferred by the SHACL constraints, and should not exist in the RDF graph before. Otherwise the validation may yield a wrong result. The DAPRECO knowledge base only refers to the GDPR, but doesn't take local privacy laws of EU member states or even federal states into account (e.g. BDSG in Germany and BayDSG in Bavaria). Thus you would have to create extending SHACL documents, which append legal variables and constraints to the GDPR constraints.

For validation you can use any SHACL validation engine, which supports SHACL-SPARQL.

6. Discussion

6.1. Suitability of DAPRECO

Unfortunately DAPRECO has some definition errors. For example in `<lrml:ConstitutiveStatement key="statements162Formula1">` the authors of DAPRECO expressed with the predicates `(prOnto:DataSubject :w) & (swrlb:lessThanOrEqual :w 250)`, that the exception of not having to create a records of processing activities holds, if the amount of data subjects is less than or equal to 250. But in the corresponding GDPR Art. 30(5) (Records of processing activities) [1], it says that not the amount of data subjects but employees of the data controller must be less than or equal 250, so that the exception holds.

Secondly the function defined in the `swrlb` prefix compares exact values, without counting its arguments. So `swrlb:lessThanOrEqual` technically checks whether a data subject (not the amount of them) is less than or equal 250 (s. definition in [35]). As a result of this, DAPRECO has both semantic as well as logical errors. Because of this it is not possible to verify the resulting SHACL document of the GDPR semantically, but only syntactically. A shallow syntactical validation was performed with [36], which unfortunately has no SHACL-SPARQL support. The successor is very unresponsive with the 1.1MB SHACL file resulting from the converter in this thesis[13]. Thus the latter one could not be used for syntactical validation. Polyfills can not be implemented as part of this thesis, because it would require manual sanitising of the DAPRECO knowledge base (as discussed before). However there exist other ontologies which at least implement some of the undefined predicates, e.g. the Data Privacy Vocabulary [64] for declarative statements.

All polyfills for expressions with names of descriptive predicates are also just functions for returning the respective attribute of an RDF subject/object e.g. `prOnto:MemberState(:y)` in DAPRECO returns `?result` in the triple `:y –prOnto:MemberState→ ?result`. Unfortunately these cannot be implemented automatically, because the predicate name (e.g. `prOnto:MemberState`) then would represent both a class (e.g. a member state) as well as a SHACL function (i.e. resolving the attribute). This ambiguity is not allowed in SHACL [60].

6.2. Legal Preconditions and Improvements

RDF graphs or BPMN models, which are later converted to RDF graphs, must be very detailed to enable a correct assessment with the SHACL constraints. So the generation of those graphs may consume very much time or resources – similar to creating records of processing activities. Therefore it might be beneficial to create a machine-readable version of the records of processing activities as RDF graphs, and then automatically generate the human-readable version of records of processing activities from that graph – instead of creating both version simultaneously. As already mentioned in Section 2.3, it might also be beneficial to remove irrelevant text, which is the same for each privacy document (e.g. rights of data subjects or

13 Safari Browser v15.6.1 on MacBook Pro (Mid 2012) with 2,9 GHz Dual-Core Intel Core i7 processor and 4GB 1600MHz DDR3 memory running macOS 10.15.7

terminology in privacy policies). As the records of processing activities mainly is for internal control of the company (s. GDPR Recital 82 [1]) as well as for supervising authorities (s. GDPR Art. 30(4) [1]), there is no human-readable version needed at all, if assessment can be done automatically. For simplification and automatic assessment, GDPR Art. 30(3) [1] could be modified to allow machine-readable versions only, in a format given by the supervising authorities.

With an obligation like above for machine-readable, up-to-date records of processing activities (s. GDPR Art. 30 "Records of processing activities" [1]) you would be able to automatically assess the controller from its machine-readable list of privacy practices. GDPR Art. 30 could be changed to make a common format obligatory or at least allow a custom format, being convertible to the format described in this thesis. This would allow for automated enforcement and actual acceleration of supervising authorities' work as stated in Section 3, as well as going through e.g. a company register and assess every company automatically for GDPR compliance.

It would also be very helpful to unify laws (e.g. different laws of GDPR, EU member states and their federal states) – even though this wouldn't matter for validation, if those laws were interoperable machine-readable constraints. As the deterministic assessment of privacy practices may allow for absolute results (i.e. no legal clause like "it depends"), this also may abolish the need for risk based assessments like the data protection impact assessment. Such an impact assessment forces data controllers to implement countermeasures for data processings with high risks for the data subjects (s. GDPR Recitals 84, 90 and 92 [1]). But especially regarding transfers to third countries without adequate level of privacy protection, sometimes the maximum possible security measures have to be taken (e.g. end-to-end encryption for data hosting) – not necessarily because of the risk, but because of countermeasures against the possibility of illegal data processing (s. Schrems II judgement of European Court of Justice [2]). There even is already a catalogue of security measurements [65] published by the European Data Protection Board. The use of such criteria for needed measurements is far more objective than the use of risk assessments, because the latter is done by data controllers and thus may vary due to different knowledge. So the legislative could modify the GDPR, so that data controllers adhere to this catalogue.

Another improvement regarding efficiency at evaluation of the GDPR ruleset may be for cross references to sections or chapters. For example GDPR Art. 48(1) [1] deals with the legal bases for data transfers to third countries without adequacy decision regarding privacy levels. But there is no reference to the exception defined GDPR Art. 44 [1], which declares that those legal bases are only valid if the level of protection is not undermined. So generating machine-readable constraints implementing those exceptions is more difficult than if those exceptions would have been mentioned in each relevant paragraph – regardless of whether the constraints were generated automatically or manually.

Sometimes the legislative defines paragraphs in an unclear way to allow for developments in a specific legal field. This is of course a necessary and desired aspect of law. Unfortunately in its current form, it hinders the development of a system being capable of yielding an absolute result e.g. whether a controller fully complies with the legal regulations or not. Many of those uncertainties in law are resolved by case law over time – or even in advance by recitals and assessments of public authorities. But the legal paragraphs are not updated according to this

case law. One example for uncertainty is GDPR Art. 6(1) lit. f (Legitimate Interest) [1], where it is not defined for which concrete conditions the interests of a data controller outweigh the interests of a data subject. To tackle the problem of weighting interests, the prOnto ontology [56] distinguishes between multiple purpose classes (as already mentioned in Section 4). Another example is the obligation to proving a given consent (s. GDPR Art. 7(1) [1]), where it is sometimes enough to prove that the process is not executed without consent (s. use of checkboxes according to GDPR Recital 33 [1]) – instead of saving a separate document where the data subject expresses the consent. A solution to this problem of incompleteness would be to update the law (or at least an internal document) on every case law decision affecting the respective legal paragraph. By this you could maintain both flexibility for developments as well as having a unified and single source of law.

6.3. Practical Relevance

The first practical relevance of this thesis, is given by its deliverable: A formalisation of the GDPR in a common, cross-platform format namely SHACL. By this the DAPRECO knowledge base becomes more usable than before – not only because LegalRuleML is not designed for direct evaluation [23]. Given that the flaws discussed in Section 6.1 would have been eradicated, the SHACL ruleset allows for automatic compliance checking of data controllers – both by supervising authorities as well as data controllers for a fast and immediate self-check. This may not only reduce the time needed for processing a complaint at a supervising authority, but also allows the systematic supervision of companies regarding privacy laws. For this the local company register can be used. It further enables a consistent and immediate enforcement by eliminating the need of currently missing resources at supervising authorities [4].

DAPRECO formalises the whole GDPR [31], not only a subset which is relevant for data controllers. As the GDPR also contains regulations and duties for public authorities like supervising authorities (s. GDPR Art. 57 "Tasks" and Art. 58 "Powers" [1]), the SHACL rules can even be used to automatically control supervising authorities, not only companies.

Another outcome as part of this thesis is the elaboration of difficulties and preconditions which have to be met for a complete conversion of legal texts to a ruleset for validation with rule engines. These outcomes may of course be adapted to other formalisations of law, not only the GDPR. They can be taken into account when creating formalisations, which need less human participation and contain less legal interpretability. Thus both the degree of automation is increased as well as the biases of legal knowledge bases are reduced (s. Section 3.1). As already seen in Section 1.2, a higher degree of automation raises efficiency significantly.

7. Conclusion

Research question 1 was answered in Section 3. The main automation part for GDPR enforcement is determining whether, why and how a company complies with the GDPR or not. Several mechanisms of a constitutional state like the right to revision of a judgement are not part of the automation, because it requires custom decisions by the data controller.

For research question 2 a converter was be built, which transforms the DAPRECO formalisation of the GDPR into SHACL rules. These can be used for a platform independent and standardised validation of data controllers' privacy practices with SHACL-SPARQL validators. It further enables automated enforcement, if the privacy practices of a company are given as an RDF graph – a standardised format for declaring facts. As the conversion logic is based on the specifications for reified I/O logic [17], LegalRuleML [15] and SHACL [18] – and not only the DAPRECO implementations – the converter may also be used for other law books formalised as reified I/O logic in LegalRuleML.

As you can see with the flaws described in Section 6.1, it might be more precise and also more efficient for validation to implement law books directly as SHACL constraints, than using LegalRuleML as intermediate format. This allows for including external dependencies into the SHACL constraints, for example the list of countries with adequate privacy level. This list is needed for assessing transfers to foreign countries (s. GDPR Chapter 5 [1]), but is only included by reference into the GDPR.

Research question 3 for legal or other preconditions, which enable a complete automation of enforcement, was answered in Section 6. Even though automatic determination of compliance is possible with the means today, it is still not viable to automate enforcement yet. This is mainly due to the fact, that companies have no obligation for complete, compatible, publicly available (or at least for the supervising authority), machine-readable records of processing activities. This precondition won't be fulfilled as long as the legislative does not adapt the GDPR.

If the machine-readable ruleset has not been semantically verified by experts, the use of the SHACL ruleset might lead to wrong results (based on the flaws in DAPRECO, s. Section 6.1). Preliminary the ruleset should only be used as an auxiliary mean for determining GDPR compliance, until the flaws in DAPRECO have been eradicated and the SHACL ruleset is recompiled – or the GDPR is manually converted to a valid ruleset. For a maximum of legal certainty, I even would ask for a machine-readable formalisation of law by the legislative itself.

List of Figures

Fig. 1: BPMN model of enforcement workflow.....13
Fig. 2: Hierarchy of Legal Rules.....16
Fig. 3: Scheme of SHACL documents.....20
Fig. 4: Encoding of quantifiers in DAPRECO.....23

Bibliography

[1] European Parliament, Council of the European Union (2016) *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)*; OJ L 119; https://eur-lex.europa.eu/eli/reg/2016/679/2016-05-04

[2] Court of Justice of the European Union (CJEU) (2020) *Judgment of the Court (Grand Chamber): C-311/18*, Publications Office of the EU; https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:62018CJ0311 ECLI:EU:C:2020:559, Document 62018CJ0311

[3] Federal Republic of Germany (2017) *Federal Data Protection Act*, BGBl. I S. 2097 amended by Art. 10 of BGBl. I S. 1858, 2022 I 1045

[4] EDPS Conference Report (2022) Publications Office of the European Union, Luxembourg; ISBN: 9789292427122, https://edps.europa.eu/data-protection/our-work/publications/brochures/2022-11-10-edps-conference-report-2022-future-data-protection-effective-enforcement-digital-world_en

[5] NOYB – European Center for Digital Rights (2022) *Data Protection Day: 41 Years of "Compliance on Paper"?!*, https://noyb.eu/en/data-protection-day-41-years-compliance-paper; Last-accessed: 2023-3-1

[6] European Data Protection Board (2019) *First overview on the implementation of the GDPR and the roles and means of the national supervisory authorities*, European Parliament; https://europarl.europa.eu/meetdocs/2014_2019/plmrep/COMMITTEES/LIBE/DV/2019/02-25/9_EDPB_report_EN.pdf

[7] NOYB – European Center for Digital Rights (2021) *Austrian DPA has option to fine Google up to €6 billion*, https://noyb.eu/en/austrian-dpa-has-option-fine-google-eu6-billion; Last accessed: 2023-4-14

[8] IAPP, EY (2019) *How would you rate your current level of GDPR compliance? [Graph]*, Statista; statista.com/statistics/1172852/gdpr-compliance-among-eu-and-us-firms/

[9] CMS Legal Services EEIG (2022) *GDPR Enforcement Tracker - list of GDPR fines*, https://enforcementtracker.com/; Last accessed: 2022-12-9

[10] Nuala O'Connor, Edward Snowden, Glenn Greenwald, Noam Chomsky (2016) *A Conversation on Privacy*, University of Arizona; https://news.arizona.edu/story/edward-snowden-compares-privacy-freedom-speech

[11] L. Thorne McCarty (1980) *Some Requirements for a Computer-Based Legal Consultant*, AAAI; https://aaai.org/Papers/AAAI/1980/AAAI80-085.pdf

[12] Xin Sun, Leendert van der Torre (2014) *Combining Constitutive and Regulative Norms in Input/Output Logic*, p. 241-257; Deontic Logic and Normative Systems - 12th International Conference, DEON 2014; https://doi.org/10.1007/978-3-319-08615-6_18

[13] Michael Aikenhead (1995) *Legal knowledge based systems: some observations on the future*; WebJCLI 1995 2; https://www.bailii.org/uk/other/journals/WebJCLI/1995/issue2/aiken2.html

[14] Akoma Ntoso Group (2018), *Akoma Ntoso*; http://akomantoso.org/

[15] OASIS LegalRuleML TC (2021), *LegalRuleML*; https://oasis-open.org/committees/tc_home.php?wg_abbrev=legalruleml

[16] World Wide Web Consortium (2005), *FOL RuleML: The First-Order Logic Web Language*; https://w3.org/Submission/FOL-RuleML/

[17] Livio Robaldo, Xin Sun (2017) *Reified Input/Output logic: Combining Input/Output logic and Reification to represent norms coming from existing legislation*; Journal of Logic and Computation 27 8; https://doi.org/10.1093/logcom/exx009

[18] World Wide Web Consortium (2017), *Shapes Constraint Language (SHACL)*; https://w3.org/TR/shacl

[19] Ping Gong, David Knuplesch, Manfred Reichert (2016) *Rule-based Monitoring Framework for Business Process Compliance*, University of Ulm; http://dbis.eprints.uni-ulm.de/1506/1/TR_Ping_2016.pdf

[20] Razieh Nokhbeh Zaeem, Safa Anya, Alex Issa, Jake Nimergood, Isabelle Rogers, Vinay Shah, Ayush Srivastava, K. Suzanne Barber (2020) *PrivacyCheck v2: A Tool That Recaps Privacy Policies for You*, p. 3441–3444; Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM '20), CIKM '20; https://doi.org/10.1145/3340531.3417469

[21] Tomer Libal, University of Luxemburg (2023) *Automated GDPR compliance checking of documents and processes*, https://www.icomplai.eu/; Research Group: https://wwwen.uni.lu/research/fstm/dcs/research_projects/automated_gdpr_compliance_checking_of_documents_and_processes, Last accessed: 2023-3-1

[22] Tomer Libal (2022) *The LegAi Editor: A Tool for the Construction of Legal Knowledge Bases*, p. 286-289; Legal Knowledge and Information Systems, JURIX 2022; https://orbilu.uni.lu/handle/10993/53269 CC BY-NC 4.0

[23] Ho-Pun Lam, Mustafa Hashmi (2003) *Enabling Reasoning with LegalRuleML*, CSIRO; https://arxiv.org/pdf/1711.06128.pdf

[24] Ho-Pun Lam, Guido Governatori (2009) *The Making of SPINdle*, p. 315-322; Rule Interchange and Applications, Lecture Notes in Computer Science; https://doi.org/10.1007/978-3-642-04985-9_29

[25] World Wide Web Consortium (2001), *Semantic Web Standard*; https://w3.org/2001/sw/wiki

[26] World Wide Web Consortium (2014), *Resource Description Framework (RDF)*; https://w3.org/RDF

[27] World Wide Web Consortium (2012), *Web Ontology Language (OWL)*; https://w3.org/OWL

[28] World Wide Web Consortium (2013), *SPARQL 1.1 Overview*; https://w3.org/TR/sparql11-query/

[29] Piero A. Bonatti, Luca Ioffredo, Iliana M. Petrova, Luigi Sauro, Ida R. Siahaan (2020) *Real Time Reasoning in OWL2 for GDPR Compliance*; Artifical Intelligence 289 103389; https://doi.org/10.1016/j.artint.2020.103389

[30] Enrico Francesconi, Guido Governatori (2019) *Legal Compliance in a Linked Open Data Framework*, p. 175-180; International Conference on Legal Knowledge and Information Systems, Frontiers in Artificial Intelligence and Applications; https://doi.org/10.3233/FAIA190321

[31] Livio Robaldo, Cesare Bartolini, Gabriele Lenzini (2020) *The DAPRECO knowledge base: representing the GDPR in LegalRuleML*, p. 5688-5697; 12th Conference on Language Resources and Evaluation, LREC; https://aclanthology.org/2020.lrec-1.698.pdf CC-BY-NC

[32] Livio Robaldo, Kolawole J. Adebayo (2021) *Compliance checking in reified I/O logic via SHACL*, arXiv; https://doi.org/10.48550/arXiv.2110.07033

[33] RuleML Inc. (2001) *RuleML in RDF*, https://ruleml.org/inrdf.html; Last-accessed: 2022-3-3

[34] World Wide Web Consortium (2013), *RIF In RDF (Second Edition)*; https://w3.org/TR/rif-in-rdf/

[35] National Research Council of Canada, Network Inference, Stanford University (2004), *SWRL: A Semantic Web Rule Language*; https://w3.org/Submission/SWRL

[36] TopQuadrant Inc. (2017) *SHACL Playground*, https://shacl.org/playground; Last-accessed: 2023-3-29

[37] CSIRO (2015) *REGOROUS*, https://research.csiro.au/data61/regorous/; Last accessed: 2023-2-1

[38] Pille Pullonen, Jake Tom, Raimundas Matulevičius, Aivo Toots (2019) *Privacy-enhanced BPMN: enabling data privacy analysis in business processes models*; Software and Systems Modeling 18 6; https://doi.org/10.1007/s10270-019-00718-z

[39] Guido Governatori, Mustafa Hashmi, Ho-Pun Lam, Serena Villata, Monica Palmirani (2016) *Semantic Business Process Regulatory Compliance Checking Using LegalRuleML*, p. 746-762; Knowledge Engineering and Knowledge Management, Lecture Notes in Computer Science; https://doi.org/10.1007/978-3-319-49004-5

[40] Arthur H.M. ter Hofstede, Chun Ouyang, Marcello La Rosa, Liang Song, Jianmin Wang, Artem Polyvyanyy (2013) *APQL: A Process-model Query Language*, p. 23-38; Asia-Pacific Conference on Business Process Management (AP-BPM 2013), Lecture Notes in Business Information Processing; https://doi.org/10.1007/978-3-319-02922-1_2

[41] Harald Störrle (2011) *A Visual Language for Ad-hoc Model Querying*; Journal of Visual Languages & Computing 22 1; https://doi.org/10.1016/j.jvlc.2010.11.004

[42] Catriel Beeri, Anat Eyal, Simon Kamenkovich, Tova Milo (2008) *Querying Business Processes with BP-QL*; Information Systems 33 6; https://doi.org/10.1016/j.is.2008.02.005

[43] Ying Liu, Sebastian Muller, Ke Xu (2007) *A Static Compliance Checking Framework for Business Process Models*; IBM Systems Journal 46 2; https://doi.org/10.1147/sj.462.0335

[44] Ahmed Awad (2007) *BPMN-Q: A Language to Query Business Processes*, p. 115-128; Enterprise modelling and information systems architectures – concepts and applications, EMISA; https://dl.gi.de/handle/20.500.12116/22195

[45] Sherif Sakr, Ahmed Awad (2010) *A Framework for Querying Graph-Based Business Process Models*, p. 1297-1300; Proceedings of the 19th International Conference on World Wide Web, WWW '10; https://doi.org/10.1145/1772690.1772906

[46] Ahmed Awad, Gero Decker, Mathias Weske (2008) *Efficient Compliance Checking Using BPMN-Q and Temporal Logic*, p. 325-341; International Conference on Business Process Management (BPM 2008), Lecture Notes in Computer Science (LNISA); https://doi.org/10.1007/978-3-540-85758-7_24

[47] Simone Agostinelli, Fabrizio Maria Maggi, Andrea Marrella, Francesco Sapio (2019) *Achieving GDPR Compliance of BPMN Process Models*, p. 10-22; Information Systems Engineering in Responsible Information Systems (CAiSE 2019), Lecture Notes in Business Information Processing (LNBIP); https://doi.org/10.1007/978-3-030-21297-1_2

[48] Manuel Blechschmidt (2008) *Using BPMN-Q to show violation of execution ordering compliance rules*, unreviewed; https://manuel-blechschmidt.de/data/BPT_Manuel_Blechschmidt_Compliance_Checking.pdf

[49] Hamza Harkous, Kassem Fawaz, Rémi Lebret, Florian Schaub, Kang G. Shin, Karl Aberer (2018) *Polisis: Automated Analysis and Presentation of Privacy Policies Using Deep Learning*, USENIX Security; https://arxiv.org/pdf/1802.02561.pdf

[50] Florian Schaub, Rebecca Balebako, Adam L. Durity, Lorrie Faith Cranor (2015) *A Design Space for Effective Privacy Notices*, p. 2; Symposium on Usable Privacy and Security 2015, SOUPS; https://usenix.org/system/files/conference/soups2015/soups15-paper-schaub.pdf

[51] Email correspondence with Marit Hansen, Unabhängiges Landeszentrum für Datenschutz Schleswig-Holstein (December 2022), Az. 10.42/22.001

[52] NOYB – European Center for Digital Rights (2022) *Final round of "One Trust" cookie banners: 226 complaints lodged against deceptive cookie banners that are still not complying with GDPR requirements*, https://noyb.eu/en/226-complaints-lodged-against-deceptive-cookie-banners; Last-accessed: 2023-3-1

[53] Article 29 Data Protection Working Party (2018) *Guidelines on Automated individual decision-making and Profiling for the purposes of Regulation 2016/679 (wp251rev.01)*, European Commission; https://ec.europa.eu/newsroom/article29/items/612053

[54] James Popple (1996) *A Pragmatic Legal Expert System*; Dartmouth Publishing Company Limited, ISBN: 1855217392; p. 38 et seq.

[55] Kevin D. Ashley (1985) *Reasoning by analogy: A survey of selected AI research with implications for legal expert systems*; p. 116 from "Computing Power and Legal Reasoning", West Publishing Company, ISBN 0314955704

[56] Monica Palmirani, Michele Martoni, Arianna Rossi, Cesare Bartolini, Livio Robaldo (2018) *Electronic Government and the Information Systems Perspective*, p. 139-152; Electronic Government and the Information Systems Perspective (EGOVIS 2018), Lecture Notes in Computer Science; https://doi.org/10.1007/978-3-319-98349-3

[57] Daniel McFeeters et al. (2011) *xs3p*, FiForms XML Definitions Project; https://xml.fiforms.org/xs3p/ v1.1.5; Last accessed: 2023-3-11

[58] OASIS (2019), *Akoma Ntoso Naming Convention Version 1.0*; https://docs.oasis-open.org/legaldocml/akn-nc/v1.0/akn-nc-v1.0.html

[59] Federal Republic of Germany (2002) *Bürgerliches Gesetzbuch*, BGBl. I S. 42, 2909; 2003 I S. 738 amended by Art. 24 of BGBl. 2023 I Nr. 51

[60] World Wide Web Consortium (2017), *SHACL Advanced Features*; https://w3.org/TR/shacl-af

[61] World Wide Web Consortium (2023) *Permanent Identifiers for the Web*, https://w3id.org/; Last accessed: 2023-3-6

[62] Angelo Di Iorio, Fabio Vitali, Francesco Poggi, Gioele Barabucci, Monica Palmirani, Silvio Peroni (2017), *ALLOT (A Light Legal Ontology On TLCs)*; https://w3id.org/akn/ontology/allot

[63] Leibniz Center for Law, Faculty of Law (2006) *Estrella Project: Legal Knowledge Interchange Format*, http://estrellaproject.org/?page_id=3; Last accessed: 2023-3-7

[64] World Wide Web Consortium (2022), *Data Privacy Vocabulary*; http://w3id.org/dpv

[65] European Data Protection Board (2021) *Recommendations 01/2020 on measures that supplement transfer tools to ensure compliance with the EU level of protection of personal data*, European Commission; https://edpb.europa.eu/our-work-tools/our-documents/recommendations/recommendations-012020-measures-supplement-transfer_en

Appendix A. Extraction Code Snippets

Preparation code for performing XPath queries in a NodeJS REPL runtime, to verify constraints by reified I/O logic on LegalRuleML

```
var dapreco = require('fs').readFileSync('exec/gdpr_2020-05.xml').
      toString();
var dom = new (require('@xmldom/xmldom').DOMParser)().
      parseFromString(dapreco,'text/xml');
var xpath2 = require('xpath2.js');
var ns = xpath2.createStaticContext(
      p=>dom.documentElement.lookupNamespaceURI(p) );
ns.defaultFunctionNamespace='http://www.w3.org/2005/xpath-functions';
var _$ = (query, root) => xpath2.evaluate(query, root, ns);
var $ = q=>_$('/lrml:LegalRuleML'+q, dom);
```

JavaScript code for extracting all used prefixes (except `ruleml`)

```
Array.from(new Set(dapreco.match(/(?<=")\w+(?=:\w+?")/giu))).join(' ')
// => GDPR rioOnto prOnto dapreco lkif swrlb allot
```

JavaScript code for extracting all used constants

```
Array.from(new Set(
      $('//(ruleml:Data|ruleml:Ind)').map(e=>e.textContent)
)).sort().join(' ')
// => 16 1M 250 3M 3y 4y 72h Article33 EuropeanUnion GDPR GDPRArt32
// GDPRArt33 GDPRArt34 GDPRArt35 GDPRArt36 GDPRChapter3 GDPRx
```

JavaScript code template for extracting all predicates for a prefix

```
// `prefix` contains the current prefix to output; one of ruleml,
//    rioOnto, prOnto, dapreco, lkif, allot (to be analysed)
Array.from(new Set(
      dapreco.match(new RegExp(prefix + ':[\\w:]+','giu'))
)).sort().join('\t')
```

JavaScript code for determining if descriptive predicates are nameless

```
// `tags` contains whitespace-separated list of descriptive predicates
tags = tags.split(/\s+/).filter(t=>t).sort(
      (a,b) => a.toLowerCase().localeCompare(b.toLowerCase())
);
// reified predicates are indicated with apostrophe (e.g. predicate')
tags.filter(t => dapreco.includes(t+"'")).join(' ')  // nameless
tags.filter(t => !dapreco.includes(t+"'")).join(' ') // not nameless
```

Appendix B. Predicate Typification

Predicates for prefixes `lkif`, `allot`

Descriptive predicates:

```
allot:Event             allot:FRBRManifestation allot:FRBRWork
lkif:Legal_Person       lkif:Public_Body        lkif:Role
```

Predicates for prefix `prOnto`

Descriptive predicates:

```
prOnto:Action           prOnto:BiometricData    prOnto:codeOfConduct
prOnto:Consent          prOnto:Contract         prOnto:Controller
prOnto:Data             prOnto:DataSubject      prOnto:Document
prOnto:DPO              prOnto:EthnicData       prOnto:GeneticData
prOnto:Health           prOnto:HealthData       prOnto:Judicial
prOnto:LegalSource      prOnto:Marketing        prOnto:Measure
prOnto:MemberState      prOnto:OpinionData      prOnto:Person
prOnto:PersonalData     prOnto:Processor        prOnto:Purpose
prOnto:Recipient        prOnto:Representative   prOnto:Research
prOnto:SensitiveData    prOnto:SexualData       prOnto:Statistic
prOnto:SupervisoryAuthority                     prOnto:ThirdParty
```

Nameless descriptive predicates:

```
prOnto:Communicate      prOnto:Delete           prOnto:designates
prOnto:exceptionCha3Sec2Art15Par4               prOnto:fairness
prOnto:InternationalOrganization                prOnto:isBasedOn
prOnto:isRepresentedBy  prOnto:lawfulness       prOnto:mitigate
prOnto:nominates        prOnto:PersonalDataProcessing
prOnto:Provide          prOnto:Pseudonymise     prOnto:PublicInterest
prOnto:publicInterest   prOnto:Request
prOnto:rioOnto:exceptionCha3Sec2Art15Par3       prOnto:Risk
prOnto:riskinessRightsFreedoms                  prOnto:Store
prOnto:Transmit         prOnto:transparency
```

Predicates for prefix `dapreco`

Logical predicates:

```
dapreco:ageOf     dapreco:allInfoAbout    dapreco:ComplaintProcedureOf
dapreco:copyOf    dapreco:countryOf       dapreco:draftOf
dapreco:HolderOfPR      dapreco:legalBasisOf    dapreco:mediaOf
dapreco:natureOf  dapreco:progressOf      dapreco:ResponsibleFor
dapreco:specified[14]
```

14 like `<ruleml:Naf>`, s. `<lrml:ConstitutiveStatement key="statements276Formula114">`

Descriptive predicates:

dapreco:ApprovedCertificationMechanism dapreco:BindingCorporateRules
dapreco:BodyACM dapreco:BodyCC dapreco:CategoryOf dapreco:ChangeOf
dapreco:CompensationFor dapreco:Complaint dapreco:contactDetails
dapreco:ContactPointFor dapreco:contextOf
dapreco:DataProtectionPolicies dapreco:dpoOrCP
dapreco:EmploymentSocialSecuritySocialProtectionLaw dapreco:feasible
dapreco:Fee dapreco:freedomOfExprAndInfo
dapreco:HasSignificantEffectOn dapreco:HealthProfessional
dapreco:highestManagementLevel dapreco:Icon
dapreco:JointDataController dapreco:JudicialRemedy dapreco:LegalClaim
dapreco:LegalEffectOn dapreco:LegalRequirement
dapreco:legitimateInterest dapreco:LetterReasonFor
dapreco:minAgeForConsent dapreco:nonDelayed dapreco:NotForProfitBody
dapreco:numberOfDSConcerned dapreco:numberOfPDRConcerned
dapreco:PersonalDataRecord dapreco:public dapreco:publicArea
dapreco:publicPowers dapreco:Register dapreco:rightsAndFreedoms
dapreco:StandardContractualClause dapreco:ThePublic
dapreco:vitalInterest

Nameless descriptive predicates:

dapreco:AbleTo dapreco:Access dapreco:AccreditedBy dapreco:accurate
dapreco:AdequateWith dapreco:AdhereTo dapreco:AdviseOn
dapreco:AssistFor dapreco:AttachTo dapreco:AuthorizedBy
dapreco:automated dapreco:AutomatedDecisionMaking dapreco:AwareOf
dapreco:Charge dapreco:clearness dapreco:CompatibleWith
dapreco:Complete dapreco:ComplyWith dapreco:confidentialWrt
dapreco:ConflictOfInterest dapreco:Contain dapreco:CooperateWith
dapreco:DataBreach dapreco:Define dapreco:Demonstrate dapreco:Describe
dapreco:Dismiss dapreco:EasyAs dapreco:electronicForm dapreco:Enter
dapreco:Execute dapreco:ExpertIn dapreco:GiveConsent
dapreco:HasBeenDamaged dapreco:Hold dapreco:IdentifiableFrom
dapreco:Identify dapreco:Implement dapreco:largeScale
dapreco:LegallyUnableTo dapreco:Limit dapreco:LimitedTo
dapreco:LinkBetween dapreco:Lodge dapreco:machineReadableness
dapreco:Mandate dapreco:Monitor dapreco:Nullify dapreco:occasional
dapreco:OfferGoodsOrServices dapreco:oralForm dapreco:Override
dapreco:PartyOf dapreco:PayFor dapreco:Penalise
dapreco:PhysicallyUnableTo dapreco:ProposedToAddress dapreco:Protect
dapreco:Publish dapreco:ReachableFrom dapreco:reasonable
dapreco:ReceiveFrom dapreco:Recognize dapreco:Rectify dapreco:Refuse
dapreco:RelatedTo dapreco:RelevantTo
dapreco:ReligiousOrPhilosophicalBeliefsData dapreco:Represent
dapreco:Request dapreco:requireTooMuchEffort dapreco:Return
dapreco:security dapreco:setToDefault dapreco:SexualOrientationData
dapreco:TakeIntoAccount dapreco:TakenToAddress dapreco:ThirdCountry
dapreco:TradeUnionMembership dapreco:TrainingFor dapreco:upToDate
dapreco:Verify dapreco:ViolationOf dapreco:WithdrawConsent
dapreco:WorkIn dapreco:WriteIn dapreco:writtenForm

Predicates for prefix `rioOnto`

Logical predicates:

```
rioOnto:and       rioOnto:atTime    rioOnto:imply     rioOnto:not
rioOnto:Obliged   rioOnto:or        rioOnto:Permitted
rioOnto:RexistAtTime
```

Descriptive predicates:

```
rioOnto:timeOf
```

Nameless descriptive predicates:

```
rioOnto:cause           rioOnto:likely    rioOnto:necessary
rioOnto:partOf          rioOnto:possible  rioOnto:exception… (49 times)
```

No predicates:

```
rioOnto:reification     rioOnto:obligationRule  rioOnto:permissionRule
rioOnto:constitutiveRule    rioOnto:constitutiveRule4MappingWithPrOnto
```

Typification for prefix `ruleml` (hardcoded into converter)

Logical predicates:

```
ruleml:After      ruleml:And        ruleml:Before     ruleml:Equal
ruleml:Naf        ruleml:Neg
```

No predicates:

```
ruleml:Atom       ruleml:Exists     ruleml:Expr       ruleml:Fun
ruleml:if         ruleml:Ind        ruleml:Rel        ruleml:Rule
ruleml:then       ruleml:Var
```