Representing Consent and Policies for Compliance

Piero A. Bonatti
Università di Napoli Federico II
Napoli, Italy
pab@unina.it

Luigi Sauro
Università di Napoli Federico II
Napoli, Italy
luigi.sauro@unina.it

Jonathan Langens
Tenforce
Leuven, Belgium
jonathan.langens@tenforce.com

***Abstract*—Being compliant with the GDPR (and data protection regulations in general) is a difficult task, that calls for manifold, computer-based automated support. In this context, several use cases related to the management and the enforcement of privacy policies and consent call for a machine-understandable policy language, equipped with reliable algorithms for compliance checking and explanations. In this paper, we outline a set of requirements for such languages and algorithms, and address such requirements with a framework based on a profile of OWL2 and a set of policy serializations based on popular formats such as ODRL and JSON. Such "external" policy syntax is translated into the "internal" OWL2 syntax, thereby enabling semantic compliance checking and explanations using specialized OWL2 reasoners. We provide a precise definition of both the OWL2 profile and the external policy language based on JSON.**

***Index Terms*—Semantic policy languages, Compliance checking, JSON policies.**

I. Introduction

Being compliant with the GDPR (and data protection regulations in general) is a difficult task, that calls for manifold, computer-based automated support. The cost of violations is high, due to sanctions and reputation loss. Consequently, *controllers* – that is, the entities that collect and process personal data – need methods and technologies for managing and enforcing privacy policies, consent, and objective fragments of the regulations. We collectively call such information *data usage policies*. Data usage policies should be encoded with machine understandable languages, in order to automate compliance checking and auditing.

In this way, the record of processing activities required by the GDPR (which is essentially a privacy policy) can be automatically validated against the formalized part of the regulations. Moreover, each data processing operation can be checked for compliance with the available consent before its execution, by analogy with access control; this approach mitigates the risk of disalignment between the controller's privacy policy and the actual behavior of its systems. If the history of data processing operations is stored in a tamper-proof log, then computer-assisted audit tools can help data subjects, data protection authorities, and the controllers themselves to monitor personal data usage, by running ex post compliance checks. Audit tools may include explanation facilities to help users in understanding the consequences of privacy policies and consent.

This work has been carried out within the H2020 European Project TRAPEZE, funded by the European Union under grant n. 883464.

All of these use cases call for an adequate data usage policy language, equipped with a set of algorithms for compliance checking and explanations. The main requirements for such languages and algorithms are the following.

Expressiveness. A data usage policy language should be able to express consent, privacy policies, and the main concepts occurring in the applicable regulations in a machine understandable way, in order to enable automated compliance checking. In particular, it should be possible to check whether privacy policies comply with (a formalized fragment of) data protection regulations, as well as data subjects' consent.

Usability. The employees of controllers should be able to write privacy policies that describe the controller's data processing activities. Also data subjects are policy authors, in some respect, as they should be able to configure their consent to personal data processing. However, in this case, the approach is different in practice; it is more similar to choosing from a set of options provided by the controller. Intelligent tools for answering questions like "what if I give this consent?" and "why can this company contact me?" may help data subjects to understand the consequences of the consent they grant.

Also *consent reuse* is related to usability: in order to improve user experience, data subjects should not be asked for consent to data processing, unless strictly necessary. For similar reasons, controllers would like to check whether new business processes and applications can be carried out based on the consent to personal data processing that is already available. In order to foster consent reuse, it is also important to give users ranges of options. If consent requests are too tight, then reuse becomes virtually impossible. If consent requests are too general, then privacy-aware users are going to deny their consent. Thus, consent reuse influences expressiveness, as well, calling for a way to express consent policies at different levels of generality, so as to fit the privacy preferences of each user. This can be achieved by means of a class-oriented approach (see next sections).

Reliability. Improper data usage may have significant costs, due to sanctions, reputation loss, and loss of data processing opportunities. Thus, compliance checking should enjoy the following properties:

- it should be *correct* (no false positives), so as to prevent violations of data protection regulations;
- it should be *complete* (no false negatives), so as not to miss any legal opportunity for data processing.

Interoperability. Personal data processing scenarios involve at least three categories of stakeholders: data subjects, controllers, and data protection authorities. Privacy policies and consent must be understood in the same way by different stakeholders. For example, both when a data subject discloses her personal data to a controller under a given consent, and when data are transferred to a third party with a *sticky policy*,[1] the recipient must understand the policy correctly in order to use the data in a compliant way.

Scalability. The main use cases related to policies are:

- *Policy verification*, that is, checking whether a given policy is coherent and complies with (a formalized fragment of) the GDPR and any other applicable regulations on data protection (including national regulations).
- *Business process compliance with data subjects' consent*. Given a policy that describes the operations carried out by a business process, this compliance check consists in verifying whether the data of a particular user can be used by the process, according to her consent.
- *Audit and explanations*, that consist in deciding – given a set of operations on personal data – whether they comply with a data subject's consent and why (or why not). Such queries can be usful to: controllers (to verify why some operations are denied), data protection authorities (to verify the correct behavior of the controller), and data subjects (to audit the controller's behavior and understand the consequences of their consent to personal data processing). Data subjects may also benefit from "what if" queries, before authorizing the use of their data.

Business process compliance may be subject to *real-time constraints*, and may have to be carried out *before* the data of a particular user are fed to the process. Consider for example the information about telephone calls and internet connections generated by the base stations and the access points of a modern telco provider. With suitable consent, such information can be fed to data-driven applications for a variety of interesting purposes. However, according to the GDPR, without consent such data cannot even be temporarily stored, waiting for a batch process to tell which data must be deleted and which can be forwarded to applications. This means that compliance checking shall be carried out on-the-fly, with a frequency of several thousand checks per second. Moreover, the compliance check – that must be carried out before the data is processed by the data-driven application – must consider *all* the operations that *might* be executed by the application. In other words, the *entire* data usage policy of the application must comply with the user's consent.

Also auditing needs scalable compliance checking algorithms, because they have to process a very large set of operations on personal data, carried out during extended time intervals.

[1]In data transfers, sticky policies specify how the recipient can use the data. Sticky policies are bundled with data when they are transferred, hence the name.

Addressing all of the above requirements simultaneously is challenging; consider that expressiveness is usually in direct contrast with both usability and scalability. In the following sections we propose a semantic approach that addresses all requirements, developed within the H2020 European project TRAPEZE.[2]

The paper is organized as follows. Section II specifies a profile of OWL2, called OWL2 PL, and discusses how a policy language based on this profile addresses the above requirements. In Section III we provide an alternative syntax based on JSON, aimed at improving the usability of the policy language. This section includes an unambiguous translation of JSON policies into OWL2 PL policies and some examples of data usage policies. The paper is concluded with two sections on related work, conclusions and future work.

II. A Semantic Approach to Data Usage Policy Languages

The semantic policy language of TRAPEZE extends the language developed by the H2020 project SPECIAL,[3] by introducing instances, negation, and fixpoints. Since the main focus of this paper is the general design strategy of a semantic policy language addressing the requirements introduced in Section I, for simplicity here we describe only the extension with instances; the other additional constructs involve interesting theoretical challenges (especially related to scalability) that will be the subject of a future paper.

A *data usage policy* can be described by a set of properties. Such properties may change depending on whether it is a privacy policy or consent, and depending on the reference legislation, whose requirements determine the relevant information to be reported in the policy. Common policy properties include

- the personal data categories involved in the processing (demographic, financial, judicial, medical, ...);
- the purpose of the processing (marketing, service provision, research, ...);
- the recipients of the information (the controller, its affiliates, third parties, ...);
- where is information stored (in the EU, in a country with similar data protection guarantees, in a "third country");
- how long is the information stored.

The *consent* of data subjects consists of the above "core" properties, plus properties such as the controller to which consent is given, and the sticky policies that apply to data transfers. The *record of processing* that all controllers have to maintain consists of the core properties, plus additional properties such as the technical measures adopted. Privacy policies may contain further properties such as obligations and legal bases. Despite these differences, the syntactic *structure* of all these policies is similar, which makes it possible to encode them all in a uniform way.

[2]https://trapeze-project.eu/

[3]https://specialprivacy.eu/

In particular, our approach consists in using a suitable *profile* of OWL2 as policy language. A profile of OWL2 specifies which of OWL2's operators can be used, and how they can be composed in compound expressions. The choice of the profile aims at the best tradeoff between expressiveness and complexity. In order to encode data usage policies we developed a suitable profile called *OWL2 PL*. Its grammar is reported in tables I and II. Its formal semantics is OWL2's *direct semantics*.[4]

Roughly speaking, the grammar in table I generates unions of "simple classes" of the form:

```
ObjectIntersectionOf(
    ObjectSomeValueFrom( property 1   class 1)
    ⋮
    ObjectSomeValueFrom( property n   class n)
    DataSomeValueFrom( int_prop 1   interval 1)
    ⋮
    ObjectSomeValueFrom( int_prop m   interval m)
).
```

Their instances represent the operations permitted by the policy, each of which is a structured object with (at least) the properties specified in the class. The value of each *property* i of the operation belongs to the corresponding *class* i or *interval* i specified in the class.

A similar language has been introduced in project SPECIAL. There, *class 1*, . . ., *class* n can be either class names or intersections with the same structure as the above class. In TRAPEZE, property values can also be:

1) *nominals*, that is, singleton classes (defined with the OWL2 operator `ObjectOneOf`);
2) *greatest fixpoints* [5] (that are not illustrated in this paper for the reasons outlined at the beginning of this section).[5]

Moreover, the consent policies of data subjects can be more complex concepts called *histories*, that may include *negation*, i.e. the OWL2 operator `ObjectComplementOf` (not illustrated in this paper).

The structure of OWL2 PL knowledge bases is described by the grammar in table II. They may contain declarations of classes, properties, and instances, as well as several kinds of axioms:

1) subclass relationships between class names and nominals,
2) disjointness relationships between classes,
3) property range restrictions,
4) property functionality assertions (that are needed for a correct reification of permitted operations).

In SPECIAL, nominals were not supported.

Focussing on profiles has the advantage of making the policy framework *vocabulary-neutral*. Parsers, reasoners (such as compliance checkers and explanation tools), and their properties (such as worst-case complexity, correctness, and completeness) do not depend on the property names, class names, and instance names used in the policies. Thus the maintenance of the vocabularies, and their extension to new application domains, impact neither implementations nor their performance. This provides a nice *extensibility mechanism*. Accordingly, we will not discuss any particular vocabulary in this paper. We suggest to adopt the work of the W3C Data Privacy Vocabularies community group (DPVCG),[6] that has recently released a rich set of terms that extensively cover the key terms of the GDPR, plus a set of more specific terms of common interest for many use cases (data categories, purposes, technical measures, and so on) [11]. The vocabularies of the DPVCG collect and harmonize the output of previous relevant initiatives and constitute an interesting, standard upper ontology, appropriate for interoperability. Other vocabularies with similar objectives are GConsent [9],[7] GDPRtEXT [10], and GDPRProv.[8]

With OWL2 PL we address the following requirements.

Expressiveness.

OWL2 PL can encode privacy policies, consent, and an objective fragment of the GDPR. Reference [4] contains examples taken from the use cases of the H2020 project SPECIAL.[9] Similarly, the policies for the use cases of TRAPEZE are being successfully encoded. The use case partners of SPECIAL and TRAPEZE comprise companies such as Caixa Bank, Deutsche Telekom, Ipsos Belgium, Proximus, Thomson Reuters, and public bodies such as Informatie Vlaanderen.

The grammars in tables I and II extend SPECIAL's language with *instances*, that are needed to specify individual controllers, recipients, and data points (such as specific locations). Further extensions are discussed below, in Section V.

At the same time, OWL2 PL is machine-understandable, thereby enabling automated compliance checking and smart explanations such as "what if" and "why" questions, that may help users in improving their understanding of privacy policies, consent, and their consequences.

OWL2 is *class oriented* and policies are encoded as classes. This is of paramount importance to achieve several goals.

First, through the hierarchy of classes and subclasses it is possible to formulate policies at different levels of granularity. A first application of this feature is comparing privacy policies with the (formalized part) of the GDPR, where the former are expressed at a fine or medium level of granularity, and the latter at a coarse level of granularity (see, for example, the restrictions on sensitive data). Moreover, by means of subclassing, it is possible to precisely tune data usage policies. For instance it is possible to state that locations are (or can be) tracked in Europe, or just in Italy, Naples, etc., down to specific

[4]https://www.w3.org/TR/owl2-direct-semantics/

[5]Note that fixpoints are not part of OWL2, so this is actually an extension to the standard.

[6]https://www.w3.org/community/dpvcg/

[7]In this paper it is erroneously stated that SPECIAL's language is restricted to a fixed, rigid vocabulary.

[8]https://openscience.adaptcentre.ie/ontologies/GDPRov/docs/ontology

[9]More examples, formulated with JSON, are included in the next section.

```
     full_class = "ObjectUnionOf(", simple_list, ")"
                | simple_class;
    simple_list = simple_class, simple_class, {simple_class};
   simple_class = "ObjectIntersectionOf(", conjunct_list, ")"
                | conjunct;
  conjunct_list = conjunct, conjunct, {conjunct};
       conjunct = class_name | instance
                | sym_property | num_property;
       instance = "ObjectOneOf(", instance_name, ")";
   sym_property = "ObjectSomeValuesFrom(", property_name,
                  full_class, ")";
   num_property = "DataSomeValuesFrom(", property_name,
                  interval, ")";
       interval = "DatatypeRestriction(", "xsd:integer",
                  "xsd:mininclusive", number,
                  "xsd:maxinclusive", number, ")";

     class_name = see OWL2 specification
  instance_name = see OWL2 specification for named individuals
  property_name = see OWL2 specification
         number = see OWL2 specification for integers
```

TABLE I
EBNF GRAMMAR FOR OWL2 PL QUERIES

```
          axiom = declaration | subclass_axiom | assertion
                | functional_prop | disjoint_axiom | range_axiom;

    declaration = "Declaration(", decl_body, ")";
      decl_body = "Class(", class_name, ")";
                | "NamedIndividual(", instance_name ")";
                | "ObjectProperty(", property_name, ")";
                | "DataProperty(", property_name, ")";

 subclass_axiom = "SubclassOf(", subclass, class_name, ")";
       subclass = class_name | instance;

      assertion =
                "ClassAssertion(", class_name, instance_name, ")";

functional_prop = "FunctionalObjectProperty(", property_name, ")"
                | "FunctionalDataProperty(", property_name, ")";

 disjoint_axiom = "DisjointClasses(", class_name, class_name, ")";
    range_axiom =
          "ObjectPropertyRange(", property_name, class_name, ")";
```

TABLE II
EBNF GRAMMAR FOR THE AXIOMS OF $\mathcal{PL}_{inst}$ ONTOLOGIES

buildings or offices. Another interesting application of class hierarchies is for enabling data reuse for similar purposes (which is explicitly allowed by the GDPR). Assessing whether two purposes are similar, in general, requires a human expert, while consenting to data usage for a class of purposes automatically allows processing for all the purposes in the class (in some sense, the notion of similarity is agreed upon in the policy). In a similar way, through subclassing, controllers can give users a range of more general or tighter consent options, so as to maximize user consent and foster consent reuse – thereby improving usability (cf. the discussion on consent reuse in the introduction).

The class-oriented nature of OWL2 is also essential to check the compliance of *policies* correctly. A policy essentially describes a set of authorizations [3]; in other words – as we mentioned above – the instances of a policy class are the operations permitted by the policy. Accordingly, a privacy policy P complies with a policy Q (be it consent or a formalization of the GDPR) if, and only if, *all the operations authorized by P are also authorized by Q*. In other words, P complies with Q if, and only if, P is a subclass of Q. In OWL2 terms, it must be possible to prove:

`SubClassOf(` P Q `)`

from the knowledge base, which means that compliance checking is reduced to *subsumption checking*.

Interestingly, instance-oriented languages such as RDF and RDFS are not adequate for this purpose, because they have not been designed to describe classes. In RDFS, it cannot be stated that a particular data usage is allowed – say – for recommendation purposes; one can only state that it is allowed for *some* (unspecified) recommendation purpose, or for a single purpose (an instance). Thus policies are either too vague or too specific. Formally, there is no relationship between the intended behavior of compliance and the semantics of the natural encoding of policies as instances in RDF and RDFS: two different instances/policies are logically unrelated even if they have the same properties (so it is not possible to reduce compliance checking to any of the standard reasoning tasks). Besides that, interpreting terms as instances rather than classes hinders the aforementioned advantages of the class-oriented approach.

Interoperability.

OWL2 is a W3C standard, with well-defined syntax and semantics. As such it fits well the basic interoperability needs.

Moreover, through OWL2 PL ontologies, it is possible to define and *share* the meaning of new terms, domain-specific terms, and terms used locally by restricted groups of data controllers. The compliance checker treats such terms correctly based on the axioms contained in the ontologies. As mentioned before, no changes to the code are needed. Adding terms to the standard vocabularies is very simple.

Example 1: In order to add a new domain-specific purpose *RecommendArtEvents* it suffices to add two lines to the ontology that defines the core data privacy vocabularies:

```
Declaration( Class( RecommendArtEvents ) )
SubClassOf(RecommendArtEvents ServiceProvision)
```

The first line declares the new term to be a class (other types are instances and properties). The second line states that it is a special case of service provision.

With reference to DPVCG's vocabularies, *ServiceProvision* is the term whose meaning is more closely related to *RecommendArtEvents*. The above two lines let the reasoner (compliance cheker) conclude that consent to use data for *ServiceProvision* purposes (or any superclass thereof) allows also the recommendation of art events. ∎

The above method for sharing the meaning of terms, and the standard, *unambiguous* formal semantics of OWL2 guarantee that policies are understood in the same way by all correct and complete user agents (of controllers, data subjects, and data protection authorities). This is essential for correct interoperability and effective compliance, as explained in the introduction.

Reliability

The formal semantics of OWL2 provides sharp correctness and completeness criteria for the subsumption checking algorithms on which compliance checking and explanations are based upon. All the algorithms developed in TRAPEZE are formally proved to be correct and complete in the following sense: the algorithms say that P is compliant with Q given the knowledge base $\mathcal{K}$ iff `SubClassOf(`P Q`)` is a logical consequence of $\mathcal{K}$. As an additional benefit, the answers returned by the compliance checker and the various explanation facilities are guaranteed to be coherent with each other.

The details of the proofs will be available soon in the dedicated deliverables. The reader may already see the proof strategy by reading the proofs for SPECIAL's language [2], [4]. The correctness and completeness of the extended engine for TRAPEZE's language follows the same argument: a characterization of canonical models, that are then used to answer subsumption queries.

Each release of the reasoning engine (i.e. the implementation of the correct and complete algorithms) is extensively tested with white-box techniques, and by comparing the output of TRAPEZE's engine over pseudo-random policies with the output of other engines for OWL2, such as Hermit.

The knowledge base that defines the vocabulary, and the classes that encode privacy and consent policies and the formalization of the GDPR, have been tested in various ways. They were checked for consistency using the reasoner for OWL2 PL. The knowledge base and the policies have been also validated empirically, with systematic knowledge engineering and software engineering methods. In particular, the formalization of the GDPR has been designed with a top-down methodology driven by the regulations' text (see SPECIAL deliverable D2.6, Sec. 4), and tested with a white-box approach, that examined all the components of the formalization.

After its validation, the formalized GDPR has been used to validate the privacy policies of the use cases, by checking

their compliance by means of the reasoner for OWL2 PL. It is worth mentioning that this automated validation of the privacy policies of controllers has been explicitly asked for by some of the industrial partners of SPECIAL, and it is already been used routinely in their business processes.

Scalability

Scalability is addressed by using a profile of OWL2 (as opposed to the full language). The standard reasoning tasks in OWL2 PL take polynomial time, while in full OWL2 they are complete for nondeterministic double exponential time. The highly scalable, specialized algorithms developed for SPECIAL are sound and complete with respect to SPECIAL's policy language. Experimental evaluations show that a sequential Java implementation takes only a few hundred μ-seconds per compliance check [4]. The same evaluation will be applied to the extended engine for TRAPEZE's languages. The structure of the corresponding algorithms is very similar to that of the algorithms for SPECIAL's language, so we are confident that TRAPEZE's engines will exhibit a similar performance.

It is worth mentioning that the specialized algorithms for OWL2 PL are much simpler than the reasoners for full OWL2. Our industrial partners have been able to re-implement them in various ways (smart contracts, Javascript). This shows that the adoption of our policy framework does not imply the adoption of any complex, black box, possibly proprietary technology.

Usability

In our framework, privacy policies are lists of property-value pairs, where values can be classes, single values, or similar property-value lists. Conceptually, they are simple to grasp. In order to improve usability, we adopt *external formats* that hide the keywords of OWL2. External formats are discussed in the next section.

So far, the use case partners of SPECIAL and TRAPEZE have succesfully mastered the policy framework. More details about this can be found in the next section, since external formats have been used.

Through the class-based approach of OWL2 PL, controllers can give their users a range of options for consent with different generality, that may reduce the number of consent request and improve user experience (see the discussion on consent reuse).

III. External Formats: JSON Policies

In order to improve usability, we propose to encode policies with suitable *external* formats, such as JSON and ODRL – that many controllers are already familiar with – provided that there exists a well-defined, unambiguous translation into the *internal* format OWL2 PL (such translation defines the formal meaning of the external format).

Some of our industrial partners have successfully tested ODRL as an external format, by proving that their employees are able to write accurate privacy policies. The choice of ODRL was motivated by previous competences in the use of this language, and by ODRL's affinity with privacy-related concepts (both licences and privacy policies describe restrictions on the use of assets). Currently, a streamlined profile of ODRL, specialized for market data, that can be translated into OWL2 PL is being developed by a dedicated W3C community group.[10]

JSON LD is another popular format that, however, is a serialization for RDFS. Recall that RDFS is not well-suited as a semantic policy framework for compliance checking, for the reasons discussed in the previous section. JSON LD can perhaps be used as a crude syntactic encoding (forgetting its intended logical semantics and mapping it onto OWL2 in a nonstandard way), but of course a similar approach is likely to create confusion, because developers tend to interpret it as RDFS, according to its standard semantics.

Thus, in this section we introduce a specialized JSON serialization of OWL2 PL, that we call *JSON PL*. Compared to ODRL, JSON PL has the advantage of being vocabulary-neutral. Moreover, JSON is extensively used in widely adopted frameworks like Ember.js, Angular, React, and Vue.js. Many of these frameworks are written in Javascript, that natively supports JSON. Consequently, typical practitioners are more familiar with JSON than ODRL, and – interestingly – the new ODRL profile for market data has a JSON serialization, too. In summary, a JSON serialization for usage policies is likely to foster the adoption of the semantic policy framework based on OWL2 PL. We are now ready to illustrate the features of JSON PL.

Both controllers and data subjects may have multiple policies, that cover different data categories, different purposes, etc. Such policies typically use the same namespaces and ontologies. In order to represent multiple policies in a compact way, the JSON representation supports structures like:

```
{
  @policySet : [ policy 1, policy 2, … ],
  @ontologies : [
      core DPV ontology,
      domain ontology 1,
      ... ],
  @context : [
      "@base" : "default namespace",
      "prefix 1" : "full namespace 1",
      ... ]
}
```

that relate the policies with the OWL2 PL ontologies where terms are defined, and define namespace abbreviations that apply to all the policies in @policySet. The ontologies specify also the type of each term (classes or instances).

The following policy illustrates most of the features of JSON PL:

```
{
"hasDataCategories":  { "@intersectionOf":
     ["Location", "Anonymized"] },
 "hasPurpose": [ "Development", "Security" ],
 "hasRecipient": "acme:GOOGLE",
 "hasDurationInDays": { "@interval": [30, 90] }
}
```

[10] https://www.w3.org/community/md-odrl-profile/

Suppose that it represents the consent of a data subject, that is, what can be done with her personal data.[11] This policy allows anonymized locations to be processed for development and security purposes; the information can be kept for a minimum of 30 days, and no longer than 90 days; the data is shared with Google (the term GOOGLE is defined in the namespace abbreviated by "acme").

The second example formalizes part of the GDPR's article on data transfers to other countries. The policy set is as follows:

```
[
  {
    "hasStorage": {
       "hasLocation": ["EU", "EULike"] }
  },
  {
    "hasRecipient": "Ours",
    "hasOrganisationalMeasure":
                      "BindingCorpRules"
  }
]
```

This policy set states that at least one of the following conditions should hold:

1) the personal data involved in the processing is stored in the EU, or in a set of countries with comparable data protection laws (represented by the term EULike);
2) the data remains within the controller's organization ("Ours"), and its usage is restricted by binding corporate rules.

The complete regulation on data transfers comprises further cases, that are omitted here due to space limitations. With a similar approach it is possible to specify the other objective parts of the GDPR, for example the obligations and the technical and organizational measures required for sensitive data and particular legal bases. For more details see SPECIAL's deliverable D2.6, Sec. 4.

The full set of JSON PL features and their translation $\tau(.)$ into OWL2 PL are illustrated in Table III.

IV. Related Work

Despite superficial similarities, TRAPEZE's policy framework and the many works on legal reasoning have different goals. The survey [12] lists several applications of logic and reasoning to the legal domain that can be grouped as follows:

a) Supporting the legislators in writing less ambiguous, possibly normalized legal documents.
b) Modeling legal concepts and definitions.
c) Interpreting the law.
d) Modeling the debates and pleadings that take place in courts, and deriving legal qualifications.

Among these, "Modeling legal concepts and definitions" is the application that most closely relates to the work on vocabularies carried out by the DPVCG, while our compliance checking with respect to consent and GDPR's partial formalization does not really match any of the above points. Our validations w.r.t. the GDPR are less ambitious than legal reasoning; we only aim at checking whether the different properties of the policy are mutually coherent (e.g. by checking that the legal basis is appropriate for the data category), and whether all relevant parts have been included (such as the required obligations in case of consent-based processing or data transfers outside the EU). The latter is a way to check whether the humans responsible for personal data processing have "ticked all the necessary boxes", and by no means tackles the legal reasoning required to assess whether the obligations have been actually and appropriately fulfilled. According to our experience in SPECIAL and TRAPEZE, algorithms and ontologies are not yet trusted on this matter – especially due to the severe consequences in case of wrong decisions.

From a more technical perspective, both privacy policy validation with respect to the GDPR, and checking whether a privacy policy complies with user preferences, require checking that policies do the right thing in *all contexts*, that is: *all* the operations permitted by the privacy policy should comply with the GDPR or user preferences, respectively (cf. the discussion about the reduction of compliance checking to subsumption checking in Section II). On the contrary, the literature on legal reasoning focusses on whether a *specific* legal qualification (such as an obligation, a permission, the validity of a contract, etc.) holds in a *specific situation* [12]. This difference has remarkable technical consequences: validation in all contexts is intractable in rule-based languages (that are common in the works on legal reasoning, since the seminal paper [14]), and even undecidable if rules are recursive [1]. This makes rule-based approaches unsuitable for checking the compliance of *entire policies* with other policies or regulations. By adopting OWL2 PL, we guarantee decidability and efficiency. This issue can also be analyzed formally: the rule-based approaches to legal reasoning are designed for *answering* conjunctive queries, while the tasks pursued by TRAPEZE are essentially *query containment* problems. For a detailed analysis of this kind, see [4, Sec. 5.3].

Recent work on legal reasoning – leveraging deontic and nonmonotonic logics – has been expressely tailored to the GDPR. The authors of [7], [8] propose the ontology PrOnto for supporting legal reasoning in general, and compliance checking w.r.t. the GDPR. In [6], PrOnto and LegalRuleML (a semantics-neutral interchange format) have been included in a larger architecture for developing GDPR-compliant cloud computing platforms for eGovernment.

The framework proposed in [6]–[8] pursues the more ambitious goals of legal reasoning at large, and operates on a dynamic, workflow-based representation of the controller's activities, more complex than our declarative policies. These choices increase the cost of framework instantiation and rely on users with the necessary legal and logical background (including deontic and nonmonotonic logics) for editing and verifying legal rules – two assumptions that are not satisfied by data subjects nor by controllers. Furthermore, these works

[11] If it were a privacy policy then it would specify what the controller does with personal data.

expr	$\tau(expr)$ (translation into OWL2 PL)
[$exp_1, \ldots, exp_n$] *	ObjectUnionOf($\tau(exp_1), \ldots, \tau(exp_n)$)
$\{p_1{:}v_1, \ldots, p_n{:}v_n\}$	ObjectIntersectionOf($\tau(p_1{:}v_1), ..., \tau(p_n{:}v_n)$) †
$p{:}v$, p object-valued and $p \neq$ "@..."	ObjectSomeValuesFrom($p\ \tau(v)$)
$p{:}v$, p datatype-valued and $p \neq$ "@..."	DataSomeValuesFrom($p\ \tau(v)$)
"@class": "*id*"	*id*
"@intersection": [$v_1, \ldots, v_n$]	ObjectIntersectionOf($\tau(v_1), \ldots, \tau(v_n)$)
"@interval": [min, max]	DatatypeRestriction(xsd:integer xsd:mininclusive min^^xsd:integer xsd:maxinclusive max^^xsd:integer)
a class name *id*	*id*
an instance name *id*	ObjectOneOf(*id*)

*With the exception of the lists in "@..." properties

†Only $\tau(p_1{:}v_1)$ if $n = 1$

TABLE III
TRANSLATION OF JSON PL INTO OWL2 PL

do not address the need for real-time compliance checking with respect to frequently changing consent. We remark again that these works focus on reasoning about specific situations, as opposed to checking whether a policy P complies (in *all* contexts) with another policy or regulation Q (see the discussion at the beginning of this section). Summarizing, we trade advanced legal reasoning capabilities for usability, scalability, and compliance checking of *policies* with respect to other policies.

For an extended comparison of the above works with our approach, see [4, Sec. 3.3].

SPECIAL and TRAPEZE's frameworks for policy compliance "in all contexts" are fundamentally different from access control and usage control (including implemented policy languages like XACML and policy models such as UCON), because access and usage control are only concerned with permission or denial of individual, specific operations and cannot be used for the compliance checks that occur in the use cases of SPECIAL and TRAPEZE (most notably, checking compliance of *entire privacy policies* with respect to regulations, consent and sticky policies). In TRAPEZE, access and usage control are possible as special cases of subsumption checking, making use of nominals to simulate *instance checking* (a standard reasoning task for OWL-based systems, that verifies whether a specific instance belongs to a class – that in our scenarios represent an operation and a policy, respectively).

The same differences hold between TRAPEZE and the mechanisms for access and usage control of BPR4GDPR[12]. BPR4GDPR has also defined ontologies for terms related to the GDPR, access control, and business processes that are consulted by a rule base to check the compliance of a policy with the GDPR. While it is interesting to address business processes, the current version of the ontologies – reported in deliverables D3.1 and D3.2 – is less comprehensive than DPVCG's vocabularies, as far as privacy and the GDPR are concerned.

There is a fundamental technical difference between TRAPEZE's logical framework based on pure OWL2 and the approaches like PrOnto and BPR4GDPR. The latter approaches combine a knowledge graph (the ontology) with expert system rules that use the knowledge graph as data (nodes and binary relationships are like atomic facts). Such rules have two purposes: modeling the inference rules of the logic (e.g. they specify the meaning of the different types of edges of the knowledge graph) *and* encoding part of the domain knowledge (for example, in BPR4GDPR there is a rule for each compliance check that should be made; they depend mainly on the GDPR and should be updated if the regulation changes). Thus, such rules must be validated with respect to both purposes, by proving their correctness and completeness. In TRAPEZE's framework there are no rule bases; inferences are performed by the subsumption checker and the explanation engine, that are proved to be correct and complete once; the proof holds for *all* given OWL2 PL classes and knowledge bases. The only things that need to be validated when the framework is insantiated in a new domain or regulations change, is the set of JSON PL policies that encode business policies, consent, or regulations (if they have been changed). This kind of validation is easier than checking the correct behavior of a set of mutually interacting rules.

[12]https://www.bpr4gdpr.eu/

V. Conclusions and Future Work

Summarizing, in order to support compliance with the GDPR it is important to design a data usage policy language, capable of encoding privacy policies, consent, and objective parts of personal data protection regulations. Such a language should be machine-understandable so as to enable automated compliance checking and explanation facilities, aimed at improving the understanding of the consequences of policies and consent. The requirements for data usage policy languages and related algorithms include expressiveness, usability, reliability, interoperability, and scalability.

We have argued that the above requirements can be successfully fulfilled through external data usage policy languages – that controllers are familiar with – that can be unambiguously translated into an internal semantic format based on OWL2 PL. The algorithms for compliance checking and explanations are specialized reasoners for this OWL2 profile.

The class-oriented approach of OWL2 proved to be essential for achieving all the desiderata; as of today, it seems impossible to obtain comparable results with instance-oriented languages like RDF and RDFS.

OWL2 PL is now being extended to support additional useful operators. One is negation, that is needed to implement dynamic consent [13]; this method for collecting consent – where users can deny consent on the fly, or remove it subsequently – requires a formalization of assertions like "location can be tracked in this city, but *not* in this street". In a naive extension of OWL2 PL with the negation operator of OWL2 (ObjectComplementOf), reasoning becomes coNP-hard, thereby introducing scalability issues. We are working at a different, tractable extension akin to overriding, based on histories of consents and denials. A second useful extension is needed to distinguish sticky policies that allow a limited chain of transfers to third parties, from policies that allow unlimited transfer chains. This feature is being addressed by means of greatest fixpoints. Also in this case, care must be taken to ensure tractability.

Finally, OWL2 PL may be extended with a richer set of datatypes, including strings and other types supported by OWL2.

Of course, usability criteria are stronger for data subjects. One of the workpackages of TRAPEZE is devoted to GUIs and dashboards that help users in monitoring how their personal data are being used.

References

[1] P. A. Bonatti. Datalog for security, privacy and trust. In *Datalog Reloaded - First International Workshop, Datalog 2010. Revised Selected Papers*, LNCS 6702, pages 21–36, 2010.

[2] P. A. Bonatti. Fast compliance checking in an OWL2 fragment. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018.*, pages 1746–1752. ijcai.org, 2018.

[3] P. A. Bonatti, S. De Capitani di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Trans. Inf. Syst. Secur.*, 5(1):1–35, 2002.

[4] P. A. Bonatti, L. Ioffredo, I. M. Petrova, L. Sauro, and I. S. R. Siahaan. Real-time reasoning in OWL2 for GDPR compliance. *Artif. Intell.*, 289:103389, 2020.

[5] G. D. Giacomo and M. Lenzerini. A uniform framework for concept definitions in description logics. *J. Artif. Intell. Res.*, 6:87–110, 1997.

[6] M. Palmirani and G. Governatori. Modelling legal knowledge for GDPR compliance checking. In *Legal Knowledge and Information Systems - JURIX 2018: The Thirty-first Annual Conference, Groningen, The Netherlands, 12-14 December 2018*, pages 101–110, 2018.

[7] M. Palmirani, M. Martoni, A. Rossi, C. Bartolini, and L. Robaldo. Legal ontology for modelling GDPR concepts and norms. In *Legal Knowledge and Information Systems - JURIX 2018: The Thirty-first Annual Conference, Groningen, The Netherlands, 12-14 December 2018*, pages 91–100, 2018.

[8] M. Palmirani, M. Martoni, A. Rossi, C. Bartolini, and L. Robaldo. Pronto: Privacy ontology for legal reasoning. In *Electronic Government and the Information Systems Perspective - 7th International Conference, EGOVIS 2018, Regensburg, Germany, September 3-5, 2018, Proceedings*, pages 139–152, 2018.

[9] H. J. Pandit, C. Debruyne, D. O'Sullivan, and D. Lewis. Gconsent - a consent ontology based on the GDPR. In P. Hitzler, M. Fernández, K. Janowicz, A. Zaveri, A. J. Gray, V. Lopez, A. Haller, and K. Hammar, editors, *The Semantic Web*, pages 270–282, Cham, 2019. Springer International Publishing.

[10] H. J. Pandit, K. Fatema, D. O'Sullivan, and D. Lewis. GDPRtEXT - GDPR as a linked data resource. In A. Gangemi, R. Navigli, M. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai, and M. Alam, editors, *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, volume 10843 of *Lecture Notes in Computer Science*, pages 481–495. Springer, 2018.

[11] H. J. Pandit, A. Polleres, B. Bos, R. Brennan, B. Bruegger, F. J. Ekaputra, J. D. Fernández, R. G. Hamed, E. Kiesling, M. Lizar, E. Schlehahn, S. Steyskal, and R. Wenning. Creating a vocabulary for data privacy. In H. Panetto, C. Debruyne, M. Hepp, D. Lewis, C. A. Ardagna, and R. Meersman, editors, *On the Move to Meaningful Internet Systems: OTM 2019 Conferences*, pages 714–730, Cham, 2019. Springer International Publishing.

[12] H. Prakken and G. Sartor. Law and logic: A review from an argumentation perspective. *Artif. Intell.*, 227:214–245, 2015.

[13] E. Schlehahn and R. Wenning. Legal requirements for a privacy-enhancing big data V2. SPECIAL deliverable D1.6, April 2018. Available through https://specialprivacy.ercim.eu/.

[14] M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, and H. T. Cory. The british nationality act as a logic program. *Commun. ACM*, 29(5):370–386, 1986.