The Review of Socionetwork Strategies (2023) 17:319–340
https://doi.org/10.1007/s12626-023-00149-9

ARTICLE

Modeling a GDPR Compliant Data Wallet Application in Prova and AspectOWL

Theodoros Mitsikas[1,2] · Ralph Schäfermeier[3] · Adrian Paschke[2,4]

Received: 4 March 2023 / Accepted: 1 August 2023 / Published online: 26 September 2023

Abstract

We present a GDPR-compliant data privacy and access use case of a distributed data wallet and we explore its modeling using two options, AspectOWL and Prova. This use case requires a representation capable of expressing the dynamicity and interaction between parties. While both approaches provide the expressiveness of non-monotonic states and fluent state transitions, their scope and semantics are vastly different. AspectOWL is a monotonic contextualized ontology language, able to represent dynamic state transitions and knowledge retention by wrapping parts of the ontology in isolated contexts, called aspects, while Prova can handle state transitions at runtime using non-monotonic state transition semantics. We present the two implementations and we discuss the similarities, advantages, and differences of the two approaches.

Keywords GDPR · Knowledge Representation · Prova · AspectOWL · Legal Reasoning

Mathematics Subject Classification 68T30 · 68T42 · 68T27

✉ Theodoros Mitsikas
mitsikas@central.ntua.gr

Ralph Schäfermeier
ralph.schafermeier@gmail.com

Adrian Paschke
adrian.paschke@fokus.fraunhofer.de

1 National Technical University of Athens, Athens, Greece

2 Institut für Angewandte Informatik, Leipzig, Germany

3 Leipzig University, Leipzig, Germany

4 Fraunhofer FOKUS, Berlin, Germany

1 Introduction

In the wake of the introduction of the European GDPR (General Data Protection Regulation) in 2016 and its effective enforcement since 2018 businesses worldwide were obliged to review and adapt their data privacy policies if they desired to continue offering their online services to EU citizens [6]. The complexity of regulatory works such as the GDPR and the large amount of parties affected, has led to an increased interest in research on the problem of automatic legal and ethical compliance checking. The foundation of such systems is an adequate formalization and Knowledge Representation (KR) of the normative rules under consideration. This paper studies and compares two different (rule-based and ontology-based) KR approaches, in a concrete application use case for GDPR.

A key GDPR concept is the concept of consent, meaning that the data controller, before initiating any data processing, is required to make an informed consent request to the user (data subject). The user is then free either to provide consent or to deny and disallow any data processing [6, Article 6]. Moreover, the user has the rights to receive the collected personal data in a machine-readable format, and to transmit them to another controller (right to data portability) [6, Article 20].

Personal data wallet infrastructures are of particular interest in this context as their principle design goal is to provide a privacy and data security aware environment for exchanging personal data. Adhering to the above principles, efforts and projects such as W3C Solid [12] aim to give users more control over their personal data. Solid uses Semantic Web technologies to decouple user data from the applications that use them by utilizing data wallets, in which users can store their data, while keeping the data under the user's ownership. This enables users to easily switch between applications that use the same data, and to switch between storage providers that host them, while having access control over them.

A typical architecture of a data wallet ecosystem such as Solid has different components, each having a specific role: an Identity Provider (IDP) manages the user identity information and also provides authentication services, a Data Wallet Provider (DWP) stores user data, and Relaying Parties are applications that can access and process the data. Decoupling user data from the applications requires data to be stored in a structured way, compatible across the DWPs, and provides the user with control over their data, as consent must provided to allow applications to access and process the data [12].

To this end, we present a set of related use cases addressing a generic distributed data wallet scenario, with consent being a central concept both for access control (sharing a picture to other users) and for personal data processing (using personal data for a personalized Web search). We provide two KR implementations using two formalisms that are both sufficiently expressive: AspectOWL [23], a version of OWL extended by means for expressing context-sensitive knowledge, which allows the representation of dynamic and deontic aspects of the domain, and the logic-programming based rule engine Prova [10], which supports non-monotonic scoped reasoning with constructive modular views on the Knowledge Base (KB) using a meta-data annotation language and guard conditions.

The main contribution of this work consists in two proof-of-concept implementations that take into account the dynamicity of the state of affairs and transitions between different states (for example, giving and subsequently retracting consent to process data) with the principal research question being to what extent the selected implementation languages AspectOWL and Prova are adequate for modeling the scenario, and what are the advantages and disadvantages.

The remainder of this paper is organized as follows. Section 2 discusses the related work. The use cases are described in Sect. 3. Section 4 introduces the languages Prova and AspectOWL, while Sect. 5 discusses the implementation in AspectOWL and Prova. Section 6 evaluates and compares the two approaches and finally, Sect. 7 concludes the paper and proposes future work.

2 Related Work

Palmirani et al. [14] introduce PrOnto, an ontology modeling the core GDPR concepts such as data types and documents, agents and roles, processing purposes, legal bases, processing operations, and deontic operations or modeling rights and duties. By integrating deontic logic, is allows for legal reasoning. Another GDPR-related ontology, emphasizing on consent is presented by Pandit et al. [15]. In addition to an OWL2-DL ontology for representation of consent and its associated information such as provenance, it also presents the methodology used in the creation and validation of the ontology as well as an example use-case demonstrating its applicability. Kurteva et al. [11] provide a survey of existing formal accounts (in terms of representational formalism) of the concept of consent. The survey includes PrOnto, and we opted for re-using PrOnto's conceptualization of consent in our work since it is the most comprehensive in terms of how the concept is related to other GDPR-related concepts. In [2], data usage policies, the consent of data subjects, and selected fragments of the GDPR are encoded in a fragment of OWL2 called $\mathcal{PL}$ (policy language); compliance checking and policy validation are reduced to subsumption checking and concept consistency checking. It proposes a tradeoff between the expressiveness requirements on $\mathcal{PL}$ posed by the modeling of the GDPR and its scalability requirements that dictate real-time compliance checking, achieved by a specialized reasoner.

Robaldo et al. [21] model the GDPR using LegalRuleML [13]. It is based on [14], and extends it by adding additional constraints in the form of if-then rules formalized in reified Input/Output logic [20], referring either to first order logic implications or to deontic statements. Robaldo [19] provides an executable implementation of the previous work by translating the rules into SHACL constraints.

A common characteristic of all abovementioned work is that it focuses on compliance checking in static situations and does not emphasize on modeling of a changing environment or transitions on the state of affairs, for example giving and revoking consent. Such a state transition is handled from an external resource which then resorts to the above systems for the compliance checking.

De Montety et al. [3] present a model of a core subset of the GDPR in Prolog and propose an architecture for a deontic-based compliance checker using rules that model both technique-oriented levels (modeled as state machines) and legal-oriented levels (modeled as property definitions). While their work is similar to ours, our work differentiates itself from it by providing two running proof-of-concept implementations.

Another work that also considers dynamic state transitions is De Vos et al. [4], the authors of which model the GDPR aspect using an ODRL template and then map it to answer set programming for (closed-world) semantics and reasoning. State transitions are represented in the form of fluents using the domain-specific action language InstAL.

3 Use Cases

In this section, we describe two data wallet use cases in terms of interaction sequences and data exchange between the different parties involved. The use cases revolve around data wallet owners that share personal data using relaying parties that provide specialized applications (a search application and a picture sharing application).

Use Case 1: Personalized Search (Fig. 1)

- Alice opens an account with an Identity Provider IdP_{Alice}. She provides consent for the IdP to store her *OpenID* and her *login credentials* for the purpose of *confirming her identity towards third parties*.
- Now Alice opens an account with a Data Wallet Provider DWP_{Alice}. She provides consent for the DWP to store data she uploads to the DWP along with her WebID document, which represents her online identity and contains a link to her OpenID (managed by her IdP).
- Alice seeks some information at the third-party app *SearchApp* (relying party, *RP*). She launches SearchApp and enters a search query. The app requests personal data about her previous search history from Alice's personal data wallet for personalization of the current search. The app also asks if Alice's data might be used for data analytics by SearchApp.
- Alice expresses her consent for both purposes by giving read permission for the requested data to SearchApp (identified by its WebID). SearchApp reads and stores the data for which Alice gave permission and derives an anonymized data set for data analytics purposes.
- Later Alice withdraws her consent to use the data for data analytics and personalization purposes by revoking the read permission.
- SearchApp may continue to use the derived (anonymized) data, but must delete the personal data it has obtained from Alice's data wallet. SearchApp is now denied to get updates from Alice's search history data from her data wallet.

Use Case 2: Sharing Pictured via a Wallet-Enabled Sharing App (Fig. 2)

- Alice decides to share a personal picture with her friends Bob and Cesar using PictureApp (relying party 2, *RP2*). She provides consent for PictureApp (identi-

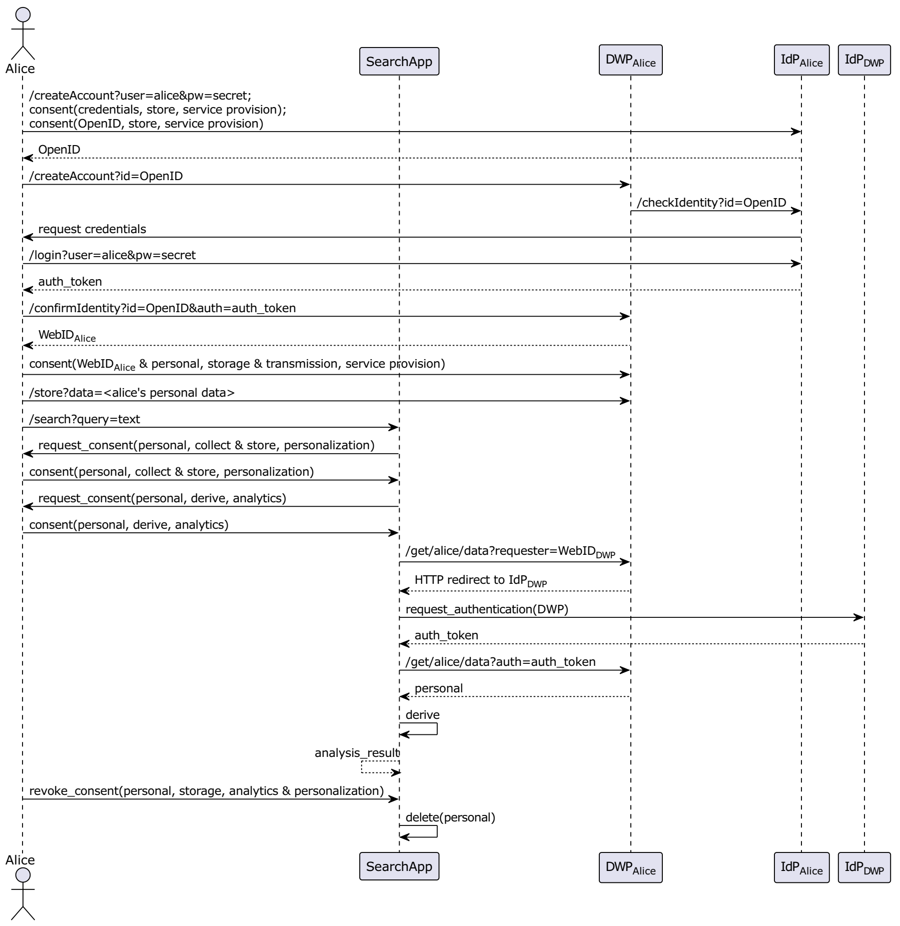
Alice
SearchApp
DWP$_{Alice}$
IdP$_{Alice}$
IdP$_{DWP}$
/createAccount?user=alice&pw=secret;
consent(credentials, store, service provision);
consent(OpenID, store, service provision)
OpenID
/createAccount?id=OpenID
/checkIdentity?id=OpenID
request credentials
/login?user=alice&pw=secret
auth_token
/confirmIdentity?id=OpenID&auth=auth_token
WebID$_{Alice}$
consent(WebID$_{Alice}$ & personal, storage & transmission, service provision)
/store?data=<alice's personal data>
/search?query=text
request_consent(personal, collect & store, personalization)
consent(personal, collect & store, personalization)
request_consent(personal, derive, analytics)
consent(personal, derive, analytics)
/get/alice/data?requester=WebID$_{DWP}$
HTTP redirect to IdP$_{DWP}$
request_authentication(DWP)
auth_token
/get/alice/data?auth=auth_token
personal
derive
analysis_result
revoke_consent(personal, storage, analytics & personalization)
delete(personal)
Alice
SearchApp
DWP$_{Alice}$
IdP$_{Alice}$
IdP$_{DWP}$

Fig. 1 First part of the use case: Alice creates an account with a DWP, provides consent for sharing personal data for the purposes of personalization and analysis, later revokes her consent for the purpose of analysis

fied by its WebID) to retrieve the picture from her data wallet and to make the picture available to her friends Bob and Cesar, both identified by their WebIDs.

- Later, Alice withdraws her consent to share the picture with Cesar by revoking read permission for Cesar. PictureApp is still permitted to store a copy of Alice's image, but has the obligation to deny Cesar access to the image.

4 AspectOWL and Prova Basics

This section provides an introduction into the two formalisms used for modeling the use cases presented in Sect. 3, focusing on the features that enable the implementation of the use cases.

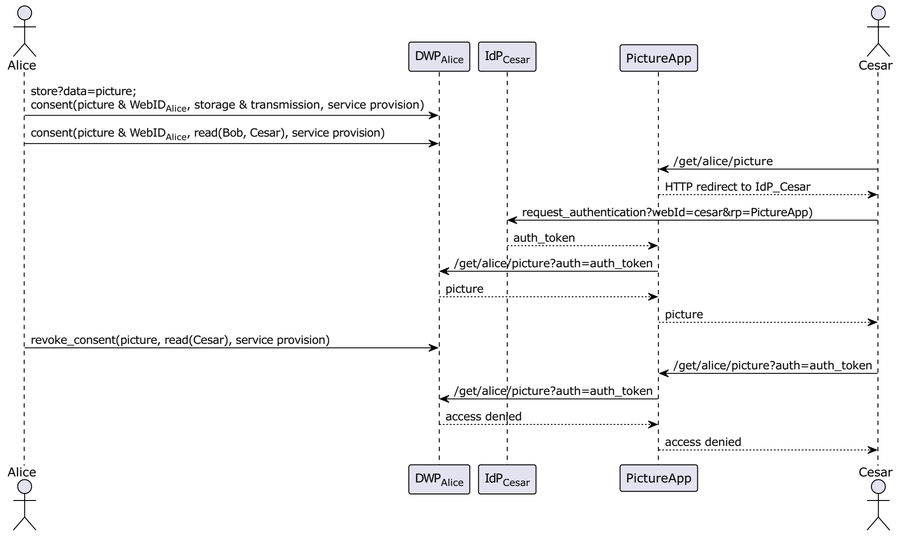
Alice
DWP_Alice
IdP_Cesar
PictureApp
Cesar
store?data=picture;
consent(picture & WebID_Alice, storage & transmission, service provision)
consent(picture & WebID_Alice, read(Bob, Cesar), service provision)
/get/alice/picture
HTTP redirect to IdP_Cesar
request_authentication?webId=cesar&rp=PictureApp)
auth_token
/get/alice/picture?auth=auth_token
picture
picture
revoke_consent(picture, read(Cesar), service provision)
/get/alice/picture?auth=auth_token
/get/alice/picture?auth=auth_token
access denied
access denied

Fig. 2 Second part of the use case: Alice shares a picture from her data wallet with Bob and Cesar. Cesar accesses her picture using a wallet-enabled sharing app. Later, Alice revokes permission from Cesar to access her picture. He and the sharing app can no longer access the picture

4.1 AspectOWL

AspectOWL [23] is an extension of the W3C OWL 2 ontology language[1] which permits the representation of contextualized knowledge by adding formal context descriptions (called *aspects*) to TBox, RBox, and ABox axioms of an OWL ontology.

AspectOWL is an instance of a general KR approach to the formalization of context, named *Aspect-Oriented Ontology Development (AOOD)* [22]. AOOD, in turn, is inspired by the Aspect-Oriented Programming paradigm [7], from which it lends most of its basic concepts and accompanying terminology.

Applied to KR formalisms, aspects can be used to convey context (e.g., temporal information) that restricts axiom validity. For this purpose, Aspect OWL introduces a new axiom type called *aspect assertion axiom*. An aspect assertion is a binary relation between an OWL axiom and an advice class expression (the context of the axiom). Syntactically, aspect assertion axioms resemble annotation assertion axioms. They differ from the latter in that they have a defined model theoretic semantics, which makes use of combined interpretations, which we call a $\mathcal{SROIQ}_{Kripke}$ interpretation.

Definition 1 A $\mathcal{SROIQ}_{Kripke}$ interpretation is a tuple $\mathcal{J} := (W, R, L, \cdot^{\mathcal{J}}, \Delta, (\cdot^{\mathcal{I}_w})_{w \in W})$ with W being a nonempty set, called *possible worlds*, and L a Kripke interpretation,

[1] https://www.w3.org/TR/owl2-overview/.

assigning truth values to propositional symbols in each world $w \in W$. For every $A \subseteq W$, $\mathcal{I}_A$ is a DL interpretation.

The semantics of an aspect of an axiom is then defined as follows:

Definition 2 Let $\mathcal{J} := (W, R, L, \cdot^{\mathcal{J}}, \Delta, (\cdot^{\mathcal{I}_w})_{w \in W})$ be a possible-world DL interpretation. We interpret an aspect under which an axiom α holds as follows: $(\mathsf{hasAspect}(\alpha, A))^{\mathcal{J}} \rightarrow A^{\mathcal{J}} \subseteq C^{\mathcal{J}} := \{w \in W \mid \mathcal{I}_w \models \alpha\}$. Because of the correspondence between Description Logics and Modal Logics [24] we can set $W = C^{\mathcal{J}}$, such that on the semantic level, each individual corresponds to a possible world. Furthermore, we set L such that $L(\alpha)^{\mathcal{J}} := A^{\mathcal{J}}$.

The modal part(s) of the multi-dimensional interpretation may be used to represent context of different modalities. For example, it permits to put an OWL axiom into a temporal or a deontic context (meaning that the axiom is valid only at a particular time or that the proposition represented by the axiom is, for example, obligatory). The kind of modality can be determined by the choosing the appropriate modal logic, which in turn is determined by the presence or absence of modal axioms. These in turn can be selected by altering the characteristics of the accessibility relation [1].

For a full description of the features and semantics of AspectOWL 2, see Schäfermeier and Paschke [23].[2]

4.2 Prova

Prova is both a (Semantic) Web rule language and a distributed (Semantic) Web rule engine. It supports reaction rule based workflows, event processing, and reactive agent programming. It integrates Java scripting with derivation and reaction rules, and message exchange with various communication frameworks [8, 10, 16].

Syntactically, Prova builds upon the ISO Prolog syntax and extends it, notably with the integration of Java objects, typed variables, F-Logic-style slots, and SPARQL and SQL queries [16]. Slotted terms in Prova are implemented using the arrow expression syntax '`->`' as in RIF and RuleML, and can be used as sole arguments of predicates. They correspond to a Java HashMap, with the keys limited to Stings [9, 16].

Semantically, Prova provides the expressiveness of serial Horn logic with a linear resolution for extended logic programs (SLE resolution) [17], extending the linear SLDNF resolution with goal memoization and loop prevention. Negation as failure support in the rule body can be added to a KB by implementing it using the cut-fail test as follows:

[2] For an overview over the complete abstract syntax of Aspect OWL 2, see http://www.aspectowl.xyz/syntax/.

```
not(A) :- derive(A), !, fail().
not(_).
```

Notice the Prova syntax for `fail` that requires parentheses, as well as the built-in meta-predicate `derive` that allows to define (sub) goals dynamically with the predicate symbol unknown until run-time [16].

Prova's reactive agents are instances of a running rulebases that include message passing primitives. These built-in primitives are the predicates `sendMsg/5`, `rcvMsg/5`, as well as their variants `sendMsgSync/5`, `rcvMult/5`. The position-based arguments for the above predicates are [9]: (1) *XID* — conversation id of the message, (2) *Protocol* — name of the message passing protocol, (3) *Destination* or *Sender* — the agent name of the receiver/sender, (4) *Performative* — the message type characterizing the meaning of the message, and (5) *Payload* — a Prova list containing the actual content of the message.

Prova defines the Java interface `ProvaService` and its default implementation `ProvaServiceImpl` that allows for a runner Java class — depending on the modularization (mapping each agent to a separate bundle vs. multiple agents in a bundle) — to embed one or more agents communicating with each other via messaging. The fundamental method is the method `send`, as follows:

```
send(String xid, String destination, String sender,
     String performative, Object payload, EPService callback)
```

The arguments have a direct correspondence with the message passing primitives, while `EPService` is a superclass of the `ProvaService` interface. Also, the message passing protocol is selected automatically.

Prova implements an inference extension called literal *guards*, specified using brackets. Using guards, we can ensure that during unification, even if the target rule matches the source literal, further evaluation is delayed unless a guard condition evaluates to true. Guards can include arbitrary lists of Prova literals including Java calls, arithmetic expressions, relations, and even the cut operator. Prova guards play even a more important role in message and event processing as they allow the received messages to be examined before they are irrevocably accepted. The guards are tested right after pattern matching but before a message is fully accepted, so that the net effect of the guard is to serve as an extension of pattern matching for literals [9].

5 Implementation

This section discusses the implementation of the use cases presented in Sect. 3 using the two formalisms introduced in Sect. 4. For AspectOWL, we emphasize the conceptualization of the legal domain and the representation of deontic context and state transitions following events with a temporal extension, while for Prova, we emphasize the different parties' interaction and the subsequent knowledge base updates.

5.1 AspectOWL

As OWL is a monotonic, declarative knowledge representation formalism it is suited for representing the static aspects of the domain under consideration. With AspectOWL, however, it is also possible to represent dynamic behavior: Different states of the universe may be represented by different contexts (in the form of OWL aspects) in which certain axioms hold respectively. The transition between states may be represented in terms of events that happen at a certain point in time with the contexts representing two subsequent states having a temporal extension either before or after the point in time.

Furthermore, AspectOWL permits the application of deontic modalities to OWL axioms. Since nesting of aspects is also allowed, it is possible to combine the two and represent dynamic change of deontic modalities.

The goal of the implementation is to model the GDPR-related actions and states of affair (and the transition between states) that can occur in our use cases. A significant amount of GDPR-related concepts could be imported from the current publicly available version of the PrOnto ontology [14], which was selected as it is, at the time of writing this article, the most comprehensive formal representation of the GDPR in OWL. PrOnto makes extensive use of ontology design patterns (ODPs), and so did we whenever applicable. Links to ODPs used in this work are provided in the footnotes.

Static Part

Data Data is the central concept of the GDPR domain around which everything else revolves. We reuse the data concept hierarchy from the PrOnto ontology [14], which makes Data a subclass of $\mathsf{InformationObject}$, which in turn is a subclass of the class $\mathsf{FRBRWork}$ from the Functional Requirements for Bibliographic Records (FRBR) vocabulary.

1. $\mathsf{Data} \sqsubseteq \mathsf{InformationObject}$
2. $\mathsf{InformationObject} \sqsubseteq \mathsf{akn} : \mathsf{FRBRWork}$[3]
3. $\mathsf{akn} : \mathsf{FRBRWork} \sqsubseteq \mathsf{owl} : \mathsf{Thing}$

Ownership of Data The GDPR is concerned about usage of data by different agents. Agents may either be human persons or non-human organizations.

4. $\mathsf{Person(Bob)}$
5. $\mathsf{ownedBy(dataHabit, Bob)}$

[3] {akn: {Akoma Ntoso XML for parliamentary, legislative \& judiciary documents (OASIS)} FRBR: {Functional Requirements for Bibliographic Records (IFLA)}}.

Agent Roles Agents, i.e., both persons and organizations, may assume roles as defined by the GDPR, namely the role of the data subject, the data controller, and the data processor. The same agent may assume several of these roles at the same time, e.g., a company may be both data controller and data processor.

6. $\exists \mathsf{hasRole.DataSubject(Bob)}$
7. $\exists \mathsf{hasRole.Controller(CompanyA)}$
8. $\exists \mathsf{hasRole.Processor(CompanyB)}$

Providing Consent The concept of concent is re-used from the PrOnto ontology, which models Concent as as subclass of Contract. We additionally add a concept of an action (ConcentAction) that creates such a contract. The action of creating a consent contract involves two participants, namely the data subject (the entity that gives consent) and the organization acting as the data controller/processor.

9. $\mathsf{ConsentAction} \sqsubseteq \mathsf{te:Action}$[4]
10. $\mathsf{ConsentAction(ca)}$
11. $\mathsf{pwo:happened(ca, t1)}$[5]
12. $\mathsf{pwo:produces(ca, co), Consent(co), Consent} \sqsubseteq \mathsf{Contract}$
13. $\mathsf{allowsAction(co, ac)}$
14. $\mathsf{bpe:actionHasParticipant(ac, org)}$[6]
15. $\mathsf{hasSubject(ac, data)}$

Data Processing Purpose As mandated by the GDPR, user consent for the processing of personal data must be explicitly given for a specific purpose and is only valid for that particular purpose.

16. $\mathsf{Purpose} \sqsubseteq \mathsf{owl:Thing}$
17. $\mathsf{Advertising} \sqsubseteq \mathsf{Purpose}$
18. $\mathsf{Marketing} \sqsubseteq \mathsf{Purpose}$
19. $\mathsf{Optimisation} \sqsubseteq \mathsf{Purpose}$
20. $\mathsf{allowedPurpose} \sqsubseteq \mathsf{owl:topObjectProperty}$
21. $\mathsf{allowedPurpose(dataHabit, advertising)}$

Data Processing Action For the conceptualization of data processing actions we re-use the existing concept Action from the PrOnto ontology along with a number of ontology design patterns. PrOnto defines actions as parts of workflows and relies on the Basic Plan Execution design pattern for doing so. The latter distinguishes between abstract workflow descriptions (which are abstract plans) and their concrete executions. Consequently, a workflow description may have arbitrarily many workflow execution instantiations, which in turn may involve arbitrarily many actions.

[4] http://www.ontologydesignpatterns.org/cp/owl/taskexecution.owl.
[5] http://purl.org/spar/pwo/.
[6] http://www.ontologydesignpatterns.org/cp/owl/basicplanexecution.owl.

22. $\mathsf{Analysis} \sqsubseteq \mathsf{DataProcessing}$
23. $\mathsf{DataProcessing} \sqsubseteq \mathsf{Workflow}$
24. $\mathsf{Workflow} \sqsubseteq \mathsf{Plan}$
25. $\mathsf{Plan} \sqsubseteq \mathsf{Description}$
26. $\mathsf{Workflow} \sqsubseteq \forall \mathsf{isSatisfiedBy.bpe : PlanExecution}$[7]

A concrete execution of an abstract workflow is represented as follows:

27. $\mathsf{pwo : WorkflowExecution} \sqsubseteq \mathsf{bpe : PlanExecution}$
28. $\mathsf{pwo : WorkflowExecution} \sqsubseteq \mathsf{tis : TimeIndexedSituation}$[8]
29. $\mathsf{bpe : PlanExecution} \sqsubseteq \mathsf{sit : Situation}$
30. $\mathsf{tis : TimeIndexedSituation} \sqsubseteq \mathsf{sit : Situation}$

And finally, an action is part of a workflow execution:

31. $\mathsf{pwo : WorkflowExecution} \sqsubseteq \exists \mathsf{pwo : involvesAction.(pwo : Action} \sqcap \exists \mathsf{te : executesTask.pwo : Step)}$

Dynamic Part

In this section, we demonstrate how dynamic processes (in terms of transitions between different states of the universe over time, usually in succession of an event) can be represented using AspectOWL. Providing consent for the processing of data by a third party leads to a transition between two states; from one in which the processing is not permitted to one where it is. It is possible to represent the two different states using two OWL aspects, each representing one state. As the transition is triggered by an event that happened at a certain point in time $\mathsf{T\_DC\_1}$ the states may also be represented as temporal contexts, the boundaries of both of which coincide at $\mathsf{T\_DC\_1}$.

32. $\mathsf{StateAspect1} \sqsubseteq \mathsf{aot : TemporalAspect} \sqcap \mathsf{time : before.\{T\_DC\_1\}}$[9]
33. $\mathsf{StateAspect2} \sqsubseteq \mathsf{aot : TemporalAspect} \sqcap (\{\mathsf{T\_DC\_1}\} \sqcup \mathsf{time : after.\{T\_DC\_1\}})$

As it is not possible for the same thing to be permitted and prohibited at the same time, it must be made sure that the two aspects representing the states are disjoint.

34. $\mathsf{StateAspect1} \sqcap \mathsf{StateAspect2} \sqsubseteq \mathsf{owl : Nothing}$

The action that is allowed in the state after providing consent is represented by a simple object property assertion:

7 http://www.ontologydesignpatterns.org/cp/owl/timeindexedsituation.owl.
8 http://www.ontologydesignpatterns.org/cp/owl/situation.owl.
9 https://ontology.aspectowl.xyz/temporal#.

35. $\mathsf{processesData(o1, data)}$, where
36. $\mathsf{Organisation(o1)} \wedge \mathsf{Data(data)}$

However, the aspect cannot be directly applied to the above assertion axiom since this would mean that starting from time point T_DC_1 the organization o1 processes data, while what we want to represent is the fact that o1 *is permitted* to process data starting at T_DC_1.

Representing the permission involves the creation of a further aspect of the type deontic aspect.

37. $\mathsf{PermissionAspect} \sqsubseteq \mathsf{aod:DeonticAspect} \sqcap \exists \mathsf{aod:legallyAccepts.aod:Reality}$[10]

Application of the deontic aspect to the assertion and the nesting of the resulting aspect assertion into the temporal aspect StateAspect2 yield the representation of the statement that o1 is allowed to process data starting at T_DC_1.

38. $\mathsf{Aspect(StateAspect2, Aspect(PermissionAspect, processesData(o1, data)))}$

To derive statements of this kind automatically when an assertion of consent is encountered in the KB, the following SWRL rule can be employed.

```
-:
ProvideConsent(?ca), pwo:happened(?ca, ?t1),
actedBy(?ca, ?ds), pwo:produces(?ca, ?co),
Consent(?co), allowsAction(?co, ?ac),
bpe:actionHasParticipant(?ac, ?org),
hasSubject(?ac, ?data),
aspectswrl:createOPA(collectsDataFrom, ?org, ?ds, ?a),
aspectswrl:temporal(?a, time:after, ?t1, true),
aspectswrl:deontic(?perm, legallyAccepts, aod:Reality),
aspectswrl:nest(?perm, ?a)
```

It uses multiple AspectSWRL built-ins, namely, `aspectswrl:createOPA`, which creates the OWL object property assertion and wraps it in an aspect bound to variable `?a`, and `aspectswrl:temporal` which creates the temporal aspect. The first parameter binds the resulting aspect to ?a. The second parameter determines the accessibility relation used, in this case time:after. The third parameter determines the time individual used in conjunction with the accessibility relation, which we set to the value of the variable ?t1 and which corresponds to the object in the pso:happened predicate. The fourth parameter is a Boolean determining whether the individual should be included in the interval defining the aspect or not. In this case, we want ?t1 to be included in the interval. aspectswrl:deontic works similarly with the parameters being the variable to which the resulting aspect should be bound, the accessibility relation, and the individual representing reality.

[10] https://ontology.aspectowl.xyz/deontic#.

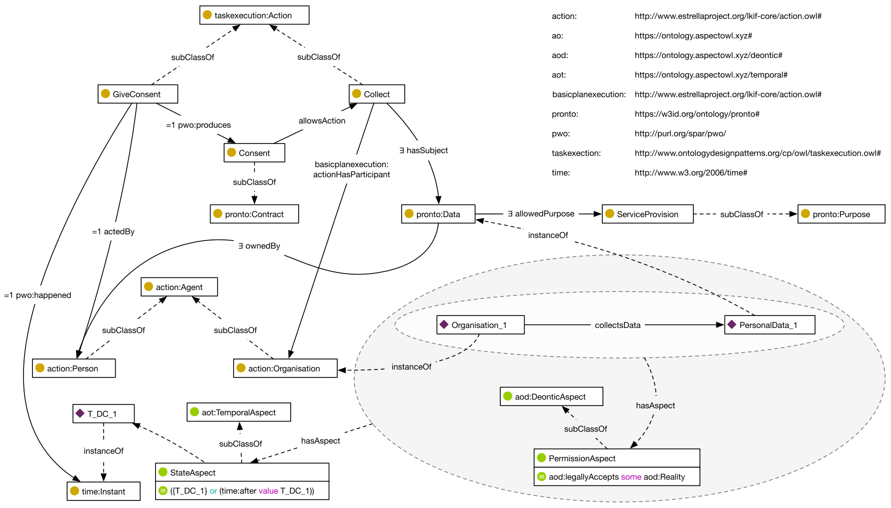
taskexecution:Action
subClassOf
subClassOf
GiveConsent
Collect
=1 pwo:produces
allowsAction
Consent
∃ hasSubject
basicplanexecution: actionHasParticipant
subClassOf
pronto:Contract
pronto:Data
∃ allowedPurpose
ServiceProvision
subClassOf
pronto:Purpose
instanceOf
=1 actedBy
∃ ownedBy
=1 pwo:happened
action:Agent
subClassOf
subClassOf
Organisation_1
collectsData
PersonalData_1
action:Person
action:Organisation
instanceOf
aod:DeonticAspect
hasAspect
T_DC_1
aot:TemporalAspect
subClassOf
instanceOf
subClassOf
hasAspect
PermissionAspect
aod:legallyAccepts some aod:Reality
time:Instant
StateAspect
({T_DC_1} or (time:after value T_DC_1))
action: http://www.estrellaproject.org/lkif-core/action.owl#
ao: https://ontology.aspectowl.xyz#
aod: https://ontology.aspectowl.xyz/deontic#
aot: https://ontology.aspectowl.xyz/temporal#
basicplanexecution: http://www.estrellaproject.org/lkif-core/action.owl#
pronto: https://w3id.org/ontology/pronto#
pwo: http://purl.org/spar/pwo/
taskexection: http://www.ontologydesignpatterns.org/cp/owl/taskexecution.owl#
time: http://www.w3.org/2006/time#

Fig. 3 An excerpt from the RECOMP AspectOWL ontology

aspectswrl:nest takes two aspects as parameters and results in a nesting of the first into the second.

When the user revokes their consent, an instance of the class RevokeConsent is created, having the same properties as the ProvideConsent instance. A second SWRL rule, similar to the one above, with the exception that it contains Revoke-Consent instead of ProvideConsent in the antecedent and legallyProhibits instead of legallyAccepts then creates the new temporal context in which the former modality of permission of the data processing is replaced by a prohibition modality. Figure 3 provides an overview of the ontology and the aspects created.[11]

5.2 Prova

The Prova implementation[12] uses the message passing primitives mentioned in Sect. 4. All parties are represented as agents that communicate via message passing. All actions of the workflow are initiated by messages sent by the Java runner class to the appropriate agent.

To demonstrate the Prova implementation, we focus on the first part of the use case presented in Sect. 3, where Alice (represented by the agent `alice`) performs a web search. The SearchApp (represented by the agent `searchApp`) requests consent for personal data access and delivers a personalized search result if the consent was provided, or a non-personalized result otherwise. Delivering a personalized search also requires prior login.

[11] The ontology files are available at https://github.com/RalphBln/recomp-use-cases.

[12] The Prova implementation is available at https://github.com/tmitsi/recomp-usecases.

The actions of `alice` are controlled from Java, essentially creating a script with the messages that `alice` receives, which in turn are forwarded to other agents. Therefore, initially we pass the following message to `alice` from Java:

```
payload.put("agent","searchApp");
payload.put("operation","search");
payload.put("webID","alice.example.com");
payload.put("query","travel suggestions");
payload.put("dwp","dwp");
service.send("xid","alice","javaRunner","request",payload,this);
```

where `payload` is a Java HashMap, with its elements corresponding to the slot names and fillers. These are the agent performing the search, the operation, `alice`'s WebID, the search query, and the agent serving as a data wallet provider. The above message is captured by the following inline reaction rule [9]:

```
alice() :-
    rcvMult(XID,P,From,request,
        ↪ {agent->A,operation->Op,webID->W,query->Q,dwp->DWP}),
    sendMsg(XID,P,A,request,
        ↪ {operation->Op,webID->W,query->Q,dwp->DWP}).
```

This rule instructs that upon receiving a message of this pattern, `alice` will send a message to the appropriate agent (e.g., to the agent `searchApp`) requesting a specific operation (e.g., `search`). The `searchApp` side is implemented using Prova guards. The message is captured by variants of the inline reaction rule that correspond to different possible states. For example, if the user is already logged-in and has provided consent, the rule is as follows:

```
searchApp() :-
    rcvMult(XID,P,F,request,
        ↪ {operation->search,webID->W,query->Q,dwp->D}
        ↪ [loggedIn(F,W,D),consent(W,personal,personalization)],
    searchHelper(W,Q,F,Result,yes),
    sendMsgSync(XID,P,F,inform,{msg->Result}).
```

The top-level `searchApp()` first captures messages having the payload pattern `operation->search, webID->W, query->Q, dwp->D`, and accepts them if the guard `[loggedIn(From,W,D), consent(W,personal,personalization)]` succeeds. If a message is accepted, Prova proceeds with the evaluation of `searchHelper` binding the results of the search to the variable `Result`, and then messages these results back to `alice`.

If the user is already logged in but did not provided consent to share personal data for personalized results, the following rule is selected instead:

```
searchApp() :-
    rcvMult(XID,P,F,request,
        ↪ {operation->search,webID->W,query->Q,dwp->D})
        ↪ [loggedIn(F,W,D),alreadyAsked(W,personal,personalization),
        ↪ not(consent(W,personal,personalization))],
    searchHelper(W,Q,F,Result,no),
    sendMsgSync(XID,P,F,inform,{msg->Result}).
```

Notice that both the above rules are capturing the same payload pattern, but differ depending on the current state, i.e., if the guard succeeds or not. In both of the above variants, the use of guards `[loggedIn(F,W,D),...]` is straightforward. However, modeling the state where the user logged in but has not been asked yet for the consent choice reveals interesting properties when using guards in message processing:

```
searchApp() :-
    rcvMult(XID,P,F,request,
        ↪ {operation->search,webID->W,query->Q,dwp->D})
        ↪ [loggedIn(F,W,D),
        ↪ not(alreadyAsked(W,personal,personalization))],
    Msg = "searchApp asks for consent... personalized results",
    sendMsg(XID,P,F,input_request,
        ↪ {operation->consent,data->personal,
        ↪ purpose->personalization,msg->Msg}),
    rcvMsg(XID,P,F,response,{answer->In}),
    assert(alreadyAsked(W,personal,personalization)),
    searchHelper(W,Q,F,Result,In),
    sendMsgSync(XID,P,F,inform,{msg->Result}).
```

This case involves capturing the same payload pattern, messaging back to `alice`, asserting her choice, and calling the `searchHelper` predicate that either performs an non-personalized search or a personalized search (possibly also asserting the user's consent).

Notice the absence of the cut operator after the assertion. Without using guards in message processing, the assertion would make `alreadyAsked/3` and possibly `consent/3` evaluate to true, enabling the evaluation of either the previous cases as the payload pattern is the same for all three cases. To prevent this, the usage of cut operator after each assertion would be necessary. Such an implementation could be as follows:

```
searchApp() :-
    rcvMult(XID,P,F,request,
        ↪ {operation->search,webID->W,query->Q,dwp->D}),
    loggedIn(F,W,D),
    not(alreadyAsked(W,personal,personalization)),
    ...
    rcvMsg(XID,P,F,response,{answer->In}),
    assert(alreadyAsked(W,personal,personalization)),
    !,
    searchHelper(W,Q,F,Result,In),
    sendMsgSync(XID,P,F,inform,{msg->Result}).
```

All similar cuts would be "red cuts", as their removal would affect the execution and results [25]. Using guards, which act as additional pre-conditional constraints on the literal [18], we avoid red cuts as the messages are fully accepted only if the guard evaluates to true, effectively creating an early selection of which messages are to be accepted, thus eliminating the need of cuts.

A similar rule exists to cover the case where the user is not logged in, where `searchApp` performs an non-personalized search and informs the user that by logging-in personalized results can be shown. Finally, the evaluation of the helper predicate `searchHelper(WebID,Q,From,Result,DWP,In)` binds a mock-up of the search results in the variable `Result`, while taking into account the consent (or, the lack of it) of the user.

6 Evaluation and Comparison of KR Approaches

In what follows, we give a requirements-based evaluation of the two approaches described in the previous sections. We evaluate and compare the two approaches within the requirements framework established in [5]. The choice of this particular evaluation framework was guided by the fact that it has been applied for the evaluation of LegalRuleML, which is the most comprehensive legal modeling language. It defines five evaluation criteria for legal rule systems that are also applicable to more general KR-based systems for the legal domain related to GDPR knowledge representation. We omit the last criterion ("the approach must be scalable and have the capability to handle any type of document"), as the use cases pertain to a restricted subset of the legislation.

C1: The representation must be both human and machine-readable and manageable independently from any system that uses the representation

AspectOWL ontologies can be read and edited by humans using the Protégé ontology editor[13] with the AspectOWL[14] and AspectSWRL[15] plug-ins installed.

[13] https://protege.stanford.edu.

[14] https://github.com/RalphBln/aspect-owl-protege.

[15] https://github.com/RalphBln/aspect-swrl-builtins.

They can be exported to various machine-readable formats, including the AspectOWL Functional-Style Syntax,[16] an extension of the OWL Functional-Style Syntax[17] and every existing RDF[18] serialization format that supports RDF statement reification. The Functional-Style Syntax is sufficiently human-readable and can be manually edited in a simple text editor. The various RDF serialization formats can be processed in RDF editors as well as simple text editors. Additionally, AspectOWL inherits the identifier paradigm from OWL, which means that every AspectOWL ontology (as well as every entity defined in it) has a unique identifier in the form of an Internationalized Resource Identifier (IRI).

Prova is a high-level language and in addition Prova programs can be considered executable specifications as it is syntactically based on ISO Prolog [25].

Since all AspectOWL ontologies (independently of the serialization format) and Prova code are stored in plain text files, all file management paradigms are applicable, such as naming, storing in folders, and versioning.

C2: There must be a close link between the digital model and the paper-based source document to enable automatic version control. This also maintains user familiarity with the structure and literal content of the source document, which is important to promote its adoption in the conventional practice

Both implementations revolve around the concepts of consent, the right to data portability, and access control. While these concepts are basic on GDPR, the presented use-cases are not exclusive to GDPR and can be compliant with other current data-privacy regulations that utilize these concepts.

However, version control in AspectOWL is available on the syntactic level by the OWL ontology version IRI (which gives each version of an ontology its own specific IRI). AspectOWL extends the simple syntactic versioning facility on the ontology level by semantic versioning on the axiom level. Since aspects as a syntactic primitive of the AspectOWL language are also entities they also carry an IRI as their identifier and can thereby be linked to from external systems, such as versioning systems. The Prova implementation could be extended to include IRIs as namespaces to facilitate such version control by encoding a particular source document version in the namespace.

C3: Availability of practical authoring tools to support the development work An authoring tool suite in the form of plug-ins for the popular Protégé ontology editor is available for AspectOWL. The extension allows users to create aspects, edit their properties and attach them directly to axioms. It also provides several ways to define sets of axioms (using different pointcut languages based on either DL-Queries, SPARQL queries, signature definitions or AspectSWRL rules) and attach aspects to the entire axiom set at once. A variety of import and export options from and to standard formats exist. A special AspectOWL reasoner is also part of the AspectOWL plug-in, which allows for ad-hoc inference checking under the AspectOWL semantics during the authoring process.

[16] https://aspectowl.xyz/syntax/index.xhtml.

[17] https://www.w3.org/TR/owl2-syntax/.

[18] https://www.w3.org/TR/rdf11-concepts/.

The development of the Prova implementation can be supported by various Java-targeting IDEs that can also support Prova's Prolog-based syntax. As Prova is available as a library in public repositories (e.g., Maven), integrating and executing Prova is possible from within a modern Java-targeting IDE.

C4: The representation must be based on an open standard technology that promotes interoperability, and supports open standard query languages

AspectOWL is an extension of OWL 2, which is a W3C standard.[19] AspectOWL can be serialized to standard formats, such as the various serialization formats of RDF. The resulting RDF graphs can be queried using standard RDF query languages, such as SPARQL.[20] Pointcut selection (the selection of sets of axioms to which an aspect is supposed to be applied) can be performed using SPARQL queries, among others.

Prova is open-source (Java also has open-source implementations) and syntactically is based on ISO Prolog. Prova code could be translated to the open-standard Reaction RuleML, a rule interchange format tailored for agents running as distributed inference services and supporting distributed event/action processing. Prova also provides built-ins for rule-based data access such as XML (DOM), SQL, RDF, XQuery and SPARQL [18].

In general, the two approaches used in this paper differ in their intended application scopes and hence their syntax, semantics, and expressiveness. However, interestingly, both could be used to represent a significant part of the presented use case.

The agent-based Prova implementation with the message passing primitives, and reactive rules combined with assertions and retractions were adequate to model was able to model all key workflow elements. Moreover, it was possible to simulate real-world practices such as storing hashed passwords instead of plain-text, and session cookies to facilitate logins.

This is out of the scope of static knowledge-representation formalisms such as OWL, as the latter lacks the necessary interaction primitives with the outside world, such as reaction rules or data stream processing facilities. It is, however, conceivable to employ some sort of dynamically updating ABox and let our AspectSWRL rules run on new ABox axioms as they are added to the KB.

An obvious difference between Prova and OWL is that the latter operates in a strictly monotonic fashion without any notion of knowledge retraction. AspectOWL is able to circumvent this to a certain extent since its ability to create contexts which restrict the validity of exiting axioms introduces a way of mimicking non-monotonicity. However, AspectOWL is still a monotonic formalism, and while knowledge may be retracted from the global scope by restricting it to a local context, the context itself (containing the knowledge) can never be retracted. In other words, the history of emerging and disappearing knowledge can never be erased in AspectOWL. This can be regarded either as an advantage or as a disadvantage, depending on the application requirements.

[19] https://www.w3.org/TR/owl2-overview/.

[20] https://www.w3.org/TR/sparql11-query/.

Prova, OWL, and AspectOWL permit the choice of the level of expressiveness and the selection of semantics. OWL 2, for example, introduced different profiles, such as OWL 2 DL, OWL 2 EL, OWL 2 RL, and OWL 2 QL. OWL 2 semantics correspond to the semantics of description logics and the set of language primitives used determines the particular description logic in which an OWL ontology can be expressed. Prova and AspectOWL permit the selection of semantic profiles, which, in the case of AspectOWL, determines the model-theoretic semantics under which a theory is interpreted. These choices lead to different computational properties of each of the formalisms.

AspectOWL, being a static KR formalism, requires recomputation of all inferences as facts are added to the KB. Incremental reasoning has not been implemented but might be in the future. In the case of the use cases presented in this paper, the resulting AspectOWL ontology has an expressiveness that corresponds to the description logic $\mathcal{ALCROIQ}$, which means that the problems of concept satisfiability and consistency checking are NExpTime-hard. Description logics are by design decidable, but AspectOWL, the semantics of which divert from pure DLs, are not guaranteed to be decidable. The non-decidability comes from the multi-dimensional interpretation. Since, however, in the underlying use case there is no interaction between the different context levels (the object and the temporal level have both access to the time individual T_DC_1, but this individual is rigid, i.e., its interpretation is context-independent), the ontology is decidable even under the AspectOWL full semantics. Prova is a combination of rule and scripting language. The semantics of the rule language correspond to Prolog, which makes Prova generally undecidable. Prova extends the declarative rule language by procedural attachments, a mechanism for making calls to procedures written in an imperative language, such as Java, which makes it impossible to make a generalized statement about the computational properties of the Prova system as a whole. Prova's messaging system, of which we primarily make use in the context of this research, is pattern-based with selectable inference regimes, such as DL reasoning. Moreover, the use of guards in message processing which act as additional pre-conditional constraints reduces the need for red cuts after positive (assertions) or negative (retractions) KB updates. This also helps to clearly distinguish the different contexts under which a KB update is allowed.

A clear advantage of AspectOWL is its full backwards-compatibility with OWL 2 and the resulting ability to import and reuse existing knowledge from the many OWL ontologies that are publicly available as we did with PrOnto and the ontology design patterns, while the Prova implementation was basically built from scratch. In the context of the data wallet scenario existing standards such as Solid use RDF as their data model and are thereby directly compatible with AspectOWL.

7 Conclusions and Future Work

We described a GDPR-related use case for a distributed data wallet. The use case defines all typical stakeholders, namely an Identity Provider, a Data Wallet Provider, Relaying Parties as applications (a PictureApp and a SearchApp), and users. The main concepts of the use case are data access (from Relaying Parties or from users), and consent-giving actions that enable this data sharing. Depending on providing consent or consent revoking actions, different possible states can emerge, rendering the use case non-monotonic.

We provided two KR implementations, one in AspectOWL, one in Prova. Both approaches were able to result in a representation of the problem domain which is sufficiently adequate for GDPR-related inference tasks, especially state transitions resulting from actions such as a user providing consent. Prova with its reaction rule messaging system and procedural attachments is directly capable of implementing the workflow of our given use cases.

Specifically, the Prova implementation is able to model the interaction of all parties (represented as agents) in real-time. The reactive messaging capabilities, the Java object support and the non-monotonic state transition semantics can model all key states of the use case. The AspectOWL implementation provides both the ontology with types and description of the domain concepts, and the state transition modeling required by the use case. However, since AspectOWL is a monotonic formalism the representation of non-monotonic states using contexts may lead to an indefinite growth of the KB.

While AspectOWL proved to be a suitable approach for the specification of the ontological domain model Prova is the more practical approach for a real-world application including the interaction between involved parties and the state transition workflows. Therefore, the two implementations are complementing each other.

Future work may consist in combining the two approaches, by integrating and reusing the existing AspectOWL ontology in Prova, combining the strengths of the two systems. This combined system may serve as a real-world back-end of an ecosystem of a distributed data wallet and applications.

It would also be interesting to implement the use cases using different approaches, for example the SHACL-based approach by Robaldo [19].

Funding Open access funding provided by HEAL-Link Greece. This work has been partially funded by the German Federal Ministry of Education and Research (BMBF) through the project Panqura (03COV3F) and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) project RECOMP (DFG-GZ: PA 1820/5-1).

Data availability We do not analyse or generate any data sets. The source code of the implementation can be obtained from https://github.com/tmitsi/recomp-usecases and https://github.com/RalphBln/recomp-use-cases.

Declarations

Conflict of Interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Blackburn, P., Benthem, J., & Wolter, F. (2006). Handbook of modal logic. In: *Studies in logic and practical reasoning*, vol 3. New York: Elsevier Science Inc.
2. Bonatti, P. A., Ioffredo, L., Petrova, I. M., Sauro, L., & Siahaan, I. R. (2020). Real-time reasoning in OWL2 for GDPR compliance. *Artificial Intelligence, 289*, 103389. https://doi.org/10.1016/j.artint.2020.103389.
3. De Montety, C., Antignac, T., & Slim, C. (2019). GDPR modelling for log-based compliance checking. In: *Trust Management XIII: 13th IFIP WG 11.11 International Conference, IFIPTM 2019*, Copenhagen, Denmark, July 17–19, 2019, Proceedings 13, Springer, pp 1–18.
4. De Vos, M., Kirrane, S., Padget, J., & Satoh, K. (2019). ODRL policy modelling and compliance checking. In P. Fodor, M. Montali, D. Calvanese, & D. Roman (Eds.), *Rules and reasoning* (pp. 36–51). Cham: Springer International Publishing.
5. Dimyadi, J., Governatori, G., & Amor, R. (2017). Evaluating LegalDocML and LegalRuleML as a standard for sharing normative Information in the AEC/FM Domain. In: *Lean and Computing in Construction Congress—Volume 1: Proceedings of the Joint Conference on Computing in Construction*, Heriot-Watt University, Heraklion, Crete, Greece, pp. 637–644. https://doi.org/10.24928/JC3-2017/0012.
6. European Commission. (2016). Regulation (EU) 2016/679 of the European Parliament and of the Council. http://data.europa.eu/eli/reg/2016/679/oj.
7. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J. M., & Irwin, J. (1997). Aspect-oriented programming. In M. Aksit & S. Matsuoka (Eds.), *Lecture Notes in Computer Science, ECOOP'97—object-oriented programming* (Vol. 1241, pp. 220–242). Berlin: Springer.
8. Kober, G., Robaldo, L., & Paschke, A. (2022). Modeling medical guidelines by Prova and SHACL accessing FHIR/RDF. Use case: the medical ABCDE approach. In: dHealth 2022, IOS Press, pp 59–66.
9. Kozlenkov, A. (2010). Prova rule language version 3.0 user's guide. https://github.com/prova/prova/tree/master/doc.
10. Kozlenkov, A., Penaloza, R., Nigam, V., Royer, L., Dawelbait, G., & Schroeder, M. (2006). Prova: rule-based Java scripting for distributed web applications: a case study in bioinformatics. In T. Grust, H. Höpfner, A. Illarramendi, S. Jablonski, M. Mesiti, S. Müller, P. L. Patranjan, K. U. Sattler, M. Spiliopoulou, & J. Wijsen (Eds.), *Current trends in database technology—EDBT 2006* (pp. 899–908). Heidelberg: Springer.
11. Kurteva, A., Chhetri, T.R., Pandit, H.J., & Fensel, A. (2021). Consent through the lens of semantics: state of the art survey and best practices. Semantic Web Preprint, pp. 1–27. https://doi.org/10.3233/SW-210438.
12. Mansour, E., Sambra, A.V., Hawke, S., Zereba, M., Capadisli, S., Ghanem, A., Aboulnaga, A., & Berners-Lee, T. (2016). A demonstration of the Solid platform for social web applications. In: *Proceedings of the 25th International Conference Companion on World Wide Web, International World Wide Web Conferences Steering Committee*, Republic and Canton of Geneva, CHE, WWW '16 Companion, pp. 223–226. https://doi.org/10.1145/2872518.2890529.
13. Palmirani, M., Governatori, G., Rotolo, A., Tabet, S., Boley, H., & Paschke, A. (2011). LegalRuleML: XML-based rules and norms. In F. Olken, M. Palmirani, & D. Sottara (Eds.), *Rule-based modeling and computing on the semantic web* (pp. 298–312). Heidelberg: Springer.
14. Palmirani, M., Martoni, M., Rossi, A., Bartolini, C., & Robaldo, L. (2018). PrOnto: privacy ontology for legal reasoning. In A. Kő & E. Francesconi (Eds.), *Electronic government and the information systems perspective* (pp. 139–152). Cham: Springer International Publishing.
15. Pandit, H. J., Debruyne, C., O'Sullivan, D., & Lewis, D. (2019). GConsent—a consent ontology based on the GDPR. In P. Hitzler, M. Fernández, K. Janowicz, A. Zaveri, A. J. Gray, V. Lopez, A. Haller, & K. Hammar (Eds.), *The semantic web* (pp. 270–282). Cham: Springer International Publishing.

16. Paschke, A. (2011). Rules and logic programming for the web. Springer: Berlin , pp. 326–381. https://doi.org/10.1007/978-3-642-23032-5_6.
17. Paschke, A., & Bichler, M. (2008). Knowledge representation concepts for automated SLA management. *Decision Support Systems, 46*(1), 187–205. https://doi.org/10.1016/j.dss.2008.06.008.
18. Paschke, A., & Boley, H. (2014). Reaction RuleML 1.0 for distributed rule-based agents in rule responder. In: *Proceedings of the RuleML 2014 Challenge and the RuleML 2014 Doctoral Consortium*, hosted by the 8th International Web Rule Symposium (RuleML 2014), CEUR.org.
19. Robaldo, L. (2021). Towards compliance checking in reified I/O logic via SHACL. In: Maranhão, J., Wyner, A.Z. (Eds.) *ICAIL '21: Eighteenth International Conference for Artificial Intelligence and Law*, São Paulo Brazil, June 21–25, 2021, ACM, pp 215–219. https://doi.org/10.1145/3462757.3466065.
20. Robaldo, L., & Sun, X. (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), 2471–2503.
21. Robaldo, L., Bartolini, C., Palmirani, M., Rossi, A., Martoni, M., & Lenzini, G. (2020). Formalizing GDPR provisions in reified I/O logic: the DAPRECO knowledge base. *Journal of Logic, Language and Information, 29*, 401–449.
22. Schäfermeier, R., & Paschke, A. (2014). Aspect-oriented ontologies: dynamic modularization using ontological metamodeling. In: Garbacz, P., Kutz, O. (Eds.) *Proceedings of the 8th International Conference on Formal Ontology in Information Systems (FOIS 2014)*. IOS Press, Frontiers in Artificial Intelligence and Applications, vol 267, pp. 199–212.
23. Schäfermeier, R., & Paschke, A. (2018). Aspect-oriented ontology development. In: Nalepa, G.J., Baumeister, J. (Eds.) *Synergies between knowledge engineering and software engineering, advances in intelligent systems and computing*, vol 626, Springer, Berlin, pp. 3–30. https://doi.org/10.1007/978-3-319-64161-4_1.
24. Schild, K. (1991). A correspondence theory for terminological logics: preliminary report. In: Mylopoulos, J., Reiter, R. (Eds.) *Proceedings of the 12th International Joint Conference on Artificial Intelligence*. Sydney, Australia, August 24–30, 1991, Morgan Kaufmann, pp. 466–471.
25. Sterling, L., & Shapiro, E. Y. (1994). *The art of Prolog: advanced programming techniques*. Cambridge: MIT Press.