Interoperable and Continuous Usage Control Enforcement in Dataspaces

Inès Akaichi[1,*], Wout Slabbinck[2,*], Julián Andrés Rojas[2], Casper Van Gheluwe[3], Gabriele Bozzi[3], Pieter Colpaert[2], Ruben Verborgh[2] and Sabrina Kirrane[1]

[1] *Institute for Complex Networks, Department of Information Systems and Operations Management, WU Vienna, Austria*
[2] *IDLab, Department of Electronics and Information Systems, Ghent University - imec, Belgium*
[3] *AI & Data, imec, Belgium*

Abstract
In many use cases, policies governing data access need to take time into account: for example, in logistics, the location of a delivery vehicle may only be exposed by a recipient for the duration of their delivery. The Open Digital Rights Language (ODRL) standard, a commonly used policy expressing language, does not support this type of dynamicity. Addressing these use cases requires support to express time-dependent constraints within policy languages, to enforce them in a continuous manner. We created a continuous, interoperable, and modular usage control enforcement framework for dynamic usage control policies, and extended ODRL to support dynamic values in a deterministic manner. This paper details the validation of our solution within a logistics use case. We show that, with a minimal change to ODRL, support for time-dependent constraints can be achieved. This work can provide the ODRL working group with the necessary input that will increase its adoption in domains that depend on time-dependent policies.

Keywords
Dataspaces, Usage Control, Enforcement, Policy

1. Introduction

In 2020, the European Union proposed the Data Governance Act (DGA) with the goal "*to create a framework which will facilitate data-sharing*" [1], thus increasing interest in and accelerating research on dataspaces. Originally, the term dataspace came from the Database Management System (DBMS) world with as goal of creating an information integration system to overcome basic data management challenges over heterogeneous sources [2]. Over the years, there have been several refinements made to the term dataspace [3], though a recurring aspect is the sharing of data between two entities. Recently, initiatives such as the International Data Space Association (IDSA)[1] and Gaia-X[2] were established in order to facilitate data sharing by proposing

The Second International Workshop on Semantics in Dataspaces, co-located with the Extended Semantic Web Conference, May 26 – 27, 2024, Hersonissos, Greece

*These authors contributed equally to this work.

ines.akaichi@wu.ac.at (I. Akaichi); wout.slabbinck@ugent.be (W. Slabbinck); John.Smith@example.org (J. A. Rojas); Casper.VanGheluwe@imec.be (C. V. Gheluwe); Gabriele.Bozzi.ext@imec.be (G. Bozzi); pieter.colpaert@ugent.be (P. Colpaert); ruben.verborgh@ugent.be (R. Verborgh); sabrina.kirrane@wu.ac.at (S. Kirrane)

[1] https://internationaldataspaces.org/
[2] https://Gaia-X.eu/

architectural solutions [4, 5] for dataspaces.

The commonality between dataspace initiatives like IDSA or Gaia-X lies in how they foster data ecosystems, which represent a conceptual framework for data space participants based on specific business needs [6, 4]. Within these ecosystems, various common roles emerge, such as *data providers* (which could also be data producers or owners), *data consumers*, and *data intermediaries* [4]. Data providers and consumers offer and obtain access to data, while intermediaries offer services like cataloging and brokering and data sovereignty assurance.

In Figure 1, we provide a conceptual view of a data ecosystem that depicts the relationship between data consumers, producers, and intermediaries. Each node represents a distinct role within the ecosystem: provider, consumer, or intermediary. The edges between nodes represent a bidirectional relationship presenting data or metadata exchange. Multiple nodes form an ecosystem, and within this larger framework, smaller ecosystems can exist. Dataspaces are the architectural backbone, facilitating communication and data sharing among participants and ecosystems. Achieving interoperability across ecosystem boundaries requires standardization, such as ensuring intermediaries adopt a unified system for identifying and describing data sources. Similarly, trust within and between ecosystems necessitates clear data usage control policies and enforcement mechanisms to ensure data sovereignty. Usage Control extends beyond traditional access control by encompassing deontic rules describing permissions, prohibitions, obligations, mutability of conditions, and continuity of enforcement[7]. Mutable conditions, also known as constraints, refine usage control rules by representing environmental or system requirements that must be satisfied for access, which may evolve over time during resource usage. Continuity of enforcement ensures ongoing policy re-evaluation, thereby maintaining system compliance with usage control policies.

Current dataspace initiatives rely on the *open digital rights language* (ODRL) [8], a W3C standard and recommendation. Originally, ODRL was proposed to express licenses for the use of digital assets. Over the years, it has evolved to, amongst others, express access control policies and privacy policies [9, 10]. For this, ODRL is designed to be a static policy language that necessitates the explicit representation of policy elements such as conditions. Traditionally, access control mechanisms are enforced at the point of request, lacking provisions for ongoing controls over prolonged access or immediate revocation. In today's dynamic distributed environments, characterized by technologies like smart solutions, accommodating mutable constraints becomes imperative for continuous control over digital resource usage. Despite this, ODRL, by design, fails to address mutable constraints, prompting various efforts to resolve this limitation [11, 12, 13, 14]. Solutions proposed by Cano-Benito et al. [11], Cimmino et al. [12], Pandit and Esteves [13] center on employing mapping languages that are technology-centric, while Fornara and Colombetti [14] focus on extending ODRL to incorporate the state lifecycles of permissions, obligations, and prohibitions.

We propose that a simple extension of the model can render ODRL dynamic without the need for any additional technologies. Introducing dynamic policies represents a crucial step towards achieving continuous usage control enforcement [7]. In this paper, we make the following contributions: (i) By drawing inspiration from a logistics use case, we define both functional and non-functional requirements crucial for guiding the development of our solution. (ii) We propose an extension to the ODRL model, enhancing its mutability and dynamic capabilities. (iii) Building upon the mutable ODRL model, we introduce a general dynamic usage control

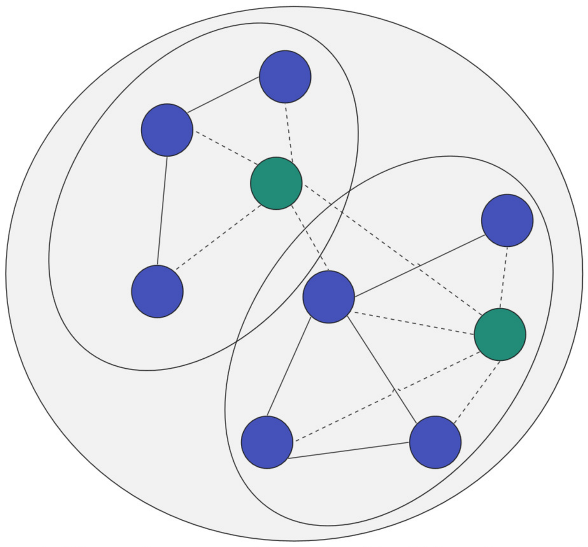
Legend: ● Data Consumer / Data Provider ● Data Intermediary
⬭ Ecosystem —— Data exchange - - - - - Metadata exchange

Figure 1: A Conceptual View of a Data Ecosystem (inspired by [6])

framework. This framework is designed to be implemented at provider and consumer nodes, fostering a shared understanding of how to enforce dynamic policies consistently. (iv) We demonstrate the practical instantiation of our dynamic usage control framework by leveraging various technologies sourced from the literature.

The remainder of this paper is structured as follows: Section 2 depicts the use case and requirements that guide our work. Section 3 presents the related work of our research. Section 4 presents the extension of the ODRL model. Section 5 presents our usage control framework. Section 6 illustrates the instantiation of our framework, accompanied by a discussion of the associated opportunities and challenges. Finally, in Section 7, we conclude and provide some pointers for future work.

2. Use case & Requirements Analysis

To guide the development of our framework, we draw upon the following logistics use case. Metal Solutions is a company that wants cargo delivered from one of its production sites (Aero Metal) to another (Bright Steel). Since late deliveries are expensive, the company wants to know where the cargo is, ensuring they can find alternatives in case delays are predicted. To transport the goods, Metal Solutions contracts a captain to move the cargo using the captain's vessel. As Captain Cove is domiciled on the vessel, they do not want to disclose their location publicly. However, Cove agrees to disclose its location under the constraint that Metal Solutions can access the location only during the transport and for the purpose of predicting the Estimated Time of Arrival (ETA). Since this matches the incentives of Metal Solutions, they can agree by specifying a usage control policy: 1) the first constraint of the policy pertains to the intended purpose of

Delivery Status	Description	Usage Control Decision
not started	voyage not started	no access
shipped	cargo on ship, voyage not started	no access
active/in transit	voyage started	read access for purpose *ETA prediction*
finished/delivered	cargo delivered	no access

Table 1
Evolving states of the delivery status need to result in different policy decisions over time

the data usage. 2) the second constraint describes the delivery status, more specifically, whether the delivery of the cargo has started or is finished. There are four different states for the delivery status, further elaborated in Table 1. This constraint changes over the course of the voyage, making the policy a dynamic one.

2.1. Requirements Analysis

The aforementioned use case gives rise to the following set of requirements.

Functional Requirements. *Authorization.* A data consumer entity is not permitted to access a resource owned or provided by a data provider without explicit authorization from the latter. *Continuous enforcement.* It is essential to continuously enforce data access once granted by the data provider. Fulfilling this requirement involves two primary sub-requirements: utilizing a *dynamic policy language* capable of representing real-time external updates. Specifically, this policy language should be able to represent updates regarding mutable constraints (e.g., delivery status). Implementing *continuous tracking* mechanisms to monitor policy updates, thereby facilitating ongoing evaluation of policy decisions. This enables the enforcement of policies during data usage, including the revocation of access if policy constraints are no longer met.

Non-functional Requirements. *Interoperability & compatibility.* Different participants coexist within a data realm, known as intra-data space. Moreover, diverse data realms foster inter-data space interoperability. Consequently, an issue arises regarding whether the policies and attributes employed to establish trust within a data realm are compatible between two authorities of data realms and within a single data realm. For this, the usage control developed need to achieve interoperability by enforcing the use of standards and protocols. *Transparency* entails that both the data consumer and Provider are informed about the enforcement of usage control on both ends, ensuring there's a documented record of policies, access requests, and decisions for accountability purposes. Furthermore, the data consumer should receive notifications whenever access is granted, denied, or revoked. This is essential for the data consumer to be aware of any decision regarding his usage of Data.

3. Related Work

In the following, we present related work in relation to ODRL modelling and usage control in dataspaces.

3.1. Dynamic ODRL Policies

The *open digital rights language* (ODRL) Information Model 2.2[3] is a W3C standard for expressing policies. The language allows expressing permission and prohibition actions over resources. Furthermore, obligations can also be defined which must be met by the parties involved. A permission, prohibition and obligation rule can be further refined by constraints. ODRL constraints always have the following three properties: (i) left operand, (ii) operator; and (iii) right operand. Together they form a boolean expression which will evaluate to either $true$ or $false$. For static constraints, these suffice.

Fornara and Colombetti proposed to extend ODRL by considering temporal aspects of obligations, permissions and prohibitions such that their lifecycle can be formalized by state machines [14]. The motivation is that the result of an evaluation of those deontic concepts depends on **when** they are evaluated. Cano-Benito et al. [11] state that ODRL policies can not deal with dynamic information. To overcome this, they propose to use the SIoTRx language to inject said dynamic values into an ODRL policy, to which their custom ODRL interpreter can interpret the result. In another paper, they list several challenges for ODRL. One of them is the fact that privacy might need to consider information that is outside of the policies themselves [12]. To overcome this challenge, they suggest using RDF mapping languages to materialize these external values. Pandit and Esteves introduced a class *TemplateQuery* which acts as a placeholder in ODRL policies to ensure their validity [13]. On evaluation of these policies, a SPARQL query is executed to instantiate the value(s).

3.2. Usage Control in Dataspaces

The concept of *usage control*, as initially proposed by Park and Sandhu [7], pertains to the continuous monitoring of digital asset usage within dynamic environments. Usage control extends traditional access control by employing machine-readable policies to specify future data usage requirements and the mechanisms necessary for enforcing these policies. The specification of usage control policies inherently raises questions regarding their enforcement.

In the realm of dataspaces, usage control is deemed indispensable for fostering and preserving trust among users and stakeholders. For instance, the International Data Spaces Association (IDSA) has proposed an IDS reference architecture [15], wherein the IDS connector serves as a pivotal component to be incorporated on the consumer and provider sides. This connector integrates essential elements such as access and usage control mechanisms. Notably, the latest version of IDS connectors stipulates the specification of usage control policies based on an ODRL profile, entitled the IDS Usage Control Policy (UCP)[4].

To enforce ODRL policies, IDSA uses MYDATA technologies[5], developed by Fraunhofer IESE [6]. In particular, ODRL policies can be translated to MYDATA policies, which follow the Event-Condition-Action paradigm [16] and are XML-based. MYDATA policies, being technology-dependent, are then enforced using an extended version of the eXtensible Access Control

[3]https://www.w3.org/TR/odrl-model/

[4]The IDS UCP, https://international-data-spaces-association.github.io/DataspaceConnector/Documentation/v6/UsageControl

[5]MYDATA, https://www.mydata-control.de/

[6]Fraunhofer IESE, https://www.iese.fraunhofer.de/

Markup Language (XACML)[7] enforcement framework. Originally designed as an access control framework, XACML has been extended to incorporate interceptors within connectors at the data consumer side[8]. This extension enables the interception of data flows between services and applications, facilitating enforcement at varying levels of data abstraction.

However, as usage control policies depend on rapidly changing environmental and system conditions [7], to the best of our knowledge, it is currently impossible with the proposed IDS solution to address dynamic policies from a specification and application point of view. Further investigation of this aspect is required for a full realization of usage control.

On the other hand, Gaia-X has emphasized the importance of usage control as a fundamental element of its overall architecture. However, there is limited readily available information on the latest advances in their usage control solution.

4. Dynamic ODRL Specification

In ODRL, policies consist of a set of rules, each of which is associated to a party, target and an action. Rules can be classified into permissions, duties, and prohibitions. Actions describe what could be executed over a target, directly or indirectly by a party, if the set of constraints associated with a specific rule are met. A constraint is described using properties such as *leftOperand*, *rightOperand*, and *operator* associated with respectively *LeftOperand*, *RightOperand*, and *Operator* classes, which are defined in the ODRL vocabulary[9]. Additionally, one can express a constraint using the *rightOperandReference* property. This property is associated with an IRI, which must be first de-referenced in order to obtain the actual right operand value. Unfortunately, the current definition of the *rightOperandReference* property leaves room for interpretation as it does not state deterministically how to get that value from that dereferenced IRI since the specification fails to describe the structure of the IRI and how it should be represented. This makes it impossible to create an interoperable constraint satisfier.

To address this limitation[10], we suggest an extension to the Constraint class by introducing the *OperandRefence* class as the object for the `odrl:rightOperandReference` property as shown in Figure 2. The *OperandRefence* has two attributes: `odrluc:reference` and `odrluc:path`. The first property points to the external source, which has the extra condition that it is exposed employing the Resource Description Framework (RDF), containing information describing updates related to a specific right operand value. Dereferencing this source will result in an RDF Knowledge Graph g. The second property states how to retrieve the latest updated dynamic value from g. For this, we opted for a *Property Path*[11], which is defined in the Shapes Constraint Language (SHACL) W3C recommendation[12]. In SHACL, this property declares the path from a specific focus node to a value it describes. Thus we re-use the syntax and definition of the

[7]XACML, http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[8]IDSA Usage Control Framework, https://github.com/International-Data-Spaces-Association/IDS-G/blob/main/UsageControl/Enforcement/README.md

[9]ODRL Vocabulary & Expression 2.2, https://www.w3.org/TR/odrl-vocab/

[10]The following prefixes are used throughout Section 4: odrl:<http://www.w3.org/ns/odrl/2#>; odrluc:<http://example.org/odrl-usage-control#>; ex:<http://example.org#>

[11]Property Path, https://www.w3.org/TR/shacl/#property-paths

[12]Shapes Constraint Language, https://www.w3.org/TR/shacl/

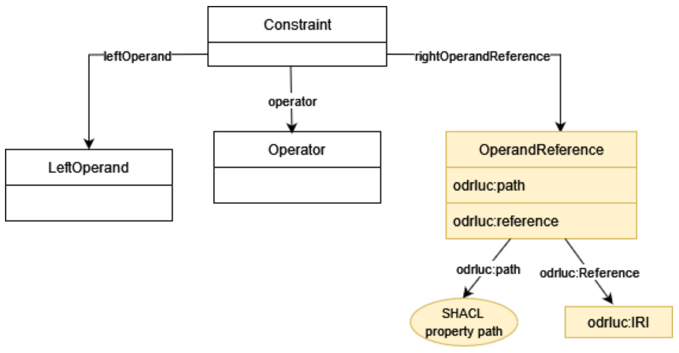

Figure 2: ODRL Constraint class with in yellow the proposed extension that refines dynamic values.

SHACL Property Path in the `odrluc:path` property and add an extra constraint imposing that the `odrluc:path` can only lead to one value, which is the latest updated value.

Listing 1 shows a dynamic policy representing the delivery status constraint. In this constraint, the state *"shipped"* of the delivery status is an instance of the class *ex:State*. Listing 3 presents the materialization of the delivery status constraint after dereferencing the external source in Listing 2. Considering our motivating scenario, we presume that the external source contains an RDF Knowledge Graph detailing updates on the delivery status of Metal Solution's cargo. The satisfiability of the materialized constraint can now be calculated by any ODRL Evaluator.

```
ex:deliveryStatusConstraint a odrl:Constraint.
    odrl:leftOperand ex:State;
    odrl:operator odrl:eq;
    odrl:rightOperandReference ex:operandReference1 .
ex:operandReference1 a odrluc:OperandReference ;
    odrluc:reference ex:external−source ;
    odrluc:path ex:updatedValue .
```

Listing 1: Dynamic ODRL Constraint

```
ex:external−source ex:updatedValue "shipped" .
```

Listing 2: External source at url *https://example.org*

```
ex:deliveryStatusConstraint a odrl:Constraint.
    odrl:leftOperand ex:State;
    odrl:operator odrl:eq;
    odrl:rightOperand "shipped" .
```

Listing 3: Materialisation of the dynamic ODRL Constraint

5. Dynamic Usage Control Framework

In this section, we present our dynamic usage control framework. Additionally, we present the necessary steps that allow for the continuous enforcement of usage control policies. The

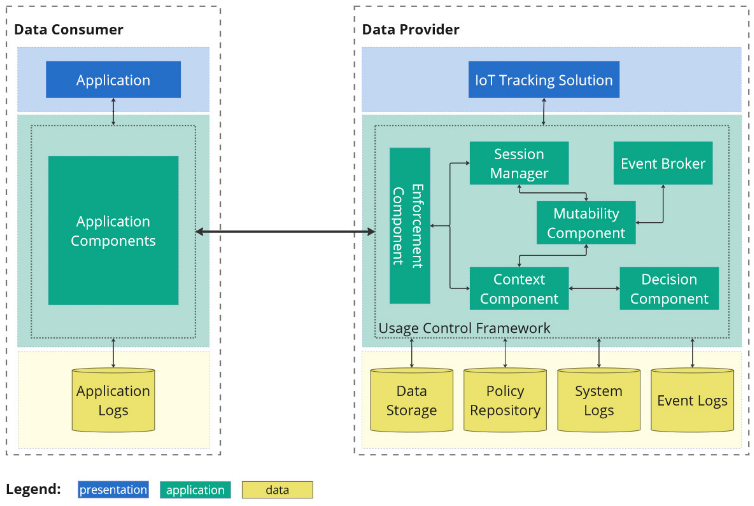

Figure 3: A Three-tier Architectural Overview of Data Provider and Data Consumer Nodes

architecture is supposed to be general enough to cater for any usage control policy. However, we assume the usage of ODRL policies especially for the enforcement of dynamic policies. We also assume that all actors participating in the architecture below are already authenticated.

5.1. Architecture

We assume that our usage control solution resides at both consumer and provider nodes, as both roles are interchangeable. Figure 3 depicts a three-tier architectural overview of data consumer and data provider views. The data consumer refers to any client application that utilizes data provided by the data provider. In our scenario, the data consumer is a data-sharing application facilitating meta-solutions to access real-time location information of Captain Cove during ongoing cargo transfer. The application and data layers present specific internal components of the client application. The data provider presents an Internet of Things (IoT) device operating at the presentation layer. It tracks Captain Cove's location using sensors installed on the ship and provides the cargo delivery status. This data is relayed to our application layer, which incorporates the usage control system framework. The framework ensures adherence to Captain Cove's policies, employing a separation of duties across its components. Next to the shared data, the data layer presents specific storage for our solution that can also be part of the dataspace data pool. The usage control framework comprises several key components:

- The *Enforcement Component* handles access requests triggered by the data consumer and enforces (i.e., allowing, denying or revoking) access decisions received from the context component.
- The *Session Manager* is responsible for managing meta-information associated with the usage session, e.g., session ID. Every time a new access request is sent to the enforcement component, the session manager is responsible for creating a session and returning a session ID to the enforcement component.

- The *Context Component* is the main orchestrator of our framework. As soon as it receives an access request from the enforcement component, it starts by (1) fetching the relevant policy from the Policy Repository. If it (2) recognizes dynamic constraints (e.g., status) in a policy, it (3) queries the latest values for dynamic constraints from the mutability component and (4) reconstructs an instance of a policy based on the latest dynamic constraint. The instantiated policy and the access request are then forwarded to the decision component.
- The *Mutability Component* is responsible for processing events regarding updates of constraint values in dynamic policies and maintaining the event log. We assume that the mutability component subscribes to the event broker in order to follow these updates as soon as a new policy is added to the policy repository.
- The *Decision Component* conducts reasoning over policies and access requests, providing decision responses such as access granted (e.g., in case the state of cargo delivery is still active), denied, or revoked (e.g., in case the state of cargo delivery is delivered).
- The *Event Broker* enables the transmission of events regarding constraints value updates (e.g., delivery status) to the rest of the components.
- The *Data Storage* contains data to be accessed and other metadata.
- The *Policy Repository* contains usage control policies.
- The *Event Logs* stores updates regarding the updates of dynamic values related to policy constraints using an event ontology.
- The *System Logs* stores system activities, including access requests, decisions, and associated sessions.

To actually enforce the policy, a distinction has to be made by which actor initiates the enforcement process. When the data consumer wants to actively request the location data, the enforcement of read access has to be evaluated at the time of this request. From now on, **Point-in-Time Enforcement** will be used to refer to this type of usage control. A second kind of enforcement, **Continuous Enforcement**, is initiated by the data provider itself. Here, the Enforcement System will continuously verify whether the current access is still allowed.

5.2. Point-in-Time Enforcement

In this flow, the data consumer seeks data from the data provider, necessitating an Access Token from the Usage Control System. This Access Token grants access to the desired data. Since the emphasis is on authorizing the data consumer, it is presumed that the identity of the data consumer has been verified, and therefore authentication is managed separately. A credential verifier framework within a dataspace node with the sole duty to verify credentials such as identity could ensure that the Usage Control framework is only reached when identity claims have been checked. The positive flow of Point-in-Time Enforcement is illustrated in Figure 4 and elaborated as follows. The data consumer initiates the flow by making an Access Token request through the data consumer Application as without such a token, data access will not be granted. For audit and transparency reasons, the Enforcement Component passes the request to the Session Manager (2). The Session Manager initiates a session based on the request and records the session details in the System Logs (3). This session remains valid until the user ceases access

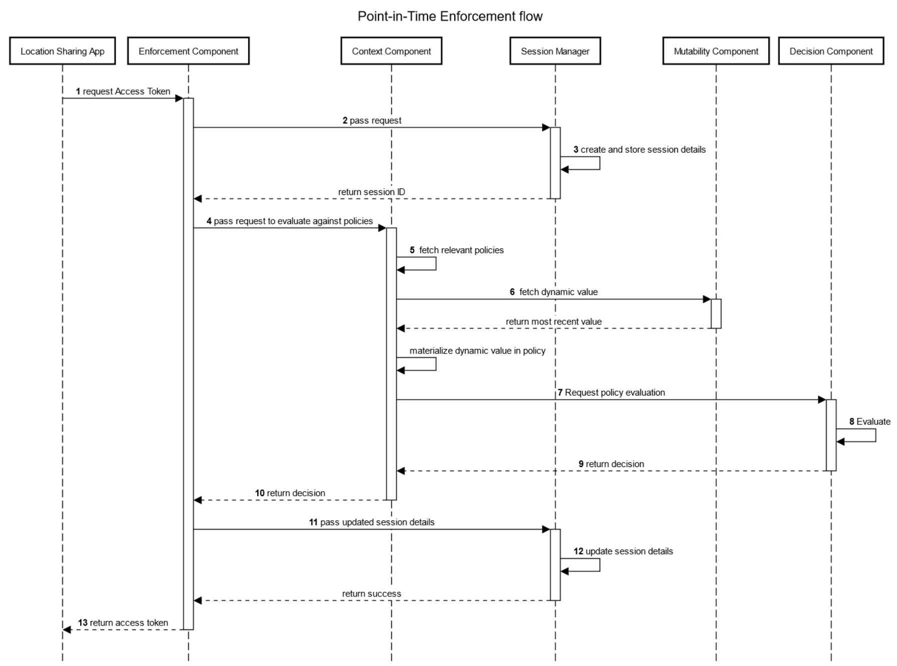

Figure 4: Positive Point-in-Time Enforcement flow showing (i) the data consumer instigating an access request (1), (ii) the logging of this request (2-3), (iii) materializing the policy with dynamic value and evaluating it (4-10), (iv) updating the session (11-12), and (v) enforcement of the decision by the Enforcement Component (13).

to the data or until a revocation decision emerges from the Decision Component all the way to the Enforcement Component. After receiving the session, the Enforcement Component passes the request to the Context Component in order to evaluate the request against the policies (4). Using details from the request, the Context Component fetches the relevant policy from the Policy Repository (5). The Context Component recognizes that there is a dynamic value in the policy and realizes that this policy must first be materialized before any evaluation can be executed. So it fetches the most recent dynamic values from the Mutability Component (6) and materializes the value using the algorithm described in Section 4. Now that the policy is materialized, it is passed together with the request to the Decision Component (7). In the Decision Component, an evaluation is performed which has as a result two options: access granted or access denied (8). The resulting decision is passed back to the Context Component (9) and subsequently transmitted to the Enforcement Component (10). The Enforcement Component updates the session with the decision. As this must be logged again, the session details are passed to the Session Manager (11). The Session Manager updates the System Logs (12). When the decision is access granted, the Enforcement Component finally returns an Access Token to the calling application (13). Otherwise, no token is returned, indicating that the application will not be able to access the data.

5.3. Continuous Enforcement

The Continues Enforcement pertains to the fact that data usage is always valid. In our use case, this is decided by the actual shipping value indicated by the IoT Tracking Solution. Figure 5 shows the positive continuous enforcement flow. It starts with the fact that the IoT Tracking Solution sends an update to the Event Broker (1). This update is transferred to the Mutability Component through a notification (2) since the Mutability Component is subscribed to the Event Broker regarding a specific topic (more specifically, an *IRI* that corresponds to the dynamic value in the policy constraint). At this stage, the Mutability Component performs two main tasks. It updates the value in the Event Logs, and then it notifies the Session Manager by passing the *IRI* and the corresponding updated value (3). The Session Manager checks whether there is an active session by querying the System Logs using the *IRI* to retrieve the most recent session details (4), which include the policy ID, the original access request, decision, timestamp, and other relevant system information. This active session is transmitted in a notification to the Enforcement Component (5). Since the Enforcement Component is triggered internally with a session, a re-evaluation needs to happen to get a decision based on the newest constraint conditions. This is done by transmitting the request to the Context Component (6). The evaluation, orchestrated by the Context Component shown in steps (7) to (12) which is the same as steps (5) to (10) from the Point-in-Time Enforcement flow, returns a decision. For transparency, the System Logs are again updated with the decision in the session (13)(14). Finally, the Enforcement Component notifies the actor in question and enforces the usage decision (15).

6. Instantiation & Discussion

In this section, we demonstrate the instantiation of our framework utilizing state-of-the-art technologies. Additionally, we address certain limitations identified within our framework and highlight relevant works that offer potential solutions to overcome these limitations.

6.1. Instantiation

Here, we discuss the design patterns required to instantiate our modular, continuous, and interoperable usage control framework. Furthermore, we provide a selection of semantic web technologies to employ these patterns. The different domains we tackle are (i) the interfacing to the outside dataspace ecosystem, (ii) how to model version events, (iii) the implications of the chosen policy model, and (iv) how to orchestrate everything within the framework.

Our framework has two components that communicate with other entities inside the dataspace ecosystem, in particular the *Enforcement Component* and the *Event Broker*. For the handling of an access request coming from the data consumer, the *Enforcement Component* acts as a server in the client-server model by providing an Access Token, for which a JSON Web Token (JWT) can be used[13]. To support continuous enforcement, the enforcement component must send notifications to other entities, potentially utilizing semantic web technologies, such as "Event Notifications in Value-Adding Networks" formed with Activity Streams 2.0[14] messages

[13]https://datatracker.ietf.org/doc/html/rfc7519

[14]https://www.w3.org/TR/activitystreams-core/

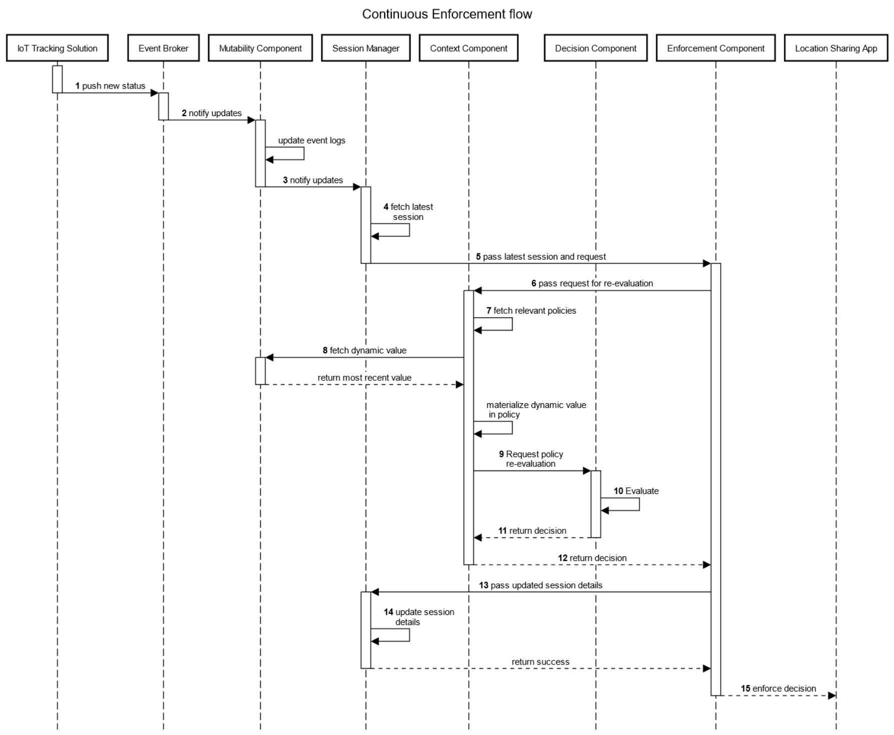

Figure 5: Positive Continuous Enforcement flow depicting (i) the initiation originating from within the data provider (1-5), (ii) continuation of the session by re-evaluation using the updated value (6-14) and (iii) enforcing the newest decision by the Enforcement Component (15).

[17]. As the *Event Broker* must listen to messages pertaining to the most recent status of external sources, it can be built as a Message-Oriented Middleware [18] using protocols such as MQ Telemetry Transport(MQTT)[15] or (Advanced Message Queuing Protocol) AMQP[16].

As the *Mutability Component* must subscribe to the *Event Broker*, there is a need for a model that captures three aspects, namely the unique identifier of the external source (also the topic), the actual value and the corresponding timestamp. For this, we suggest the use of the Sensor, Observation, Sample, and Actuator (SOSA) ontology[17] or the Smart Applications REFerence (SAREF)[18], as these models offer these three properties using their respective `Observation` and `Measurement` classes. When the messages do not arrive in the aforementioned models, a mapping is required by employing any of the RDF generation languages as described in the survey paper by Van Assche et al. [19]. To build the event log and to materialize the most recent constraint values, versioning techniques are required. For this purpose, the Memento framework[19] or the versioning Linked Data Event Streams[20] (LDES) can be leveraged. A similar

[15]https://datatracker.ietf.org/doc/html/rfc9431
[16]https://www.amqp.org/
[17]SSN/SOSA ontology, https://www.w3.org/TR/vocab-ssn/
[18]Smart Applications REFerence, https://saref.etsi.org/
[19]https://www.rfc-editor.org/rfc/rfc7089.html
[20]https://w3id.org/ldes/specification/

versioning solution is required for the *Session Manager* as it needs to persist the most recent status of a session.

An implementer deciding to use ODRL as a policy language can use an ODRL Evaluator for the *Decision Component.* An ODRL evaluator, which the ODRL Formal Semantics Community group is currently working on, would solve the problems of policy interpretability by relying on a formalization of the ODRL model.

Once all of the above is set in stone, the *Context Component* can now perform its tasks based on the chosen protocols and technologies.

6.2. Meeting the Requirements and Discussion

Using our solution, we were able to address the following requirements: *Authorization.* data consumer nodes need to receive an authorization token to be able to access data on the provider side. *Continuous enforcement.* is achieved, on the one hand, using our *dynamic policy language* that dynamically represents the mutable constraints and corresponding updates in the right operand IRI. On the other hand, continuous enforcement is achieved via *continuous tracking* of mutable constraints, in particular, traces of continuous access are managed using our Session Manager and stored in the System Logs. Additionally, the continuous evaluation of our dynamic policy is facilitated by having a mutability component that can persist policy updates in Event logs, which would trigger policy re-evaluation. *Interoperability & Compatibility.* is achieved by using standards such as ODRL, and semantic web technologies to describe events in our framework and policy updates. This allows the creation of interoperable components that can communicate with other nodes inside the ecosystem. *Transparency.* We made sure that the framework activities (e.g., decisions, policies, constraint updates, and requests) are being monitored and recorded for accountability purposes. Additionally, when access requests are denied, actors are usually notified by the enforcement component. All these various requirements are essential in achieving trust between the different actors of the ecosystem.

Our solution leaves space for open issues and interpretations. Using our use case, the main dynamic constraint is the delivery status. We can already foresee other dynamic constraints, such as temporal, location, etc., that can be expressed using our model. Temporal constraints introduce challenges, particularly concerning the Maximum Acceptable Latency and determining the appropriate interval for the Event Broker to disseminate updates to other framework components. Notably, the continuous nature of time necessitates more frequent update notifications compared to delivery status. Van de Vyvere et al. [20] already tried to tackle this issue in real-time systems, which can bring some inspiration to our solution. Ensuring continuous access to location data within our enforcement framework presents minimal challenges as it primarily involves read actions. However, difficulties may arise in enforcing actions such as data deletion obligations post-usage, which are often challenging to observe and enforce effectively. Auditing mechanisms and regulations are essential to tackling this issue. Enforcement of specific policy actions, like data transformations across various data abstraction levels, presents another noteworthy challenge. This often requires enforcing data usage at the flow level through multiple enforcement points. For this, the use of interceptors [15], or controlling messages across services [21], at multiple sites showed efficacy in dealing with such cases. Additionally, using logs to trace system activities is an essential component as part of our framework. However, establishing trust in log components is deemed to be an open issue in our solution. Dataspace solutions, such as IDSA, propose to address this through the use of certifications for usage components. Other approaches include ensuring that logs are immutable using blockchain technologies and Trusted Execution Environments [22].

7. Conclusion & Future Work

In this paper, we introduced an extension to ODRL, transforming it into a mutable policy language. Moreover, we presented a comprehensive dynamic usage control framework capable of enforcing dynamic policies. We anticipate that the extension of ODRL will stimulate discussions on how to effectively evolve this standard to accommodate the dynamic nature of use cases within dataspaces. Additionally, we see a need to instantiate our solution and implement it using some of the technologies that we suggested in our paper, taking into account the limitations. This would allow us to evaluate the performance and scalability of our framework in decentralized ecosystems.

Acknowledgments

Supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10) and SYTADEL (SYnchromodal proTotype for Data Sharing and PLanning), c-SBO project(VIL, imec, UAntwerpen and Vlerick), funded by VLAIO. Ines is funded by the European Union Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 860801. Sabrina Kirrane was partially funded by the FWF Austrian Science Fund and the Internet Foundation Austria under the FWF Elise Richter and netidee SCIENCE programmes as project number V 759-N.

References

[1] Proposal for a regulation of the european parliament and of the council on european data governance (data governance act), 2020. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A52020PC0767.

[2] M. Franklin, A. Halevy, D. Maier, From databases to dataspaces: a new abstraction for information management, ACM SIGMOD Record 34 (2005) 27–33.

[3] J. Theissen-Lipp, M. Kocher, C. Lange, S. Decker, A. Paulus, A. Pomp, E. Curry, Semantics in Dataspaces: Origin and Future Directions, in: Companion Proceedings of the ACM Web Conference 2023, ACM, Austin TX USA, 2023, pp. 1504–1507.

[4] L. Nagel, D. Lycklama, Design principles for data spaces - position paper, 2021.

[5] V. Siska, V. Karagiannis, M. Drobics, Building a Dataspace: Technical Overview, 2023. URL: https://www.gaia-x.at/wp-content/uploads/2023/04/WhitepaperGaiaX.pdf.

[6] B. Otto, The Evolution of Data Spaces, Springer International Publishing, Cham, 2022, pp. 3–15.

[7] J. Park, R. Sandhu, The UCONABC usage control model, ACM Transactions on Information and System Security 7 (2004) 128–174.

[8] W3C Working Group, The open digital rights language (odrl), https://www.w3.org/TR/odrl-model/, 2018.
[9] B. Esteves, H. J. Pandit, V. Rodríguez-Doncel, Odrl profile for expressing consent through granular access control policies in solid, in: 2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), 2021, pp. 298–306.
[10] M. D. Vos, S. Kirrane, J. Padget, K. Satoh, Odrl policy modelling and compliance checking, in: RuleML+RR, 2019.
[11] J. Cano-Benito, A. Cimmino, R. García-Castro, Injecting data into ODRL privacy policies dynamically with RDF mappings, in: Companion Proceedings of the ACM Web Conference 2023, ACM, Austin TX USA, 2023, pp. 246–249.
[12] A. Cimmino, J. Cano-Benito, R. García-Castro, Practical challenges of ODRL and potential courses of action, in: Companion Proceedings of the ACM Web Conference 2023, WWW '23 Companion, Association for Computing Machinery, New York, NY, USA, 2023, pp. 1428–1431.
[13] H. J. Pandit, B. Esteves, Enhancing Data Use Ontology (DUO) for health-data sharing by extending it with ODRL and DPV, Semantic Web Preprint (2024) 1–26. Publisher: IOS Press.
[14] N. Fornara, M. Colombetti, Using semantic web technologies and production rules for reasoning on obligations, permissions, and prohibitions, Ai Communications 32 (2019) 319–334. Publisher: IOS Press.
[15] A. Eitel, C. Jung, R. Brandstädter, A. Hosseinzadeh, S. Bader, C. Kühnle, P. Birnstill, G. Brost, Gall, F. Bruckner, N. Weißenberg, B. Korth, Usage control in the international data spaces, 2021.
[16] A. Hosseinzadeh, A. Eitel, C. Jung, A systematic approach toward extracting technically enforceable policies from data usage control requirements, in: International Conference on Information Systems Security and Privacy, 2020.
[17] P. Hochstenbach, H. Van de Sompel, M. Vander Sande, R. Dedecker, R. Verborgh, Event Notifications in Value-Adding Networks, in: G. Silvello, O. Corcho, P. Manghi, G. M. Di Nunzio, K. Golub, N. Ferro, A. Poggi (Eds.), Linking Theory and Practice of Digital Libraries, Springer International Publishing, Cham, 2022, pp. 133–146.
[18] E. Curry, Message-oriented middleware, Middleware for communications (2004) 1–28.
[19] D. Van Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. De Meester, A. Dimou, Declarative RDF graph generation from heterogeneous (semi-) structured data: A systematic literature review, Journal of Web Semantics 75 (2023) 100753. Publisher: Elsevier.
[20] B. Van de Vyvere, P. Colpaert, R. Verborgh, Comparing a polling and push-based approach for live open data interfaces, in: M. Bielikova, T. Mikkonen, C. Pautasso (Eds.), Web Engineering, Springer International Publishing, Cham, 2020, pp. 87–101.
[21] J. Schuette, G. S. Brost, Lucon: Data flow control for message-based iot systems, in: 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 2018, pp. 289–299.
[22] D. Basile, C. Di Ciccio, V. Goretti, S. Kirrane, Blockchain based resource governance for decentralized web environments, Frontiers in Blockchain 6 (2023).