Automated Policy Negotiation: a Four-Course Meal

Patrick Hochstenbach[1,2,*,†], Beatriz Esteves[2,†] and Ruben Verborgh[2,†]

[1]Ghent University Library, Ghent, 9000, Belgium
[2]IDLab, Ghent University - imec, Ghent, Belgium

Abstract

Ensuring that humans and autonomous agents can accurately interpret and act on complex policies is critical to maintaining trust, compliance, and accurate data exchange across the Web. Without interoperable and machine-interpretable policies governing the exchange of personal data on the Web, there is a strong risk of misinterpretation and legal ambiguity when automated agents negotiate and instantiate data-sharing agreements. In this regard, this article identifies a set of challenges that need to be tackled by policy-based agents to have (semi)automated communication and negotiation of data exchange terms for the sharing and using personal data on the Web. A literature review was conducted to assess the progress made in addressing the set of challenges across four domains: (i) compliance checking, (ii) consistency checking, (iii) requirement checking, and (iv) negotiation. By examining the state of the art, we demonstrate that these challenges are interrelated but are addressed using a heterogeneous mix of solutions, prioritizing pragmatism over a formal foundation that applies across all four domains. The way forward lies in the resurgence of logic programming languages that offer essential support for built-in predicates, negation, and meta-programming.

Keywords

Policies, compliance checking, consistency checking, requirement checking, negotiation, autonomous agents

1. Introduction

Current web practices, such as invasive user tracking practices using cookies, have led to distrust in online data and service exchange scenarios [1, 2]. To mitigate this distrust and in an attempt to comply with GDPR requirements, websites have implemented mechanisms such as cookie consent pop-ups to allow users to set their privacy preferences. However, such pop-ups have unclear privacy policies, which often do not comply with the law, undermine user experience, and reduce their willingness to exercise agency over their data [3, 4]. Moreover, no standardization is available for setting these preferences and policies. This not only impairs the clarity and comprehension of data exchanges, leading to inaccurate and unaccountable data transfers, but also eliminates any possibility of automation.

In this context, the development of standardized interfaces that transport data within a *"trust envelope"* — a secure vessel embedding usage policies, provenance and other contextual information — could ensure that information flows with integrity, accountability, and purpose [5]. Such envelopes would allow the sharing of data with trust, allowing people, organizations, and machines to have an evolvable and mutually beneficial relationship, where all parties involved can derive value from data and data-related service exchanges. However, for diverse parties to operate within such dynamic ecosystems, they must agree on using interoperable policies to express data exchange conditions. These policies should be expressive enough to account for technical, societal, and legal constraints while enabling autonomous web agents to negotiate and instantiate precise data-sharing agreements on behalf of humans and organizations. The advent of non-deterministic agents, such as large language models, presents privacy risks [6, 7] and offers little transparency [8] regarding the reasoning behind their decisions. Trustworthy agents must rely on deterministic policy engines that guarantee a uniform

OPAL'25: ODRL And Beyond: Practical Applications And Challenges For Policy-Base Access And Usage Control, colocated with the Extended Semantic Web Conference 2025, June 1–5, 2025, Portorož, Slovenia
*Corresponding author.
Patrick.Hochstenbach@UGent.be (P. Hochstenbach); Beatriz.Esteves@UGent.be (B. Esteves); Ruben.Verborgh@UGent.be (R. Verborgh)
0000-0001-8390-6171 (P. Hochstenbach); 0000-0002-8596-222X (R. Verborgh)

interpretation of policies, so that they can act and interact reliably in decentralized systems such as the web.

This article examines a set of technical challenges that must be addressed when deterministic policy-based agents participate in data-sharing agreements. The ultimate goal is to achieve trustworthy (semi-)automated policy negotiation. These challenges are linked to policy *compliance checking*, *consistency checking*, *requirements checking*, and *negotiation*, all of which are necessary features that agents must possess to be able to negotiate data exchange terms for the sharing and usage of personal data on the web. We provide a definition for each challenge and describe the metrics to be used to evaluate the integration of such features into a policy engine. Moreover, building on these definitions, we describe our vision for a fully automated web agent, which requires a solution for all identified challenges, and provide insight into current technologies that tackle at least one of these challenges. Building on this analysis, we argue that a solution to all challenges must be based on interoperable standards and specifications, including policy languages such as the W3C's Open Digital Rights Language (ODRL) [9]. However, formal semantics must support these standards, as multiple technologies may need to work in tandem to overcome these challenges. Without formal semantics, selecting a technology to address one challenge may impede progress on others. We hypothesize that any trustworthy autonomous web agent that negotiates data exchanges requires at least a technical solution for all four challenges.

The remainder of this article is structured as follows: Section 2 provides an overview of the identified challenges, including definitions and examples; in Section 3, we discuss our vision to solve said challenges towards the development of policy-based web agents; Section 4 provides an overview of related work per identified challenge; and, finally, Section 5 concludes the article and offers future lines of research.

2. Challenges

This section outlines four key challenges in automating policy negotiation, derived from our insights following a literature review on the current state of the art of automated policy negotiation. We identify the minimal capabilities that any policy-negotiating agent should possess: a mechanism to verify whether a state of the world aligns with a set of policies, a method to compare policies and detect inconsistencies, a technique to query a set of policies to uncover missing requirements for fulfilling a policy, and a means to generate new policies. These mechanisms can be found with many names in the literature. We settled for terminology we found in [10] and [11]. An alternative model can be found in Kiruthika's "Lifecycle Model of a Negotiating Agent" [12]. Our model offers a more holistic perspective on the various challenges an automated agent will encounter, whereas Kiruthika presents a more process-oriented view of the different stages of negotiation.

Definition 1. ***Compliance checking*** *verifies whether a state of the world conforms to specified norms.*

For example, consider the following policy and state of the world:

Policy 1. *Alice is* ***permitted*** *to* ***use*** *the file* `data.txt`.

SotW 1. *Alice* ***reads*** *the file* `data.txt`.

We use the terminology "state of the world" as defined in Slabbinck [13] as: "a set of knowledge representing real-world information aiding the evaluation of ... policies". This definition is quite generic and can encompass any facts, actions, or outstanding obligations that can be used as inputs for policy decisions.

Given Policy 1 and SotW 1, a compliance checking process would classify SotW 1 as permitted. We assume in this example that some background information is available indicating that *reading* is a subclass of *using*. The nature of this background information is dependent on the policy framework. When multiple actions are available in the state of the world, then a compliance checking process should indicate if the combinations of actions conforms to the specified norms. This can lead to interesting

challenges when the deontic nature of the verbs used in policies is taken into account. The policy "Alice is **prohibited** to **drink** and **drive**," would permit a state of the world where Alice *drinks*, a state of the world where Alice *drives*, but not a state of the world where Alice *drinks and drives*.

The characteristic feature of compliance checking software is the ability to transform a policy language into computationally *executable* representations. When provided with the state of the world, the compliance assessment should determine the fulfillment status of obligations and the permissibility or prohibition of proposed actions in this state. Metrics to evaluate such a system include evaluating processing speed and scalability, expressivity of the state of the world and supported policies, adherence to (web) standards, and benchmarking with test cases.

Definition 2. ***Consistency checking*** *is detecting whether two or more policies contradict each other.*

Consider a new added following policy:

Policy 2. *Alice is* ***prohibited*** *to* ***use*** *file* `data.txt`.

In this example, Policies 1 and 2 *contradict* each other by granting and denying the same subject the right to perform the same action. In our case, there is no need to introduce any potential state of the world. Regardless of the input, the combination of Policies 1 and 2 remains problematic. In logic, anything can be proven from a contradiction, a principle known as *ex falso quodlibet*. However, beyond logical inconsistency, there is also a trust aspect when two parties create or apply policies containing contradictory norms. While our example is relatively simple, one can easily envision scenarios where interactions between parties handling personal data require careful scrutiny to avoid ambiguities.

Costantino et al. [14] defined other types of conflicts. A policy is considered an *exception* to another policy if it grants or denies the right to perform an action while one or more terms of the policy are a subclass of the terms of the other. For example, when Alice is granted the right to use a file in one policy and denied the right to print the file in another policy (considering print as a subclass of use). Policies are considered *correlated* if they have different effects (granting versus denying an action) and the conditions of the rules intersect each other. For example, if Alice is granted the right to use the file on weekends but denied this right on the first day of the month.

The ODRL policy language provides *defeasibility* mechanisms within their expressivity that can resolve inconsistencies such as the ones demonstrated in the previous examples. One can specify a conflict strategy that demands that the permission rule in Policy 1 must override the prohibition rule in Policy 2. Such an override is known as a *superiority relation* in defeasible logic. To indicate that Policy 1 takes precedence over Policy 2, the rule "Policy 1 > Policy 2" can be introduced, e.g., in a policy enforcement mechanism. Although superiority relations may help mitigate conflicts, they do not guarantee the resolution of every policy conflict. In Example 35 of the ODRL Information Model 2.2 [9], it is demonstrated that specifying `"conflict":"perm"`, i.e., permissions override prohibitions, in one policy alongside `"conflict":"prohibit"`, i.e., prohibitions override permissions, in another policy gives rise to a direct contradiction that cannot be solved and results in a void policy.

A key characteristic of consistency checking software is its capacity to analyze policies in their executable form to identify potential contradictions without needing access to the current state of the world. Instead of executing the policy with the current state of the world as input, the semantics of the policy itself need to be analyzed. Metrics to evaluate such systems include complexity analysis of the algorithms used, processing speed, scalability, and benchmarking with test cases.

Definition 3. ***Requirements checking*** *is querying a set of policies to determine which rights are granted or denied, which obligations must be fulfilled, and which constraints must be satisfied for a right to be granted.*

For example, consider the following set of policies:

Policy 3. *Alice is* ***permitted*** *to* ***play*** *the file* `1999.mp3` *with the* ***duty*** *to* ***pay*** *5 euro.*

Policy 4. *Alice is **prohibited** from **selling** the file `1999.mp3`.*

Given Policies 3 and 4, and considering "Policy 4 > Policy 3" as a superiority relation to deal with policy inconsistencies, a requirements checking tool should be able to answer queries such as:

Q1. Is Alice **prohibited** to **sell** the file `1999.mp3`? *Yes.*
Q2. Is Alice **prohibited** to **pay** 5 euro? *No.*
Q3. Is Alice **prohibited** to **play** the file `1999.mp3`, if she is **not paying** 5 euros? *Yes.*

The queries illustrate both the syntactical and semantic requirements for requirements checking tools. If these policies were expressed in web language such as RDF, Q1 could be addressed syntactically using a SPARQL query using graph pattern matching. However, Q2 and Q3 require an interpretation of the deontic verbs that are used in the policies and queries, and may also need support for negations and defeasible norms. For example, answering Q2 and Q3 requires understanding the relation between a **prohibition** and a **duty** and the deontic definitions of a **duty** (D), **prohibition** (F) and **permission** (P): $D(A) =_{def} F(\neg A) =_{def} \neg P(\neg A)$. A prohibition to "**pay** 5 euro" is identical to a duty "**not paying** 5 euro". A requirements checker should be able to calculate that a duty "not **paying** 5 euro" and a duty "**paying** 5 euro" are incompatible, regardless of the state of the world, even under an open-world assumption. Moreover, to answer Q3, the meaning of **not** needs to be clarified – does **not** indicate that Alice is unwilling to pay (a fact), or that there is no information about the payment (which requires some interpretation in an open-world assumption)? Both interpretations of **not** may lead to the same conclusions, or they may not, depending on the policy framework.

Part of what is expected from a requirements checker overlaps with that of a compliance checker. What differentiates a requirements checker is its ability to query the output of a compliance checking process – for example, to identify the duties that must be fulfilled or to interpret the meaning of a negative compliance result ("computer says no"). Querying both the policies and the output of the compliance process offers insight into the capabilities and outcomes of policy compliance.

The key characteristic of requirements checking software is its ability to make policies in their syntactic or executable form *explainable* to human and machine targets. Possible metrics to evaluate these tools include the expressivity of the query language. Can the tool perform only graph pattern matching, or does it understand the semantics of the policy language? Can the tool answer only 'Yes'/'No' questions, or can it produce more elaborate answers for questions such as "Show all the actions Alice is permitted to do". Furthermore, benchmarks can be imagined to evaluate the scalability or completeness of the results, and user surveys can provide insights into the ability of the tool to provide a human-interpretable result.

Definition 4. ***Policy negotiation*** *is a (semi-)automated process in which two parties establish the conditions and terms for data exchange and usage. The result of such a negotiation process is a policy.*

Consider a scenario in which Alice is the provider of the dataset (D1) containing her personal information, including dietary habits. FoodMarket (FM) is a website that Alice uses to buy groceries. When Alice connects to FoodMarket through her web browser, the FM system automatically filters products that match her dietary preferences, provided that Alice consents to sharing this information with the system (FM). An automated agent in Alice's web browser negotiates with the FM a data exchange and usage policy based on her personal preferences. Alice finds only products that match her diet on the FM website as a result of this negotiation.

For instance, Alice's preferences include the policies:

Policy 5. *Grocery stores are **permitted** to **use** my dietary information for filter queries*

The FoodMarket could have a policy:

Policy 6. *FoodMarket provides filter queries for customers that provide **data sharing** of dietary information.*

The policy agreement after the negotiation process includes:

Policy 7. *FoodMarket is **permitted** to **use** Alice's allergy, intolerance, dietary restrictions, and eating habits data for the purpose of filtering search results.*

This is a theoretical example, without expressing strict GDPR compatibility, aimed at demonstrating the possible outcome of a negotiation. The generated policy materialized *Grocery stores* as *FoodMarket*, *dietary information* as *allergy*, *intolerance*, *dietary restrictions*, and *eating habits*, and *filter queries* as *filtering search results* into an agreement that establishes the conditions for a concrete data exchange instance.

The key characteristics of policy negotiation solutions depend on their ability to analyze the policy requirements of both parties, evaluate these requirements, generate a new policy, check the generated policy for inconsistencies against existing preferences, and present the results of this process in a clear and explainable manner to both parties. Metrics for evaluating such systems involve all the metrics for compliance checking, consistency checking, and requirements checking, as well as user surveys that include user satisfaction and user correction rates (the frequency with which users override the generated policies).

3. Policies Are Essentially a Computer Program.

In this section, we argue that the four challenges of Section 2 are interrelated and arrive at our hypothesis that a trustworthy automated agent requires a solution for all four challenges.

Compliance checking requires software solutions to transform policy documents into executable formats, in some cases necessitating the integration of multiple technologies. The literature, explored in Section 4, highlights implementations in ASP, Prolog, Haskell, OWL, OWL with SPARQL, OWL with SHACL, and OWL with RIF. Each solution has its capabilities, strengths, and weaknesses. Each solution is treated as a black box for this discussion.

Figure 1 provides a graphical illustration of the compliance checker as a black box. The state of the world is codified by the INPUT file that the compliance checker processes into a Yes/No output: "Yes", if the input complies with the policies, and "No" if not. The cases in which compliance software cannot provide an output in this simple abstraction are beyond the scope of this analysis. Moreover, we argue that a policy (or a set of policies) is in effect a computer program. Policies are rules that the compliance checker must follow to arrive at an output. The syntactical form of the policy, e.g., the ODRL document, has to be differentiated from its executable form, e.g., by involving description logics, ASP, SPARQL, SHACL, etc. In this document, we will refer to this executable form as the *formalization* of a policy language.

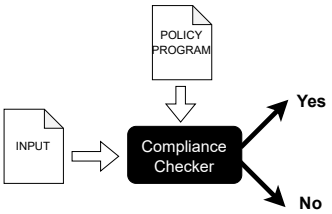

Figure 1: Compliance checking software as a black box.

An automated agent tasked with negotiating policies must compare the terms of data exchange and usage from different parties to generate a new policy with the agreed terms. Within our abstraction, such an agent, in effect, compares *computer programs* representing each party's policies to produce a new program. The terms, conditions, and resulting policy must consistently enable effective data exchange. A policy negotiation agent must also present both parties with an explainable version of the generated policy, ensuring transparency and facilitating mutual understanding.

Such an analysis, comparison, and presentation of explainable policies "as computer programs" is only possible if there is a mutual understanding between the different software solutions – across all four challenges – regarding the meaning and scope of the policies. Such mutual understanding is facilitated by the formalization of a policy's semantics. In such a case, a local system can analyze policies and search for conflicting rules. In the absence of formalization, however, the implementation

defines the policies' semantics. In this case, policy consistency checkers and requirements checkers must interpret the possible execution states of a program, presenting a much more complex challenge.

4. Related Work

A brief literature review was conducted across the four challenges to validate the presented vision. This review is not extensive and provides only an insight into the technologies that are already explored, and if they match our insights. In future work, this analysis will be expanded to share further insights and expand the coverage of the literature review. An overview of the literature mentioned in this section is provided in Table 1. When a solution could apply to more than one challenge, we chose the most relevant categorization.

4.1. Compliance Checking

The topic of policy compliance checking provides the most extensive sources of literature. Three significant trends can be recognized: an early phase that utilizes logic programming languages such as Prolog, a middle phase focused on OWL and Semantic Web techniques, and a current resurgence of traditional programming languages.

Wieringa [15] provides an extensive overview of the application of deontic logic in computer science. He reports on early work in 1985 to implement compliance checking software in Prolog for the Imperial College library and the formalization of the British Nationality Act. Chong [16] introduces "LicenseScript", a language for implementing digital rights management in Prolog.

Gandon [17] uses a combination of RDF Named Graphs, OWL, and SPARQL to formalize LegalRuleML using deontic reasoning. Named graphs are used to capture the state of the world, but because OWL does not have named graph support, extensions in SPARQL had to be added. A similar approach was taken by Francesconi [11] without requiring named graphs. The authors acknowledge that their implementation works for simple cases however it needs to be further extended to model complex modalities and constraints. SHACL is proposed for more complex reasoning. Fornara [18] requires a combination of OWL and RIF to cope with reasoning over temporal intervals modeled with ODRL. Kirrane [19] formalizes the SPECIAL policy language using OWL to develop a compliance checking architecture. Compliance checking is achieved using off-the-shelf reasoners by restricting the deontic requirements of policies to permissions.

In recent years, traditional programming languages have gained renewed attention to formalize policies. De Vos [20] formalized the Institutional Action Language (InstAL) using the ASP programming language. One of the reasons not to use OWL was computational tractability issues when using the RDF open world assumption. Robaldo [21] formalized LegalRuleML using ASP and compared it with a SHACL-based formalization. The authors motivate their choice by the lack of expressivity of OWL to encode defeasibility in normative reasoning. Van Binsbergen [22] formalized the eFLINT policy language in Haskell. One of the reasons for this choice was the fact that the complex conditions in law, regulations, and data sharing agreements could not be captured with existing solutions. Recently, Slabbinck [13] formalized the ODRL policy language using Notation3 with an implementation in Prolog. To validate the approach, an extensive suite of test cases was developed.

4.2. Consistency Checking

On the topic of policy consistency, some theoretical works are available. Gangadharan [23] explores the requirements for service license composition using the "matchmaker" algorithm applied to an extension of the ODRL policy language for service licenses: ODRL/L(S). Villata [25] explores algorithms for license compatibility in the context of data reuse use cases. The authors model licenses using the Creative Commons (CC) vocabulary. Rotolo [26] extends the work of Villata by supporting a broader set of

Table 1
Overview of the related literature per challenge theme.

Challenge	References	Language	Formalization
Compliance checking	Wieringa 1994 [15]	Libary Regulations	Prolog
	Chong 2006 [16]	LicenseScript	Prolog
	Gandon 2017 [17]	LegalRuleML	OWL+SPARQL
	Fornara 2019 [18]	ODRL	OWL+RIF
	De Vos 2019 [20]	InstAL	ASP
	Kirrane 2021 [19]	SPECIAL	OWL
	Van Binsbergen 2022 [22]	eFLINT	Haskell
	Francesconi 2023 [11]	LegalRuleML	OWL+SPARCL+SHACL
	Robaldo 2023 [21]	LegalRuleML	ASP/SHACL
	Slabbinck 2025 [13]	ODRL	Prolog
Consistency checking	Gangadharan 2007 [23]	ODRL/L(S)	*Theory*
	Sensoy 2012 [24]	OWL-POLAR	OWL+Pellet
	Villata 2012 [25]	CC	*Theory*
	Rotolo 2013 [26]	CC,ODC,GNU,...	*Theory*
	Costantino 2018 [14]	CNL4DSA	Maude
	Pellegrini 2018 [27]	ODRL	ASP
	Inclezan 2023 [28]	$\mathscr{AOPL}$	ASP
Requirements checking	Pandit 2018 [29]	GDPRov, GDPRtEXT	SPARQL
	Okoyomon 2019 [30]	*Consent screens*	Regex
	Hamdani 2021 [31]	*Privacy policies*	NLP
	Akaichi 2023 [10]	ODRL	OWL+SHACL
	Adhikari 2025 [32]	*Survey paper*	-
Negotiation	Baarslag 2017 [33]	-	*Protocol*
	Kiruthika 2020 [12]	*Survey paper*	-
	Yumasak 2024 [34]	ODRL	IDSA
	IDSA 2025 [35]	ODRL	*Protocol*

licenses, including Open Data Commons (ODC) and GNU licenses. The authors formalize such licenses using a combination of deontic and defeasible logic.

Sensoy [24] introduces the policy language OWL-POLAR and formalizes it using OWL. The reasoning mechanism for consistency checks is executed using the Pellet reasoner. However, consistency checking in OWL-POLAR is limited due to its reliance on non-monotonic features in the formalization, which are non-standard in OWL-DL. Costantino [14] formalizes data privacy agreements using a language named "Controlled National Language for Data Sharing Agreements" (CNL4DSA). CNL4DSA is an XML language with a formal foundation based on a labelled transition system. This allows for a translation to rewriting logic-based languages of which the Maude System[1] is an implementation. Maude is the formal reasoning tool for which the authors provide conflict detection algorithms. Pellegrini [27] describes the Data Licenses Clearance Center (DALICC) framework, which supports the legal securing of derivative works created from third-party data and software. One of the tasks of the DALICC framework is to analyze license compatibility. The authors formalize licenses using ODRL, which are then translated into ASP for further analysis, enabling the detection of potential conflicting licensing terms. Inclezan [28] describes the $\mathscr{AOPL}$ policy language, designed to specify policies for autonomous agents operating in dynamic environments. The $\mathscr{AOPL}$ language is formalized using ASP, enabling the analysis of policies for inconsistencies (rules that contradict each other), underspecification (rules that never trigger), and ambiguities (rules that permit an action in one answer set but deny it in another).

4.3. Requirements Checking

Three main approaches can be identified when it comes to requirements checking solutions.

[1] https://maude.cs.illinois.edu/wiki/The_Maude_System

The first is a syntactic analysis of the formalization of a policy language. This approach is exemplified by Pandit [29], who formalized the General Data Protection Regulation (GDPR) using the GDPRov and GDPRtEXT ontologies. With these ontologies, subsets of consent and personal data obligations can be queried using SPARQL.

The second approach uses semi-automated heuristics to analyze policies that are published as plain text. Okoyomon [30] uses regular expression (Regex) matching to analyze the consent screens of 68 thousand apps in the Google Play Store. Hamdani [31] employs a natural language processing (NLP) approach to extract data practices from privacy policies. The extracted information is then formalized into rules, allowing further analysis and evaluation. A survey paper of NLP techniques for analyzing privacy polices can be found in Adhikari [32].

The third approach requires the semantic analysis of a policy language. This approach is exemplified by Akaichi [10] in the Generic Graph Pattern-based Policy Framework for Usage Control (GUCON), which investigates the applicability of OWL and SHACL to formalize the ODRL policy language. The authors present an algorithm that analyzes applicable permissions, prohibitions, obligations, and dispensations relevant to a given action.

4.4. Policy Negotiation

Negotiation is a complex process, and its automation is equally intricate. Kiruthika [12] provides an overview of the lifecycle models of negotiation agents in the literature, evaluating techniques such as artificial intelligence, game theory, and evolutionary programming. To our knowledge, no fully automated agents are available to negotiate policies. However, extensive research is available on trust negotiation, determining under what conditions an agent can access a particular piece of information. Semi-automated techniques for negotiating policies can be found in Baarslag [33], who creates a protocol for permission management, and Yumasak [34], who utilizes the (International Data Spaces Association (IDSA) contract negotiation protocol [35]. Both approaches implement a policy negotiation engine, which assists users in streamlining complex negotiations. While the burden on the user to reach an agreement is reduced, Baarslag observes that this does not necessarily lead to decreased user engagement with the systems involved.

5. Conclusions and Future Work

A four-course solution is required if trustworthy (semi-)automated policy negotiation is our ultimate goal. This relies on portable semantics for the policy language across the four components of the challenges as presented in this paper. Implementing policy languages in an RDF model, such as ODRL, ensures the portability of syntax and entity meanings, but this alone does not guarantee a portable policy logic. To achieve this, the conclusions drawn by a compliance checker must align with those of a consistency checker, requirements checker, and policy negotiator, requiring a common logic.

The logic behind implementing RDF-based policy languages using combinations such as OWL+SPARQL+SHACL remains unclear. In its executable form, e.g., as input to a consistency checker, a policy becomes a complex interplay of these technologies' expressivity, obscuring the intended logic of the policy language. The syntax, portability, and application of ODRL for expressing data-related service exchanges are very promising; however, the informal semantics of ODRL complicate the path to a comprehensive solution. Combining deontic and defeasible logic, with support for various forms of negation and possible disjunction, may require formalizations beyond what standard Semantic Web technologies can offer. As highlighted in our literature review, there are signs that the community is gravitating back toward logic programming languages.

We see significant advantages in moving away from Semantic Web languages and returning to logic programming languages like ASP and Prolog for the formalization of policy languages. Our literature study revealed few compelling reasons to use Semantic Web languages, apart from their off-the-shelf availability and the optimistic expectation that Semantic Web techniques can be applied universally. One of the most compelling reasons to use Semantic Web languages can be found in compliance

checking use cases, given the availability of many ontologies to describe the many facets of real-world policies. However, one of the most significant disadvantages is their limited support for built-ins, such as date-time calculations, string parsing, and cryptography, which are indispensable when handling privacy-related data. Semantic Web languages also typically have limited support for classical negation and negation-as-failure, which is required for consistency checking. Exceptions might be found in the Notation3 [36] and RDF Surfaces [37] languages; both are not web standards but provide a potential for a best-of-both-worlds approach. Notation3 provides a rich set of built-ins and a monotonic version of negation-as-failure. RDF Surfaces is a sub-language of Notation3 based on classical first-order logic and supports classical negation.

Our community lacks many implementations for the second, third, and fourth courses. However, developing a non-trivial subset of ODRL with formal semantics for the first three courses may inspire progress on the ultimate challenge – the fourth course, the "dessert" of policy negotiation. With ODRL, we have a standardized syntax for expressing data policies; now, we need to identify the logic that can support the formal meaning of these policies across all challenges. To find inspiration for our path, we should not wait for the perfect solution but instead pursue any direction that may lead us toward a fulfilling outcome. As demonstrated by Google and Apple, the path to real-world agentic AI remains long and challenging.[2] Our insights do not resolve the issues of agency and planning that automated agents must overcome. However, we emphasize that deterministic AI, particularly in the form of symbolic reasoning, should be an integral part of the solution to building trustworthy agents. Achieving this requires a unified syntax and logic approach for all use cases. Mixing many logics to formalize a policy language will only push us further from our shared ultimate goal.

Acknowledgments

This research was funded by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10).

Declaration on Generative AI

During the preparation of this work, the author(s) used Grammarly in order to: Grammar and spelling check. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] N. Samarasinghe, M. Mannan, Towards a global perspective on web tracking 87 (2019) 101569. doi:10.1016/j.cose.2019.101569.

[2] M. Degeling, C. Utz, C. Lentzsch, H. Hosseini, F. Schaub, T. Holz, We value your privacy ... now take some cookies: Measuring the GDPR's impact on web privacy, in: Proceedings 2019 Network and Distributed System Security Symposium, 2019. doi:10.14722/ndss.2019.23378.

[3] E. Papadogiannakis, P. Papadopoulos, N. Kourtellis, E. P. Markatos, User tracking in the post-cookie era: How websites bypass GDPR consent to track users, in: Proceedings of the Web Conference 2021, WWW '21, Association for Computing Machinery, 2021, pp. 2130–2141. doi:10.1145/3442381.3450056.

[4] G. Kampanos, S. F. Shahandashti, Accept all: The landscape of cookie banners in greece and the UK, 2021. doi:10.48550/arXiv.2104.05750.

[5] R. Verborgh, No more raw data, 2023. URL: https://ruben.verborgh.org/blog/2023/11/10/no-more-raw-data/.

[2] See for examples the struggles Google and Apple have in rolling out AI agents in https://www.computing.co.uk/event/2025/agentic-ai-complex-google-demis-hassabis and https://www.nytimes.com/2025/03/14/podcasts/hardfork-siri-starlink.html

[6] B. C. Das, M. H. Amini, Y. Wu, Security and privacy challenges of large language models: A survey 57 (2025) 1–39. doi:10.1145/3712001.
[7] B. Yan, K. Li, M. Xu, Y. Dong, Y. Zhang, Z. Ren, X. Cheng, On protecting the data privacy of large language models (LLMs) and LLM agents: A literature review (2025) 100300. doi:10.1016/j.hcc.2025.100300.
[8] J. Marques-Silva, X. Huang, Explainability is not a game, Commun. ACM 67 (2024) 66–75. doi:10.1145/3635301.
[9] R. Iannella, S. Villata, ODRL Information Model 2.2, W3C Recommendation 15 February 2018 (2018). URL: https://www.w3.org/TR/odrl-model/.
[10] I. Akaichi, G. Flouris, I. Fundulaki, S. Kirrane, GUCON: A generic graph pattern based policy framework for usage control enforcement, in: A. Fensel, A. Ozaki, D. Roman, A. Soylu (Eds.), Rules and Reasoning, Springer Nature Switzerland, 2023, pp. 34–53. doi:10.1007/978-3-031-45072-3_3.
[11] E. Francesconi, G. Governatori, Patterns for legal compliance checking in a decidable framework of linked open data 31 (2023) 445–464. doi:10.1007/s10506-022-09317-8.
[12] U. Kiruthika, T. S. Somasundaram, S. K. S. Raja, Lifecycle model of a negotiation agent: A survey of automated negotiation techniques 29 (2020) 1239–1262. doi:10.1007/s10726-020-09704-z.
[13] Wout Slabbinck, Julián Andrés Rojas, Beatriz Esteves, Pieter Colpaert, Ruben Verborgh, Interoperable Interpretation and Evaluation of ODRL Policies, in: Accepted for Semantic Web - 22nd International Conference, ESWC 2025, 2025.
[14] G. Costantino, F. Martinelli, I. Matteucci, M. Petrocchi, Efficient detection of conflicts in data sharing agreements, in: P. Mori, S. Furnell, O. Camp (Eds.), Information Systems Security and Privacy, Springer International Publishing, 2018, pp. 148–172. doi:10.1007/978-3-319-93354-2_8.
[15] R. J. Wieringa, J.-J. C. Meyer, Applications of deontic logic in computer science: a concise overview, in: Deontic logic in computer science: normative system specification, John Wiley & Sons, Inc., 1994, pp. 17–40.
[16] C. N. Chong, R. Corin, J. Doumen, S. Etalle, P. Hartel, Y. W. Law, A. Tokmakoff, LicenseScript: A logical language for digital rights management 61 (2006) 284–331. doi:10.1007/BF03219910.
[17] F. Gandon, G. Governatori, S. Villata, Normative requirements as linked data, in: Legal Knowledge and Information Systems, IOS Press, 2017, pp. 1–10. doi:10.3233/978-1-61499-838-9-1.
[18] N. Fornara, A. Chiappa, M. Colombetti, Using semantic web technologies and production rules for reasoning on obligations and permissions, in: M. Lujak (Ed.), Agreement Technologies, Springer International Publishing, 2019, pp. 49–63. doi:10.1007/978-3-030-17294-7_4.
[19] S. Kirrane, J. D. Fernández, P. Bonatti, U. Milosevic, A. Polleres, R. Wenning, The SPECIAL-k personal data processing transparency and compliance platform, 2021. doi:10.48550/arXiv.2001.09461.
[20] M. De Vos, S. Kirrane, J. Padget, K. Satoh, ODRL policy modelling and compliance checking, in: P. Fodor, M. Montali, D. Calvanese, D. Roman (Eds.), Rules and Reasoning, volume 11784, Springer International Publishing, 2023, pp. 36–51. doi:10.1007/978-3-030-31095-0_3.
[21] L. Robaldo, F. Pacenza, J. Zangari, R. Calegari, F. Calimeri, G. Siragusa, Efficient compliance checking of RDF data 33 (2023) 1753–1776. doi:10.1093/logcom/exad034.
[22] L. T. van Binsbergen, M. G. Kebede, J. Baugh, T. v. Engers, D. G. van Vuurden, Dynamic generation of access control policies from social policies 198 (2022) 140–147. doi:10.1016/j.procs.2021.12.221.
[23] G. R. Gangadharan, M. Weiss, V. D'Andrea, R. Iannella, Service license composition and compatibility analysis, in: B. J. Krämer, K.-J. Lin, P. Narasimhan (Eds.), Service-Oriented Computing – ICSOC 2007, Springer, 2007, pp. 257–269. doi:10.1007/978-3-540-74974-5_21.
[24] M. Sensoy, T. J. Norman, W. W. Vasconcelos, K. Sycara, OWL-POLAR: A framework for semantic policy representation and reasoning 12-13 (2012) 148–160. doi:10.1016/j.websem.2011.11.005.
[25] S. Villata, F. Gandon, Licenses compatibility and composition in the web of data, 2012. URL: https://inria.hal.science/hal-01171125.
[26] A. Rotolo, S. Villata, F. Gandon, A deontic logic semantics for licenses composition in the web of

data | proceedings of the fourteenth international conference on artificial intelligence and law, in: ICAIL13, 2013. doi:10.1145/2514601.2514614.

[27] T. Pellegrini, V. Mireles, S. Steyskal, O. Panasiuk, A. Fensel, S. Kirrane, Automated rights clearance using semantic web technologies: The DALICC framework, in: T. Hoppe, B. Humm, A. Reibold (Eds.), Semantic Applications: Methodology, Technology, Corporate Use, Springer, 2018, pp. 203–218. doi:10.1007/978-3-662-55433-3_14.

[28] D. Inclezan, An ASP framework for the refinement of authorization and obligation policies 23 (2023) 832–847. doi:10.1017/S147106842300011X.

[29] H. J. Pandit, D. O'Sullivan, D. Lewis, Queryable provenance metadata for GDPR compliance 137 (2018) 262–268. doi:10.1016/j.procs.2018.09.026.

[30] E. Okoyomon, N. Samarin, P. Wijesekera, A. Elazari Bar On, N. Vallina-Rodriguez, I. Reyes, A. Feal, S. Egelman, On the ridiculousness of notice and consent: Contradictions in app privacy policies, 2019. URL: https://dspace.networks.imdea.org/handle/20.500.12761/690, accepted: 2021-07-13T09:37:41Z.

[31] R. E. Hamdani, M. Mustapha, D. R. Amariles, A. Troussel, S. Meeùs, K. Krasnashchok, A combined rule-based and machine learning approach for automated GDPR compliance checking, in: Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law, ACM, 2021, pp. 40–49. doi:10.1145/3462757.3466081.

[32] A. Adhikari, S. Das, R. Dewri, Natural language processing of privacy policies: A survey, 2025. doi:10.48550/arXiv.2501.10319.

[33] T. Baarslag, A. T. Alan, R. Gomer, M. Alam, C. Perera, E. H. Gerding, M. Schraefel, An automated negotiation agent for permission management (2017) 380–390.

[34] S. Yumusak, S. Gheisari, J. O. Salas, S. A. Moqurrab, L.-D. Ibáñez, G. Konstantinidis, Data sharing negotiation and contracting, 2024. URL: https://ceur-ws.org/Vol-3828/paper39.pdf.

[35] P. Koen, M. Kollenstart, J. Marino, J. Pampus, A. Turkmayali, S. Steinbuss, A. Weiß, Dataspace protocol 2025-1-RC1, 2025. URL: https://eclipse-dataspace-protocol-base.github.io/DataspaceProtocol/2025-1-RC1/#example-contract-agreement-request.

[36] D. Arndt, W. Van Woensel, D. Tomaszuk, G. Kellogg, Notation3, 2023. URL: https://w3c.github.io/N3/spec/.

[37] P. Hochstenbach, RDF surfaces: Computer says no, in: ESWC 2023 Workshops and Tutorials Joint Proceedings, volume 3443, 2023. URL: https://ceur-ws.org/Vol-3443/.