Modeling and Managing Temporal Obligations in GUCON Using RDF 1.2 and SPARQL 1.2

Ines Akaichi [a,*], Giorgos Flouris [b], Irini Fundulaki [b] and Sabrina Kirrane [a]
[a] *Institute for Complex Networks, Vienna University of Economics and Business, Austria*
E-mails: ines.akaichi@wu.ac.at, sabrina.kirrane@wu.ac.at
[b] *Institute of Computer Science, FORTH, Heraklion, Greece*
E-mails: fgeo@ics.forth.gr, fundul@ics.forth.gr

Abstract. In the digital age, data frequently crosses organizational and jurisdictional boundaries, making effective governance essential. Usage control policies have emerged as a key paradigm for regulating data usage, safeguarding privacy, protecting intellectual property, and ensuring compliance with regulations. A central mechanism for usage control is the handling of obligations, which arise as a side effect of using and sharing data. Effective monitoring of obligations requires capturing usage traces and accounting for temporal aspects such as start times and deadlines, as obligations may evolve over time into different states, such as fulfilled, violated, or expired. While several solutions have been proposed for obligation monitoring, they often lack formal semantics or provide limited support for reasoning over obligation states. To address these limitations, we extend GUCON, a policy framework grounded in the formal semantics of SPARQL graph patterns, to explicitly model the temporal aspects of an obligation. This extension enables the expressing of temporal obligations and supports continuous monitoring of their evolving states based on usage traces stored in temporal knowledge graphs. We demonstrate how this extended model can be represented using RDF 1.2 and SPARQL 1.2 and propose an Obligation State Manager that monitors obligation states and assesses their compliance with respect to usage traces. Finally, we evaluate both the extended model and its prototype implementation. Results show a linear performance trend with respect to increasing numbers of obligations and larger knowledge graph sizes.

Keywords: Knowledge Graphs, Usage Control, Temporal Constraints, Policies

1. Introduction

In modern decentralized environments, such as data spaces and knowledge graph applications, data routinely moves across organizational and jurisdictional boundaries. The unrestricted flow of information brings with it the need for continuous and context-aware enforcement of policies to ensure the governance of data usage. In this context, usage control has emerged as a paradigm for safeguarding privacy, protecting intellectual property rights, and ensuring compliance with regulations [1]. Unlike traditional access control, which primarily focuses on determining who is permitted to access a given resource, usage control extends this notion by regulating not only access but also how, when, and under what conditions the data may subsequently be used, modified, or shared [47].

A central mechanism for the operationalization of these conditions and ensuring continuous policy enforcement is the concept of obligations [29]. Obligations specify the required actions that must be performed before, during,

*Corresponding author. E-mail: ines.akaichi@wu.ac.at.

1570-0844/$35.00 © 0 – IOS Press. All rights reserved.

or after data usage thereby ensuring accountability, compliance, and trust throughout the entire data usage process. Examples of such obligations include the requirement to delete data within a period of five years, or to notify the data owner whenever a document is shared with third parties. For this, effective monitoring of obligations requires the incorporation of temporal aspects such as the start time and deadline, as obligations may evolve through different states, such as fulfilled, violated, or expired [19], as time passes. The enforcement of obligations follows two main strategies [51]: preventive and detective. Preventive mechanisms delay an attempted usage request until the corresponding obligations have been fulfilled, ensuring that actions are taken before access or further processing occurs. In contrast, detective mechanisms focus on verifying compliance after an obligation has been executed, typically through auditing or logging tools. The latter depends upon a representation of the state of affairs that captures domain knowledge and records events in the form of logs or usage traces [2]. As highlighted by Hilty et al. [29], many obligations are difficult to enforce proactively; for example, confirming the permanent deletion of data is challenging to ensure in real time but can be assessed retrospectively through log records.

Several models have been proposed for usage control and obligation monitoring in particular. The Usage CONtrol model (UCON) [47] introduced obligations, decision continuity, and attribute mutability, yet despite various formalizations and integration attempts [13, 42], no standard specification or widely adopted implementation exists. Policy languages such as the Obligation Specification Language (OSL) [29], Rei [37], Ponder [14], Proteus [62], KAoS [64], and the Dynamic Spectrum Access Policy Framework (DSA policy framework) [58] support temporal constraints alongside obligations but lack models for obligation states, limiting their ability to track obligation life cycles. The Open Digital Rights Language (ODRL) [33], while widely recognized, lacks official formal semantics, hindering rigorous reasoning about temporal aspects such as obligation start times and deadlines. Furthermore, the majority of the proposed solutions do not model the state of affairs, an essential component for storing usage traces and enabling dynamic reasoning over obligation monitoring and compliance with usage policies in general.

In order to address these gaps, we initially proposed the Generic Graph Pattern-based Policy Framework for Usage Control Enforcement (GUCON) [2], which is a usage control framework that allows permission, prohibition, obligation, and dispensation policies to be expressed. We also define the notion of the state of the affairs, which is a knowledge graph that captures domain knowledge and events, and serves as the basis for reasoning about and enforcing usage control policies. In this work, we extend the GUCON framework by incorporating temporal properties into obligations, enabling their monitoring over time and demonstrate how it can be instantiated using RDF 1.2 and SPARQL 1.2. Our main contributions are as follows: (i) we extend the original GUCON obligation model with start times and deadlines, allowing reasoning about obligation states; (ii) we formally define two key reasoning tasks that determine the states of temporal obligations (such as active, fulfilled, or violated) and assess the compliance of the knowledge base with respect to these defined obligations; (iii) we demonstrate how this extended model can be represented using RDF 1.2 and SPARQL 1.2 and propose an Obligation State Manager that monitors obligation states and checks compliance of obligations against the state of affairs; and (iv) we evaluate the extended model and its prototype implementation.

The remainder of this paper is structured as follows: Section 2 presents a motivating use case describing the process of inpatient care in a hospital. Section 3 introduces the preliminaries in relation to RDF, SPARQL, and GUCON. Section 4 defines the formal semantics of the extended GUCON model in order to cater for temporal obligations. Section 5 describes our RDF 1.2, SPARQL 1.2, and Obligation State Manager instantiations and Section 6 evaluates both the extended obligation model and our Obligation State Manager. Section 7 subsequently positions our work with respect to the state of the art. Finally, Section 8 concludes and discusses future work.

2. Use Case

Consider an imaginary application, OncoAid, which is designed to support cancer patient health monitoring by enabling them to manage their medical and health-related data. The app is integrated with a smartwatch that continuously collects and monitors vital signs (e.g., heart rate and blood pressure), which are stored in the patient's personal Knowledge Graph (KG). In addition to real-time monitoring, OncoAid provides access to key medical data, including electronic medical records, lab results, imaging, medication plans, and doctor diagnostic notes. This data is maintained in the hospital's KG. Patients using OncoAid can view and manage their personal data through

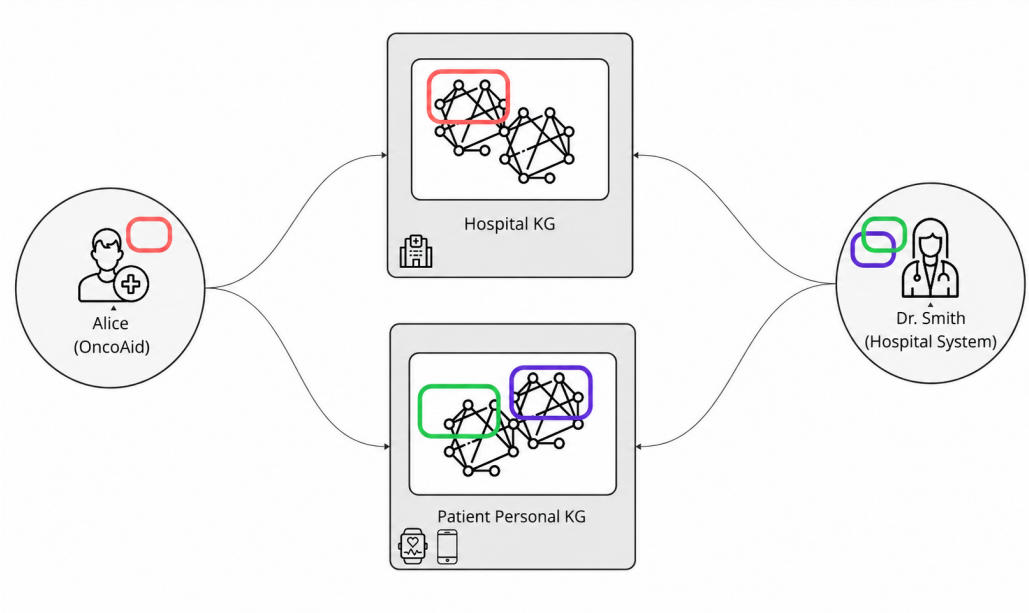


Figure 1. A Depiction of the OncoAid Use Case

their own KG while also accessing relevant data from the hospital KG. While, doctors are responsible for managing hospital data and, when necessary, accessing the patient's KG to provide informed medical care.

Running Example. The running example, which is depicted in Figure 1, illustrates the process of inpatient care at CityCare Hospital, where Alice is admitted for treatment. Upon admission, her corresponding doctor, Dr. Smith, performs different laboratory tests. By the end of her stay, a diagnosis is concluded, and all diagnosis notes and laboratory results are stored in the hospital KG. During Alice's hospitalization, we envisage the following scenarios:

> **Scenario 1 (S1).** During Alice's stay, Dr. Smith orders lab tests, in which the results are stored in the hospital KG. Dr. Smith *must* devise a treatment plan and share it with Alice after her lab results are ready.
> **Scenario 2 (S2).** As Alice's expected discharge date approaches, she *must* review and electronically sign her discharge form before the scheduled end of her inpatient care. This electronically signed document is securely stored in the hospital's KG.
> **Scenario 3 (S3).** After Alice is discharged from CityCare Hospital, her doctor *must* sign and finalize her diagnostic report within 12 hours. Alice's discharge from CityCare marks the actual end of her inpatient care.

The different scenarios highlight various obligations governing data usage. Each obligation is regulated by temporal rules that determine whether an action must occur before, after, or within a specified time window. All throughout the paper, the different scenarios in this use case will be modeled using the Hospital Inpatient Care (HIC) ontology[1].

3. Preliminaries

The Generic Graph Pattern-based Policy Framework for Usage Control Enforcement (GUCON) [2] provides an abstract and semantically grounded structure to specify usage control policies that support their implementation. Additionally, the framework defines approaches for policy reasoning tasks, including consistency, requirements, and compliance checking[2]. These tasks are defined based on the concepts of state of affairs and usage control policies,

[1]The HIC ontology is accessible on our GitHub repository, https://github.com/Ines-Akaichi/Temporal-GUCON/tree/main/ontologies/domain
[2]In this paper, we focus primarily on compliance and requirements checking, as our emphasis is solely on obligations.

which rely on RDF and SPARQL. Accordingly, the formal basis for reasoning is provided by SPARQL semantics. In the following, we begin by presenting the essential SPARQL preliminaries, followed by an introduction to GUCON.

3.1. RDF & SPARQL

In our work, we rely on the syntax and semantics of graph pattern expressions presented in [25]. Throughout the paper, we assume two pairwise disjoint and countably infinite sets I and L to denote respectively Internationalized Resource Identifiers (IRIs) and literals. We denote by IL the union of $I \cup L$. Note that blank nodes are omitted for simplicity. We introduce the concepts of subject `s`, property `p`, and object `o` to form `subject-property-object` RDF triples. An RDF triple $tr = (s, p, o) \in TR$, where $TR = (s, p, o) \in (I) \times (I) \times (I \cup L)$. A set of RDF triples forms an *RDF graph D*. We assume additionally the existence of an infinite set V of variables disjoint from the above sets. As a notational convention, we will prefix variables with "?"(e.g., ?x, ?y).

Syntax of Graph Patterns. The definition of graph patterns is based on triple patterns. A triple pattern is defined as $(sp, pp, op) \in (I \cup V) \times (I \cup V) \times (I \cup L \cup V)$. The variables occurring in a graph pattern G are denoted as var(G).

Definition 1 (Graph Pattern)**.** A graph pattern is defined recursively as follows:

- A triple pattern is a graph pattern.
- If $G1$ and $G2$ are graph patterns, then (G1 AND G2), (G1 OPT G2), (G1 UNION G1), (G1 MINUS G2) are graph patterns.
- If G is a graph pattern and R is a filter expression, then (G FILTER R) is a graph pattern. A filter expression is constructed using elements of the sets $I \cup L \cup V$, logical connectives ($\neg, \wedge, \vee$), inequality symbols ($<, \leqslant, \geqslant, >$), equality symbol ($=$), plus other features (see [52] for a complete list).

Semantics of Graph Patterns. The semantics of graph pattern expressions are defined based on a partial mapping function μ, where $\mu : V \rightarrow TR$. For a triple pattern tp, we denote by $\mu(tp)$ the triple obtained by replacing the variables in tp according to μ. The domain of μ, $dom(\mu)$, is the subset of V where μ is defined.

Definition 2 (Evaluation of a Graph Pattern)**.** Let D be an RDF graph over TR. Mapping a graph pattern against D is defined using the function $[[.]]_D$, which takes a graph pattern expression and returns a set of mappings Ω.

Two mappings μ_1 and μ_2 are said to be compatible, denoted by $\mu_1 \sim \mu_2$ when, for all $x \in dom(\mu_1) \cap dom(\mu_2)$, it is the case that $\mu_1(x) = \mu_2(x)$. The evaluation of a compound graph pattern is defined as follows:
$[[G_1 \text{ AND } G_2]] = \Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \,|\, \mu_1 \in \Omega_1, \mu_2 \in \Omega_2, \mu_1 \sim \mu_2\}$
$[[G_1 \text{ UNION } G_2]] = \Omega_1 \cup \Omega_2 = \{\mu_1 \cup \mu_2 \,|\, \mu_1 \in \Omega_1 \cup \mu_2 \in \Omega_2, \mu_1 \sim \mu_2\}$
$[[G_1 \text{ OPT } G_2]] = \Omega_1 \mathbin{⟕} \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 / \Omega_2)$, where
$\Omega_1 / \Omega_2 = \{\mu | \mu \in \Omega_1 \wedge \nexists \mu' \in \Omega_2, \mu \sim \mu'\}$

3.2. GUCON

The GUCON framework is composed of two core components: *the state of affairs* and *the usage control policies*. The state of affairs is represented by an RDF graph, whereas the usage control policies consist of rules that are defined based on SPARQL graph patterns. These components are fundamental to the definition and implementation of the reasoning tasks of the framework, which are based on the formal semantics of SPARQL. In what follows, we first present the syntax of the state of affairs and the usage control policies (with a focus on obligations), followed by a formal definition of their semantics and the associated reasoning tasks.

3.2.1. Specification

The state of affairs captures both the relevant domain knowledge and observed events, serving as the foundation for evaluating and enforcing usage control policies. In this paper, we refer to the state of affairs as the knowledge base (KB), represented as an RDF graph that describes the set of actual knowledge, i.e., the current state of affairs.

Obligations guide the behavior of entities in their use of resources by specifying the actions they are required to perform under certain conditions. For instance, Scenario **S3** specifies *an obligation for Dr. Smith (entity) to sign (action) Alice's diagnosis report (resource).*

We assume three disjoint countably infinite sets, N (entity names), C (action names), and R (resource names), such that $N, C, R \subseteq I$. We define an *action* as a special RDF triple $(n, c, r) \in N \times C \times R$. In practice, we will allow variables to be present in the n, c, r positions, to allow generally applicable restrictions to be expressed. We refer to such a triple (i.e., a triple with variables) as an *action pattern*.

Definition 3 (Action Pattern)**.** An action pattern is a triple $(np, cp, rp) \in (N \cup V) \times (C \cup V) \times (R \cup V)$. We denote by $\mathcal{AP}$ the set of all action patterns.

Considering Scenario **S3** from our use case, the corresponding action pattern that expresses the signing action performed by a certain entity over a certain resouce, can be represented as follows[3]:

Example 1 (Action Pattern)**.**

$$(?doctor, :sign, ?diagnosisReport)$$

where *?doctor* is a variable ranging over entities in N, *:sign* is an action in C, and *?diagnosisReport* is a variable ranging over resources in R.

An obligation pattern can be defined as follows.

Definition 4 (Obligation Pattern)**.** Let **O** denote an *Obligation*. An *obligation pattern* is a statement of the form $\mathbf{O}a$, where $a \in \mathcal{AP}$.

In the context of an action pattern (np, cp, rp), $\mathbf{O}(np, cp, rp)$: indicates that an entity np is *obliged* (as indicated by the letter **O** before the action pattern) to perform an action cp over a resource rp. There may be other types of patterns (e.g., permission patterns, prohibition patterns), that are not supported in this work but can be easily expressed using GUCON. Following Scenario **S3**, the following obligation pattern states that a certain entity is obliged to sign a certain resource:

Example 2 (Obligation Pattern)**.**

$$\mathbf{O}(?doctor, : sign, ?diagnosisReport)$$

An obligation usually applies under specific conditions, giving rise to conditional obligation rules (e.g, *the obligation for Dr. Smith to sign a Alice's diagnosis report applies only under certain conditions. Dr. Smith must be a doctor, Alice must be a patient, a diagnosis report for Alice must already exist, etc.*). Using an obligation pattern, a graph pattern describing the *condition* of the rule, and the operator $\rightsquigarrow$ in-between, a conditional deontic rule (simply called an obligation rule) can be defined as follows.

Definition 5 (Obligation Rule)**.** An obligation rule is of the form: $cond \rightsquigarrow \mathbf{O}a$, where *cond* is a graph pattern, and $\mathbf{O}a$ is an obligation pattern. We denote by $\mathcal{R}$ the set of all obligation rules. For each rule, it is required that $var(a) \subseteq var(cond)$.

The obligation rule corresponding to Scenario **S3** can be formally represented using the HIC ontology. Intuitively, the rule expresses that whenever a doctor is responsible for a patient, and a diagnosis report is linked to that patient's hospital admission, then the doctor is obliged to sign the corresponding diagnosis report:

Example 3 (Obligation Rule)**.**

$$(?doctor, :type, :Doctor).$$
$$(?patient, :hasResponsibleDoctor, ?doctor).$$
$$(?admission, :type, :Admission).$$
$$(?admission, :hasPatient, ?patient).$$
$$(?diagnosisReport, :type, :DiagnosisReport).$$
$$(?diagnosisReport, :hasAdmission, ?admission).$$
$$\rightsquigarrow \mathbf{O}(?doctor, :sign, ?diagnosisReport)$$

[3]For all of our formal examples, we omit prefixes for simplicity.

An obligation policy consists of a collection of such obligation rules.

Definition 6 (Obligation Policy)**.** A set of obligation rules $R \subseteq \mathcal{R}$ is called an obligation policy.

3.2.2. Reasoning over GUCON Policies

To effectively perform the reasoning tasks in our framework, it is important that we have the ability to reason about the rules governing obligation policies. This is primarily accomplished through the process of evaluating these rules against the KB, resulting in the identification of *active rules*. Active rules are characterized by having a *satisfied condition*, ensuring their applicability in the given KB. In particular, an obligation of the form $cond \rightsquigarrow \mathbf{O}a$, can be read as follows: If the condition ($cond$) is satisfied by the KB, then the obligation pattern ($\mathbf{O}a$) must also hold. The restriction $var(a) \subseteq var(cond)$ ensures that all variables appearing in the obligation are already bound by the condition. Without this restriction, the model would implicitly generate infinitely many rules, which would render it impractical.

Definition 7 (Satisfied Condition)**.** Let K be a KB, R an obligation policy, and $r = cond \rightsquigarrow \mathbf{O}a \in R$ an obligation rule. A condition $cond$ is satisfied for μ, denoted by $K \rhd cond$, if and only if there exists a mapping μ such that $\mu \in [[cond]]_K$.

Note that multiple mappings may be used to satisfy a given rule.

Definition 8 (Active Obligation Rule)**.** An obligation rule $r = cond \rightsquigarrow \mathbf{O}a \in R$ is active for a mapping μ, if and only if $cond$ is satisfied for $\mu \in [[cond]]_K$.

Note that a rule may be active for multiple mappings. Based on the definition of an active rule, a usage control policy R is called *active* if at least one of its rules is active. Otherwise, it is called *inactive*.

Based on the defined formal semantics, different reasoning tasks are envisioned such as compliance and requirements checking[4]. Compliance checking aims to identify any breaches or non-compliant behavior of an entity, thereby ensuring the proper and secure operation of the system. In the context of a KB, compliance checking involves verifying that the stored facts and usage traces adhere to predefined policies. The criteria for determining whether a KB is compliant with a given policy can vary depending on the specific rule being considered, as well as the KB and mappings used to interpret that rule.

Definition 9 (KB Compliance Against an Obligation)**.** Given a KB K and an obligation rule $r = cond \rightsquigarrow \mathbf{O}a$, we say that K complies with r, denoted by $K \rhd r$, if and only if $\mu(a) \in K$ whenever $\mu \in [[cond]]_K$.

On the other hand, requirements checking is a task that enables a system to query a policy and receive information regarding the set of active obligations given a KB K, denoted by R_{active}.

Definition 10 (Requirements Checking)**.** Given a KB K and a set of rules R, the set of active rules R_{active} is defined as follows: $R_{active} = \{\mu(r) | r \in R, \text{r is active for } \mu\}$.

Intuitively, R_{active} contains the concrete rule instances that are triggered by the current state of the KB, obtained by instantiating the general rules R with substitutions that satisfy their conditions.

4. Formal Semantics of Temporal GUCON Obligations

While GUCON enables the specification of various data usage conditions through graph patterns, incorporating constraints like temporal restrictions requires extending the rule's deontic pattern. This extension allows for the expression of obligations with a temporal dimension, such as deleting data within 30 days of a deletion request or after a period of five years from the time of acquisition. Building upon previous work in temporal knowledge representation [23], we extend the GUCON model of obligation rules to encompass temporal properties. Consequently, the KB is also extended to represent these temporal attributes.

[4]Note that these tasks were defined in the original GUCON paper in the context of a specific entity n. For the sake of generalization and simplicity, we redefine these tasks in a general sense, independent of any entity.

4.1. Temporal Knowledge Base

As part of our extension, we include the notion of an event that presents an action executed at a specific time.

Event & Event Pattern. When an action (n, c, r) is executed at a time t_{exec}, then an event (n, c, r, t_{exec}) is created in the KB. An event is a tuple $(n, c, r, t_{exec}) \in (N \times C \times R \times T)$. Similarly, an event pattern is a tuple $(np, cp, rp, tp_{exec}) \in (N \cup V) \times (C \cup V) \times (R \cup V) \times (T \cup V)$.

Our KB stores two kinds of triples: (1) facts that are assumed to be time-invariant in our model. For example, *Dr. Smith is a doctor*. Although such facts may evolve over time (e.g., Dr. Smith's medical license might be revoked), for simplicity, we exclude their temporal aspects. These facts are stored in the Data Knowledge Base (DKB), which is represented as an RDF graph. (2) Events that capture actions occurring at specific points in time, such as *a doctor shares a patient record at time t*. These are stored in the Event Knowledge Base (EKB), represented as a temporal graph. Both the DKB and the EKB are part of the KB. Building on the definition of temporal graphs proposed by Gutierrez et al. [23], an EKB is defined as follows.

Definition 11 (Event Knowledge Base)**.** An event is a tuple (n, c, r, t_{exec}). An EKB is a finite set of events.

An example of an event corresponding to Scenario **S3** is Dr. Smith signing Alice's diagnosis report, which can be expressed as follows:

Example 4 (Event)**.**

(:dr-angelika-smith, :sign, :diagnosis-report-alice-waltz-2025-07-15, "2025-07-20T12:30"*xsd:dateTime*)

Given an EKB K^e, we can define a snapshot of K^e at time t.

Definition 12 (Event Knowledge Base Snapshot)**.** Given a time t, a snapshot of an EKB, denoted as K^e_t, includes only events that happened before t: $K^e_t = \{(n, c, r) | \exists t_{exec}, (n, c, r, t_{exec}) \in K^e \text{ and } t_{exec} <= t\}$.

Intuitively, an EKB snapshot at time t captures the state of the KB by including only those events that have already occurred up to t. Given a DKB K^s and a snapshot of an EKB K^e_t, the full snapshot of the KB at t is the union: $K_t = K^s \cup K^e_t$.

Consider the following snapshot from the hopsital KG descrbing facts and events up until August, 21st, 2025 at 10 AM:

Example 5 (Event Knowledge Base Snapshot)**.**

(:*dr-angelika-smith*, :*type*, :*Doctor*).
(:*patient-alice-waltz*, :*type*, :*Patient*).
(:*hasResponsibleDoctor*, :*type*, :*dr-angelika-smith*).
(:*admission-alice-waltz*-2025-07-15, :*type*, :*Admission*).
(:*admission-alice-waltz*-2025-07-15, :*hasActualAdmissionStartDate*, "2025-07-15T09:30"*xsd:dateTime*).
(:*admission-alice-waltz*-2025-07-15, :*hasActualAdmissionEndDate*, "2025-07-20T10:30"*xsd:dateTime*).
(:*admission-alice-waltz*-2025-07-15, :*hasPatient*, :*aliceWaltz*).
(:*diagnosis-report-alice-waltz*-2025-07-15, :*type*, :*DiagnosisReport*).
(:*diagnosis-report-alice-waltz*-2025-07-15, :*hasAdmission*, :*admission-alice-waltz*-2025-07-15).
(:dr-angelika-smith, :sign, :diagnosis-report-alice-waltz-2025-07-15, "2025-07-20T12:30"*xsd:dateTime*)

4.2. Temporal Obligation Rules

To enable GUCON obligations to account for temporal properties, we enhance the deontic actions within the rules by incorporating temporal attributes into both actions and action patterns.

Extended Action & Action Pattern. We extend the action pattern of an obligation with meta-properties describing the start time and deadline of the obligation. The start time of an obligation t_{start} presents the time when the obligation starts being applicable, whereas the deadline $t_{deadline}$ presents the time by which the obligation must be fulfilled. Scenario **S3** illustrates *the obligation for Dr. Smith to sign Alice's diagnosis report within 12 hours of her actual hospitalization end date*. The start time refers to the actual hospitalization end date, and the deadline falls exactly 12 hours after. In some cases, t_{start} or $t_{deadline}$ can be undefined. However, both t_{start} and $t_{deadline}$ cannot be undefined at the same time. When t_{start} is undefined; the obligation expresses an action that needs to be executed before a certain deadline but without any specific starting point; in other words, an obligation that is in force until it expires at its deadline. Scenario **S2** illustrates *Alice's obligation to sign a discharge form before the scheduled end of her hospitalization*. The undefined t_{start} is represented using the symbol $-\infty$. When $t_{deadline}$ is undefined, i.e., when the obligation has no expiration date, the obligation expresses an action that needs to be executed after a certain point of time marked by the start time. Scenario **S1** shows *the doctor's obligation to share a treatment with Alice after her lab results are ready*. In this case, the undefined $t_{deadline}$ is represented using the symbol ∞. We assume the set T presenting all literals that represent time, including ∞ and $-\infty$. An extended action is a tuple $(n, c, r, t_{start}, t_{deadline}) \in (N \times C \times R \times T \times T)$; with the additional constraint: $(t_{start} = -\infty) \oplus (t_{deadline} = \infty)$, i.e., either t_{start} or $t_{deadline}$ are defined as specific time points[5].

Definition 13 (Extended Action Pattern)**.** An extended action pattern is a tuple $(np, cp, rp, tp_{start}, tp_{deadline}) \in (N \cup V) \times (C \cup V) \times (R \cup V) \times (T \cup V) \times (T \cup V)$. We denote by $\mathcal{EAP}$ the set of all extended action patterns.

Considering Scenario **S3**, the corresponding extended action pattern indicating the signing action within a specific time frame can be represented as follows:

Example 6 (Extended Action Pattern)**.**

$$(?doctor, :sign, ?diagnosisReport, ?tp_{start}, ?tp_{deadline})$$

where $?tp_{start}$ and $?tp_{deadline}$ are variables ranging over all time literals in T.

Given the extended action patterns, an obligation pattern is also extended as follows.

Definition 14 (Extended Obligation Pattern)**.** An *extended obligation pattern* is of the form **O***ea*, where $ea \in \mathcal{EAP}$.

Following Scenario **S3**, the following extended obligation rule indicates the obligation of performing the signing action within a specific time frame:

Example 7 (Extended Obligation Pattern)**.**

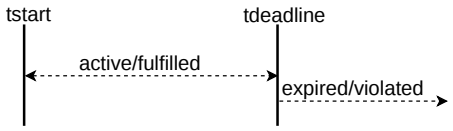
$$\mathbf{O}(?doctor, :sign, ?diagnosisReport, ?tp_{start}, ?tp_{deadline})$$

Similarly, an obligation rule is extended as follows.

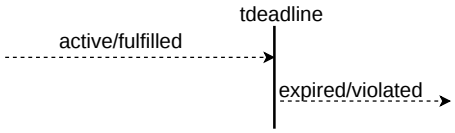
Definition 15 (Extended Obligation Rule)**.** An extended obligation rule is of the form: $cond \rightsquigarrow \mathbf{O}ea$, where *cond* is a graph pattern, and **O***ea* is an extended deontic pattern. We denote by $\mathcal{EO}$ the set of all extended obligation rules.

Building on Scenario **S3**, the following obligation rule formalizes the requirement to perform the signing action within a specific time frame, where the start time corresponds to the actual admission end date of the patient and the deadline is defined as 12 hours after this point:

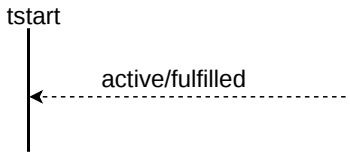
[5]Note that $\oplus$ represents the exclusive disjunction (XOR).



(a) With only deadline defined, an obligation is active when its condition holds within the interval; it is fulfilled if executed within the interval, expired if the deadline has passed, and violated if it remains unperformed after expiration.



(b) With an only defined deadline, an obligation is active when its condition holds before the deadline; it is fulfilled if executed up before the deadline, expired if the deadline has passed, and violated if it remains unperformed after expiration.



(c) With an only start time defined, an obligation is active when its condition holds after the start time; it is fulfilled if executed after the start time.

Figure 2. States of Obligations with the dotted arrows representing time.

Example 8 (Extended Obligation Rule)**.**

$(?doctor, :type, :Doctor).$
$(?patient, :hasResponsibleDoctor, ?doctor).$
$(?admission, :type, :Admission).$
$(?admission, :hasPatient, ?patient).$
$(?admission, :hasActualAdmissionEndDate, ?actualAdmissionEndDate).$
$(?diagnosisReport, :, :DiagnosisReport).$
$(?diagnosisReport, :hasAdmission, ?admission).$
$BIND(?actualAdmissionEndDateAS\,?tp_{start}).$
$BIND(?startTime + "PT12H"xsd:durationAS\,?tp_{deadline})$
$\rightsquigarrow \mathbf{O}(?doctor, :sign, ?diagnosisReport, ?tp_{start}, ?tp_{deadline})$

4.3. Reasoning Tasks

Building upon the extension of GUCON to incorporate temporal properties, the different reasoning tasks such as requirements and compliance checking can be further refined. In particular, requirements checking can address queries like identifying fulfilled, expired, or violated obligations at a time t, commonly referred to as obligation states [20]. Generally, given a snapshot K_t of a KB at time t, an extended obligation rule of the general form $cond \rightsquigarrow \mathbf{O}(np, cp, rp, tp_{start}, tp_{deadline})$, the obligation rule can either be *active*, *fulfilled*, *violated*, *expired*, or *not satisfied*. Figure 2 illustrates the states of obligations with regard to the start time and/or deadline. The different states of obligations depend on the identification of rules such as the condition is satisfied given a snapshot of a KB at a time t. For what follows, we assume the presence of an obligation OR of the form $cond \rightsquigarrow \mathbf{O}(np, cp, rp, tp_{start}, tp_{deadline})$ and a snapshot K_t of a KB at time t. We extend the definition of satisfied condition by including the time aspect as follows.

Definition 16 (Satisfied condition)**.** A condition $cond$ is satisfied for μ given a snapshot K_t, denoted by $K_t \rhd cond$, if and only if there exists a mapping μ such that $\mu \in [[cond]]_{K_t}$.

Given the snapshot presented in Example 5, the obligation rule in Example 8 is deemed satisfied since its condition can be directly matched to the snapshot and the mapped rule showing the actual start time `2025-07-20T10:30` and deadline `2025-07-20T 22:30:0` of the obligation is described as follows:

Example 9 (Satisfied condition)**.**

(:*dr-angelika-smith*, :*type*, :*Doctor*).
(:*patient-alice-waltz*, :*hasResponsibleDoctor*, ?*doctor*).
(*admission-alice-waltz*-2025-07-1, :*type*, :*Admission*).
(*admission-alice-waltz*-2025-07-1, :*hasPatient*, :*patient-alice-waltz*).
(*admission-alice-waltz*-2025-07-1, :*hasActualAdmissionEndDate*, "`2025-07-20T10:30`"*xsd*:*dateTime*).
(:*diagnosis-report-alice-waltz*-2025-07-15, :*type*, :*DiagnosisReport*).
(:*diagnosis-report-alice-waltz*-2025-07-15, :*hasAdmission*, "`2025-07-20T10:30`"*xsd*:*dateTime*).
$\rightsquigarrow$ **O**(`:dr-angelika-smith, :sign, :diagnosis-report-alice-waltz-2025-07-15,` "`2025-07-20T10:30`"*xsd*:*dateTime*, "`2025-07-20T22:30`"*xsd*:*dateTime*)

An active obligation is an obligation whose condition is satisfied and has not yet expired, i.e., it remains enforceable for a specified period. If the start time or deadline is undefined, the obligation still qualifies as active as long as at least one of these time points is specified.

Definition 17 (Active Obligation)**.** An obligation *OR* is active for a mapping μ at time t, if and only if *cond* is satisfied for μ at t and $\mu(tp_{start}) <= t <= \mu(tp_{deadline})$. We denote by $\mathcal{AO}$ the set of active obligations at a time t.

Example 10 (Active Obligation)**.** Given the snapshot in Example 5, the rule in Example 8 is classified as not active as the time of the snapshot (`2025-08-21T10:00`) is outside of the obligation timeframe (between `2025-07-20T10:30` and `2025-07-20T 22:30:0`),

A fulfilled obligation requires the corresponding action to be executed within the designated time frame. If either the start time or the deadline is undefined, the obligation is still considered fulfilled as long as at least one of these time points is specified.

Definition 18 (Fulfilled Obligation)**.** An obligation *OR* is fulfilled at time t if only if *cond* is satisfied for a mapping μ at t, $\exists t_{exec}$, such that $(\mu(np, cp, rp), t_{exec}) \in K_t$ and $\mu(tp_{start}) <= t_{exec} <= \mu(tp_{deadline})$. We denote by $\mathcal{FO}$ the set of all fulfilled obligations at time t.

Example 11 (Fulfilled Obligation)**.** Given the snapshot in Example 5, the rule in Example 8 is considered fulfilled as the obligation (`:dr-angelika-smith, :sign, :diagnosis-report-alice-waltz-2025-07-15`) is executed at `2025-07-20T12:30`, which is within the obligation timeframe (`2025-07-20T10:30` and `2025-07-20T 22:30:0`),

A violation of obligation represents an action that was not performed while the deadline has passed. In case the deadline is undefined, an obligation is never violated.

Definition 19 (Violated Obligation)**.** An obligation *OR* is violated at time t if and only if *cond* is satisfied for a mapping μ at t, $\nexists t_{exec}$, such as $(\mu(np, cp, rp), t_{exec}) \in K_t$ and $t > \mu(tp_{deadline})$. We denote by $\mathcal{VO}$the set of violated obligations at a time t.

Example 12 (Violated Obligation)**.** Given the snapshot in Example 5, the rule in Example 8, is deemed not violated because it was already fulfilled.

An expired obligation is an obligation that is no longer enforceable passed a specific deadline. In case the deadline is not defined, an obligation would never expire.

Definition 20 (Expired Obligation)**.** An obligation *OR* is expired at time t if and only if *cond* is satisfied for a mapping μ at t and $t > \mu(tp_{deadline})$. We denote by $\mathcal{EO}$ the set of expired obligations at time t.

Example 13 (Expired Obligation)**.** Given the snapshot in Example 5, the rule in Example 8 is considered expired, because the time of the snapshot (`2025-08-21T10:00`) is passed the obligation deadline `2025-07-20T22:30:0`.

A not satisfied obligation is an obligation that has not yet been fulfilled, but the deadline has not passed. If either the start time or the deadline is undefined, the obligation is still considered not satisfied as long as at least one of these time points is specified.

Definition 21 (Not Satisfied Obligation)**.** An obligation *OR* is not satisfied at time t if only if *OR* is active for a mapping μ, $\nexists t_{exec}$, such as $(\mu(np, cp, rp), t_{exec}) \in K_t$, and $\mu(tp_{start}) <= t <= \mu(tp_{deadline})$. We denote by $\mathcal{NSO}$ the set of not satisfied obligations at time t.

Example 14 (Not Satisfied Obligation)**.** Given the snapshot in Example 5, we can not infer that the rule in Example 8 is not satisfied because it was already fulfilled and is not active anymore.

Compliance checking evaluates whether the KB reflects the possible states of obligations. A KB is compliant if no expired obligations are shown as violated.

Definition 22 (Compliant Knowledge Base at time t)**.** A KB K is compliant at time t if and only if $\mathcal{VO} = \emptyset$.

Example 15 (Compliant Knowledge Base at time t)**.** Given the rule in Example 8, we can consider that the snapshot in Example 5 is compliant because there are no violated obligations at the time `2025-08-21T10:00`.

5. Implementation: An obligation Manager for Temporal GUCON Obligations

In this section, we present the syntax used to express temporal GUCON. We also provide a proof of concept implementation that supports the reasoning tasks described in Section 4, along with the ontologies employed to support the proposed solution[6].

5.1. Implementing Temporal GUCON using RDF 1.2 and SPARQL 1.2

In what follows, we present an overview of RDF 1.2 [27, 48] and SPARQL 1.2 [28] and its syntactical representation[7]. Then, we demonstrate how to represent our GUCON temporal obligations using RDF 1.2 and SPARQL 1.2.

5.1.1. RDF 1.2 and SPARQL 1.2

RDF 1.2 extends RDF by introducing *triple terms*, which allow triples to appear as the object of other triples. We begin by defining RDF 1.2 triple patterns and triple terms. Note that here again blank nodes are omitted for simplicity.

Definition 23 (RDF 1.2 Triple and Triple Terms)**.** Let RT be the set of RDF triples, defined as follows:

- If $s \in I$, $p \in I$, and $o \in (I \cup L)$, then $(s, p, o) \in RT$.
- If $s \in I$, $p \in I$, and $o \in RT$, then $(s, p, o) \in RT$.

A *triple term* is an RDF triple considered as an RDF term. In particular, the definition of a triple is now recursive. That is, a triple can itself have an object component, which is another triple. The semantics of SPARQL 1.2 graph patterns extend those of standard SPARQL graph patterns while preserving backward compatibility. In particular, solution mappings may bind variables to RDF 1.2 terms, including triple terms.

[6] Throughout the paper, we use the prefixes *gucon*, *hc*, and *ucp* to denote respectively the GUCON core, the Hospital Inpatient Care (HIC) and the Usage Control Policy (UCP) vocabularies.

[7] Throughout this section, all examples illustrating RDF 1.2 and SPARQL 1.2 are presented using the Turtle 1.2 serialization format.

5.1.2. Syntax for Reifying Triples.

The main use for triple terms in RDF 1.2 is reification. In particular, using triple terms enables concise representation of statement-level metadata without relying solely on standard RDF reification. A triple term is represented as $\ll\ (spo)\ \gg$ and can serve as the object of a triple using the new predicate *rdf:reifies*. The subject of such a triple is called a *reifier*. The reifier can be used to assert facts about an occurrence of statements denoted by the triple term itself. We call such a nested triple a *reifying triple*. Assume a triple term $\ll$ (:*dr-angelika-smith* :*married* :*dr-brandon-smith*) $\gg$, a reifying triple can be expressed as:

$$:wlr1\ rdf{:}reifies\ \ll (:dr\text{-}angelika\text{-}smith\ :married\ :dr\text{-}brandon\text{-}smith) \gg\ . \quad (1)$$

Such as :*wlr*1 is the reifier. Knowing that reification allows us to attach different metadata to the same statement, RDF 1.2 offers a shorthand for the reifiyng triple shown in (1), that can be written as << :*dr-angelika-smith* :*married* :*dr-brandon-smith* $\sim$:*wlr*1 >>. Such reifying triples can be placed in the subject or object position of an RDF triple. A reifying triple in the subject position can be expressed as follows:

$$\ll :dr\text{-}angelika\text{-}smith\ :married\ :dr\text{-}brandon\text{-}smith \sim :wlr1 \gg\ :certifiedBy\ :marriageCertificate\text{-}2024\text{-}88. \quad (2)$$

Where :*certifiedBy* and :*marriageCertificate*-2024-88 represent respectively a meta-property and a meta-object that describe provenance information concerning the triple in the subject position. The reifier :*wlr*1 can be omited from the triple pattern, as shown below:

$$\ll :\ dr\text{-}angelika\text{-}smith\ :\ married\ :\ dr\text{-}brandon\text{-}smith \gg\ :certifiedBy\ :marriageCertificate\text{-}2024\text{-}88. \quad (3)$$

Additionally, one can simply use the reifier alone in the subject position. To illustrate how reifying triples can be queried in SPARQL 1.2, the same statement-level pattern can be directly used in graph patterns. For example, the reified statement expressed in (2) and (3) can be retrieved as follows:

$$SELECT*$$
$$WHERE$$
$$\{\ \ll ?s\ :married\ ?o \gg\ :certifiedBy\ ?cert.\ \}$$

5.1.3. Syntax for Temporal GUCON

In order to represent the temporal aspects of GUCON rules and the temporal KB, we adopt SPARQL 1.2 and RDF 1.2 that enable the use of reifying triples to annotate actions and action patterns with temporal metadata. We do not use reifiers to identify action patterns; instead, we use the alternative syntax (3). Note that reifiers are important when we want to associate the same triple with two different reifiers, in order to associate different metadata to each, but this is not a necessary feature in our case, because actions are anyway unique and will be assigned a unique set of metadata.

Temporal Obligation Rules. Temporal rules are enriched with the temporal predicates *gucon:startTime* and *gucon:deadline*, which respectively denote the start time and deadline associated with an obligation. An extended action pattern is expressed as a reifying triple pattern:

$$\ll ?n\ ?a\ ?r \gg\ gucon{:}startTime\ tp_{\text{start}};\ gucon{:}deadline\ tp_{\text{deadline}}.$$

where $?n\ ?a\ ?r$ represents a triple pattern and tp_{start}, tp_{deadline} map to literals of type *xsd:dateTime*. Rule conditions may also contain reifying triple patterns.

Algorithm 1: Evaluate States of Policy Rules at Time t

Input : Policy P, Knowledge Base K, Time t
Output: Object `states`

```
Function GET_OBLIGATIONS_STATES(P, K, t):
    K_t ← GET_SNAPSHOT(K, t)
    states ← new ObligationStates()
    foreach rule r ∈ P do
        ω ← GET_MAPPINGS(r.condition, K_t)
        if ω ≠ ∅ then
            foreach binding μ ∈ ω do
                mappedRule ← μ(r)
                if mappedRule.startTime ⩽ t ⩽ mappedRule.deadline then
                    states.activeO ← states.activeO ∪ {mappedRule}
                    if mappedRule.executionTime = null then
                        states.notSatisifedO ← states.notSatisifedO ∪ {mappedRule}
                    end
                end
                else if t > mappedRule.deadline then
                    states.expiredO ← states.expiredO ∪ {mappedRule}
                    if mappedRule.executionTime = null or mappedRule.executionTime >
                     mappedRule.deadline then
                        states.violatedO ← states.violatedO ∪ {mappedRule}
                    end
                end
                else if mappedRule.executionTime ≠ null then
                    if mappedRule.startTime ⩽ mappedRule.executionTime ⩽ mappedRule.deadline then
                        states.fulfilledO ← states.fulfilledO ∪ {mappedRule}
                    end
                end
            end
        end
    end
    return states
```

Temporal Knowledge Base. Temporal facts in the KB are modeled using the predicate *gucon:executionTime*, which links an executed action to its *xsd:dateTime* value. This is expressed as:

$$\ll n\ a\ r \gg \ \textit{gucon:executionTime}\ t_{\text{exec}}.$$

Where t_{exec} is a value of type *xsd:dateTime*. The KB is then capable of expressing both static and temporal information.

Rule Evaluation with Temporal Events. Given a rule of the form:

$$\textit{cond} \ \rightsquigarrow\ \mathbf{O}\left\{\ll ?n\ ?a\ ?r \gg \ \textit{gucon:startTime}\ tp_{\textit{start}};\ \textit{gucon:deadline}\ tp_{\text{deadline}}\right\}.$$

Algorithm 2: Check Compliance of a Knowledge Base at Time t

Input : Policy P, Knowledge Base K, Time t
Output: Compliance status

```
1 Function CHECK_COMPLIANCE(P, K, t):
2     states ← GET_OBLIGATIONS_STATES(P, K, t)
3     if states.violatedO ≠ ∅ then
4         return NON_COMPLIANT
5     end
6     return COMPLIANT
```

To retrieve possible execution times from the KB, the rule is *augmented* with an optional temporal binding as follows:

$$\textit{cond}\ \texttt{OPTIONAL}\,\{\ll ?n\ ?a\ ?r \gg\ \textit{gucon:executionTime}\ tp_{\text{exec}}\}$$
$$\rightsquigarrow \mathbf{O}\,\{\ll ?n\ ?a\ ?r \gg\ \textit{gucon:startTime}\ tp_{\text{start}};\ \textit{gucon:deadline}\ tp_{\text{deadline}}\}\,.$$

The use of `OPTIONAL` ensures that if an execution time exists in the KB for the action pattern, it is retrieved; otherwise, the rule can still be processed using only the other bindings. This enriched rule is then matched against the KB, allowing the values for start time, deadline, and (if available) execution time to be extracted and used in further reasoning tasks. Listing 1 shows an instantiation of Scenario **S3** using SPARQL 1.2. While *?startTime* and *?deadline* are not always explicitly provided, they can be computed dynamically using the `BIND` operator. These values are eventually derived from the actual start time stored in the KB and the duration specified in the policy. Scenarios **S1** and **S2** can be expressed in a similar way, using one of the temporal predicates.

5.2. System Architecture

In order to show a proof of concept of our defined semantics for temporal GUCON obligations, we developed a GUCON Obligation State Manager that implements the reasoning tasks defined in Section 4. The GUCON Obligation Manager is implemented in Java using Apache Jena 5.1.0. The manager expects three primary inputs: a temporal KB and a policy, both provided as Turtle files, and a time instant t, expressed using the W3C XML Schema Definition Language (XSD)[8].

The Obligation Manager comprises five core components: the *Knowledge Base Manager*, the *Rule Manager*, the *Obligation State Manager*, the *Compliance Checker*, and the *Report Generator*. The Knowledge Base Manager loads the KB from a Turtle file into Jena TDB2. The Rule Manager loads and maintains the rules in memory as Java objects, enabling their evaluation against the KB. Additionally, it is responsible for augmenting the rules with optional temporal bindings. The Obligation State Manager is responsible for reasoning over the states of obligations, while the Compliance Checker assesses the compliance of the KB. The Report Generator produces a compliance report detailing the state of each obligation and the overall compliance status.

The core logic of the Obligation State Manager and the Compliance Checker is implemented through the following key algorithms, which expect as input a GUCON policy P, a KB K and a time t. Algorithm 1 handles the management of obligation states in three main steps: i) It extracts a snapshot Kt of the KB K at time t (line 2); ii) For each rule, it determines whether the rule is satisfied in Kt by invoking the `GET_MAPPINGS` function, which evaluates the rule over the snapshot and returns a set of mappings that populate the rule with concrete values (lines 5–8); iii) Finally, the algorithm evaluates the state of each mapped obligation based on its start time, deadline, and the input time t (lines 9–25). For simplicity, we illustrate only the case in which both the start time and deadline are defined; for the other scenarios, the same reasoning is followed. While Algorithm 2 is responsible for checking the compliance of the KB. It assumes that the status of each rule has already been determined by invoking the function

[8] W3C XML Schema Definition Language (XSD), https://www.w3.org/TR/xmlschema11-2/

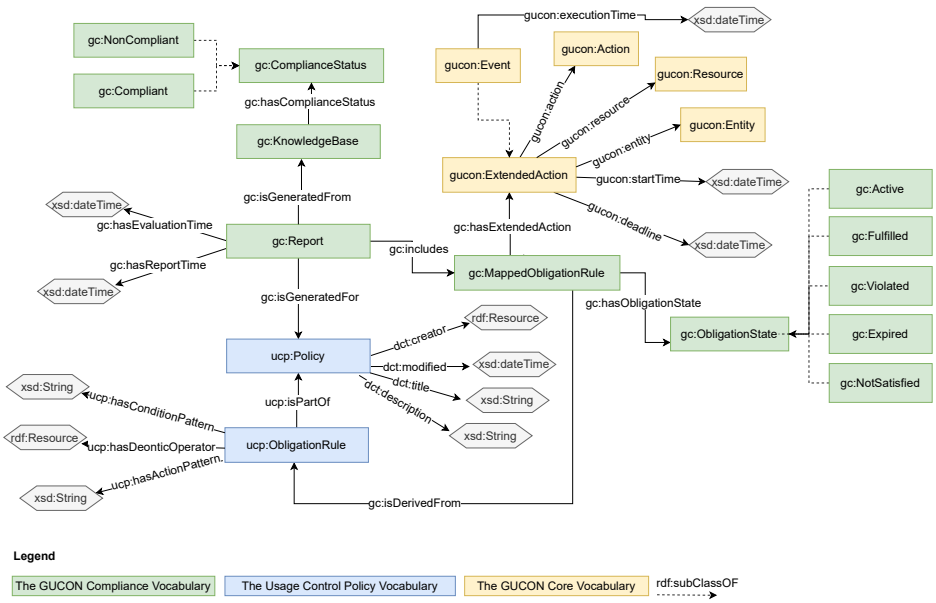


Figure 3. An ontology for a Compliance Report

in Algorithm 1 (line 2). If the set of violated rules is empty (line 3), then K is considered compliant. The final output indicates the compliance status of the input KB at a given time t.

5.3. *Ontology-Driven Representation for the GUCON Obligation State Manager*

In the following, we present the OWL-based vocabularies used to model the input and output of our Obligation State Manager. We also provide an RDF instantiation of our use case, demonstrating how the defined vocabularies are applied in practice.

5.3.1. *Core Ontologies Supporting GUCON*

To streamline the management of inputs and outputs within the GUCON Obligation Manager, we developed dedicated ontologies that encode both the GUCON Policy and the Compliance Report. In contrast, the KB integrates ontologies specific to the application domain, with a particular focus on vocabularies for temporal representation.

An RDF 1.2 Vocabulary for GUCON Policies. To ensure interoperability and facilitate automated processing, GUCON policies are represented in RDF 1.2. Figure 3 (highlighted in blue) presents the Usage Control Policy (UCP) vocabulary, which defines GUCON obligation rules through the properties *ucp:hasConditionPattern*, *ucp:hasActionPattern*, and *ucp:hasDeonticOperator*. The property *ucp:isPartOfPolicy* indicates the policy to which a given rule belongs. Furthermore, instances of *ucp:Policy* can be enriched with metadata, such as creation date, creator, and description, by reusing properties from the Data Catalog Vocabulary (DCAT)[9].

An RDF 1.2 Vocabulary for Compliance Reports. Figure 3 shows the ontology depicting a compliance report. A compliance report provides a detailed description of the states of mapped rules and the compliance status of the KB. The class *gc:Report* is generated for a *ucp:Policy* and associated with a *gc:KnowledgeBase*. The evaluation time of the report indicates the time t at which a snapshot of the input KB is taken, whereas the report time refers to the actual moment the report is generated. The compliance status of the KB is represented by the *gc:ComplianceStatus* class. The ontology also captures the relationship between mapped obligation rules and the original input rules. A *gc:MappedObligationRule* represents an obligation rule that has been instantiated with specific data from the

[9]Data Catalog Vocabulary (DCAT), https://www.w3.org/TR/vocab-dcat-3/

KB. It is linked to a *gucon:ExtendedAction*, which is composed entirely of constants. The state of each mapped rule is captured using the *gc:ObligationState* class. A *gucon:ExtendedAction*, connects to a *gucon:Entity*, a *gucon:Action*, and a *gucon:Resource*. This action is further characterized by the temporal properties *gucon:startTime* and *gucon:deadline*, indicating the start and deadline of the associated obligation. Additionally, the property *gucon:executionTime* is used to specify the exact time at which an event occurs. When an extended action is executed, it is represented as an instance of the *gucon:Event* class. The various ontologies can be found on our GitHub[10].

5.3.2. Use Case Instantiation

To illustrate the practical applicability of the proposed GUCON Compliance ontology and the Obligation State Manager, we model the use case described in Section 2 using the HIC ontology. Specifically, we demonstrate how the GUCON State Manager processes input data and generates compliance outcomes based on defined policies and observed actions with a focus on Scenario **S3**, as it involves both a start time and a deadline. Listing 2 shows the RDF encoding of Scenario **S3** following the UCP vocabulary.

```
{?doctor a hc:Doctor .

?patient hc:hasResponsibleDoctor ?doctor .

?admission a hc:Admission .
?admission hc:hasPatient ?patient .
?admission hc:hasActualAdmissionEndDate
    ?actualAdmissionEndDate .

?diagnosisReport a hc:DiagnosisReport .
?diagnosisReport hc:hasAdmission  ?admission .

BIND (?actualAdmissionEndDate AS ?startTime) .
BIND(?startTime +"PT12H"^^xsd:duration AS ?deadline)}
->
O
{
<<?doctor gucon:sign ?diagnosisReport >>
gucon:startTime ?startTime; gucon:deadline ?deadline.
}
```

Listing 1: Scenario 3 Modeled as a GUCON Rule

```
exp:rule-obligation-sign-diagnosis-report
    a ucp:ObligationRule;
    ucp:hasActionPattern
        "<<?doctor gucon:sign ?report>> ..."

ucp:hasConditionPattern
    "?patient rdf:type  hc:Patient ...";

ucp:hasDeonticOperator
    ucp:ObligationRule;

ucp:isPartOfPolicy
    exp:policy-obligation-sign-diagnosis-report.

exp:policy-obligation-sign-diagnosis-report
    a ucp:Policy ;
    dcat:description "an example policy" ;
    dcat:creator  ex:ines-akaichi ;
    dcat:modified
    "2025-07-20T10:30:00+02:00"^^xsd:dateTime .
```

Listing 2: RDF-Based Encoding of Scenario 3

Listing 3 provides a corresponding snapshot of the KB. We assume that when this KB is loaded into the system, it is assigned a unique IRI *ex:kb-trace-doctor-angelika-smith* for identification. The KB shows that the condition (lines 1-10) in Listing 1 is satisfied. Notably, the KB also stores the execution of the required action, signing the diagnosis report, using the predicate *gucon:hasExecutionTime*.

```
ex:doctor-angelika-smith a hc:Doctor ;
    hc:name "Angelika Smith" .

 ex:patient-alice-waltz a hc:Patient ;
    hc:name "Alice Waltz" ;
    hc:hasResponsibleDoctor  ex:doctor-angelika-smith .

ex:admission-alice-waltz-2025-07-15 a hc:Admission ;
    hc:hasActualAdmissionStartDate "2025-07-15T09:30:00+02:00"^^xsd:dateTime ;
    hc:hasActualAdmissionEndDate "2025-07-20T10:30:00+02:00"^^xsd:dateTime;
    hc:hasPatient ex:patient-alice-waltz .

ex:diagnosis-report-alice-waltz-2025-07-15  a hc:DiagnosisReport ;
    hc:hasAdmission ex:admission-alice-waltz-2025-07-15 .

<< ex:doctor-angelika-smith  gucon:sign ex:diagnosis-report-alice-waltz-2025-07-15 >>
    gucon:executionTime  "2025-07-20T12:30:00+02:00"^^xsd:dateTime;
```

Listing 3: Scenario 3 Corresponding Knowledge Base

[10]https://github.com/Ines-Akaichi/Temporal-GUCON/tree/main/ontologies

Based on the policy definition in Listing 2 and the KB content in Listing 3, Listing 4 shows an excerpt from the resulting compliance report. The report shows that the mapped obligation rule *instance-obligation-sing-diagnosis-form-01* is classified as both fulfilled and expired. It further specifies that the obligation's start time corresponds to the patient's hospitalization end date, as stated in the main policy rule, while the evaluation time occurs after the rule's deadline. At the time of evaluation, the KB confirms that the doctor *ex:kb-trace-doctor-angelika-smith* completed the required action, signing the diagnosis report, within the permissible time window defined by the obligation's start time and deadline. Therefore, the KB *exp:kb-trace-doctor-angelika-smith* is deemed fully compliant at the time of policy evaluation.

```
exp:instance/obligation-sign-diagnosis-report-01      rdf:type     ucp:MappedObligationRule;
     gucon:hasExtendedAction      exp:sign-alice-waltz-sign-diagnosis-report-2025-07-20   ;
     gucon:isDerivedFrom          exp:rule-obligation-sign-diagnosis-report    ;
     gucon:hasObligationState     gucon:FULFILLED, gucon:EXPIRED .

exp:sign-alice-waltz-sign-diagnosis-report-2025-07-20     rdf:type     gucon:Event ;
      gucon:hasEntity    ex:doctor-angelika-smith;
      gucon:hasAction <gucon:sign> ;
      gucon:hasResource   ex:diagnosis-report-alice-waltz-2025-07-15 ;
      gucon:hasStartTime   "2025-07-20T10:30:00+02:00"^^xsd:dateTime;
      gucon:hasDeadline    "2025-07-20T22:30:00+02:00"^^xsd:dateTime ;
      gucon:hasExecutionTime   "2025-07-20T12:30:00+02:00"^^xsd:dateTime .

exp:report-2025-07-20T11:01:00+02:00
     gucon:hasEvaluationTime   "2025-07-21T10:00:00+02:00"^^xsd:dateTime;
     gucon:hasReportTime    "2025-07-21T10:00:12:00+02:00"^^xsd:dateTime;
     gucon:isGeneratedFor       exp:policy-obligation-sign-diagnosis-report;
     gucon:isGeneratedFrom      exp:kb-trace-doctor-angelika-smith;
     gucon:includes             exp:instance-obligation-sign-diagnosis-report-01 .

exp:kb-trace-doctor-angelika-smith      gucon:hasComplianceStatus       gucon:COMPLIANT .
```

Listing 4: Scenario 3 Corresponding Compliance Report

5.4. Properties of the Extended GUCON Model

Inspired by previous work that applied Thörn's quality assessment criteria [61] to policy models [58], we use these criteria to describe the properties of our extended GUCON model. Thörn's model quality criteria provide a systematic framework for characterizing modeled artifacts and encompass the following factors: *changeability*, *reusability*, *formalness*, *correctness*, *mobility*, and *usability*.

Changeability. This criterion illustrates the model's ability to evolve alongside its applications while preserving its core purpose. A fundamental aspect of usage control is the capability to express and reason about temporal aspects, enabling the monitoring of obligations [47]. Other constraints, such as spatial, purpose-based, and event-driven restrictions are also important for capturing realistic usage control scenarios [29]. We have first addressed the temporal dimension, demonstrating how GUCON can represent and reason about temporal constraints. The same approach can be extended to incorporate other constraint types, such as spatial restrictions, by enriching the extended obligation action pattern with additional meta-properties. For example, spatial attributes can be captured through latitude and longitude values, resulting in an enhanced extended action pattern of the form: $(np, cp, rp, tp_{start}, tp_{deadline}, tp_{latitude}, tp_{longitude})$, which expresses an action that must be executed within a specific time frame at a certain location. SPARQL 1.2 is a particularly suitable option for encoding this extended action pattern, as it offers a concise representation without excessive verbosity.

Reusability. This refers to the ability to reuse the model (or parts of the model) across different use cases or applications. GUCON rules are designed to be general and adaptable, requiring only domain-specific information to represent particular scenarios. Our extended GUCON obligation model captures the core elements of an obligation: the entity, the resource, the actions, and their associated temporal aspects. By combining the model with a suitable ontology or vocabulary, it can be tailored to represent specific use cases. A broad spectrum of usage control scenarios can be envisioned, spanning domains such as healthcare, mobility, energy, and agriculture [1]. In our work, we

Table 1
Test Cases Representing the Hospital Inpatient Care Scenarios

Scenario	Case	Start Time	Deadline	Time	Execution Time	Expected States
S1	S11	yes	no	after or equal to start time	none	active, not satisfied
	S12	yes	no	after start time	after or equal to start time	active, fulfilled
S2	S21	no	yes	before or equal to deadline	none	active, not satisfied
	S22	no	yes	before or equal to deadline	before or equal to deadline	active, fulfilled
	S23	no	yes	after deadline	none	expired, violated
	S24	no	yes	after deadline	before or equal to deadline	expired, fulfilled
S3	S31	yes	yes	during interval	none	active, not satisfied
	S32	yes	yes	during interval	during interval	active, fulfilled
	S33	yes	yes	after deadline	none	expired, violated
	S34	yes	yes	after deadline	during interval	expired, fulfilled

developed the HIC ontology to model the medical use case described in Section 2. Both the ontology and the represented scenarios are publicly accessible in our GitHub repository[11]. Other ontologies could similarly be employed to capture domain knowledge in mobility [40], energy [7], or agriculture [32].

Formalness. This refers to the model's ability to be defined and managed in a formalized way. In our case, the GUCON model is grounded in the formal semantics of SPARQL graph patterns, which are based on model-theoretic semantics. This foundation facilitates the precise specification and management of the model's semantics, leveraging the well-established formal semantics underpinning SPARQL [49]. Our model employs operators such as `AND` and `UNION` to represent conjunctive and disjunctive conditions. The `OPT` operator functions similarly to the outer join in SQL, while `MINUS` allows the expression of negation in conditions (e.g., to identify all patients who did not sign the discharge form, one could write *?patient rdf:type hc:Patient* `MINUS` *?patient gucon:sign ?dischargeForm*. The `FILTER` operator is used to specify conditions on particular sub-elements of a triple. Additionally, assignment properties such as `BIND` can be used to compute derived values, particularly date-time values. For instance, in Scenario **S3**, we demonstrated how `BIND` can be used to calculate the deadline of an obligation.

Correctness. This factor concerns the model's effectiveness in capturing and addressing real-world requirements, particularly through the representation of domain-specific artifacts. We employ the developed HIC ontology to represent the use case scenarios described in Section 2. Furthermore, to assess the model's expected output with respect to obligation states, we design test cases for each scenario. These test cases ensure the coverage of our approach by considering all relevant expected states, which depend on the values of the start time, deadline, and execution time of the corresponding obligation, together with the given input time t. Table 1 presents the developed test cases together with their expected states. The corresponding KBs and rules for each scenario are made available on our GitHub[12].

Mobility. This criterion addresses the model's interoperability and its potential for integration with other systems. In this paper, we demonstrated how the extended GUCON model can be represented using the SPARQL 1.2 syntax. Beyond SPARQL 1.2, our model can also be instantiated with other widely adopted W3C languages, such as the Shapes Constraint Language (SHACL)[13] and the Web Ontology Language (OWL)[14]. SHACL was originally designed for validating RDF graphs, but it has also attracted attention for expressing policies [53, 54], particularly through its advanced features[15]. For example, a usage control obligation can be represented in SHACL as a `sh:SPARQLRule`, where the `sh:construct` property defines a construct query. Using this representation, the template clause specifies the action pattern, while the `where` clause can be directly mapped to the condition of the obligation. Similarly, OWL has been extensively used to express policies in the Semantic Web [37, 58]. In OWL, an obligation can capture the entity, resource, and action using `owl:equivalentClass` together with

[11] https://github.com/Ines-Akaichi/Temporal-GUCON/tree/main/use-case
[12] https://github.com/Ines-Akaichi/Temporal-GUCON/tree/main/test-cases
[13] SHACL, https://www.w3.org/TR/shacl/
[14] OWL, https://www.w3.org/TR/owl2-primer/
[15] SHACL advanced features, https://www.w3.org/TR/shacl-af/

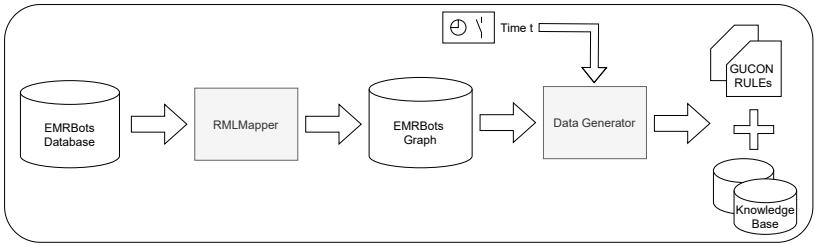


Figure 4. Data Generation Pipeline

`owl:intersectionOf` (to represent conjunctions, i.e., logical AND). These elements of a rule can be recursively described using `owl:hasValue` or `owl:allValuesFrom`. The logical `UNION` operator can be expressed via `owl:unionOf`, while `owl:complementOf` can be used as a workaround for expressing optional patterns (`OPT`) and for enabling soft negation to represent `MINUS` operations. Although OWL does not natively support the `FILTER` operator, constraints on property values can still be expressed using OWL restrictions and property assertions. However, OWL does not have a direct equivalent of `BIND`. Instead, the start time and deadline of an obligation can be directly assigned as fixed values in the policy using data property assertions or class-level restrictions. In our GitHub repository[16], we provide instantiations of Scenario **3** using SHACL and OWL. One avenue worth exploring is how SHACL implementations and OWL reasoners can be leveraged to carry out our defined reasoning tasks.

Usability. This criterion concerns the aspects of the model that affect a user's ability to define and work with it effectively. In our approach, the syntax of GUCON rules is intentionally aligned with existing Semantic Web standards: both the condition and the extended action of a rule follow the SPARQL 1.2 syntax, which benefits users already familiar with RDF and SPARQL-based technologies. This design choice also facilitates integration with existing Semantic Web tools and infrastructures. However, we acknowledge that these considerations alone are not sufficient to fully demonstrate usability in practice. A comprehensive usability evaluation would require empirical validation involving potential users, such as Semantic Web engineers, data governance practitioners, or policy administrators, to assess aspects such as learnability, intuitiveness, and ease of use [35]. Conducting user studies and developing higher-level policy authoring interfaces or dedicated editors therefore constitute important directions for future work.

6. Evaluation

In this section, we evaluate the performance and scalability of the GUCON State Manager following the benchmark design inspired by the methodology developed within the HOBBIT 2020 project [57]. In particular, we design our evaluation based on two key aspects: i) choosing appropriate data to feed our prototype and defining the corresponding data generation process; ii) designing benchmarking tasks based on choke point analysis. A choke-point analysis identifies important technical challenges that should be evaluated in the context of the underlying architectural solution. These tasks are evaluated using specific key performance indicators primarily execution time.

6.1. Data Generation

For our experimental evaluation, we use a dataset that closely reflects our target use case. The EMRBots dataset [38] provides synthetic electronic medical records for 100 patients. It emulates the structure and content of real-world medical databases, including patient admissions, demographics, socioeconomic information, lab results, and more. The EMRBots dataset encompasses records for 100 patients, including 372 admissions, 372 diagnosis reports, and a total of 110,107 lab test entries. The RDF graph derived from the EMRBots dataset forms the basis for

[16]https://github.com/Ines-Akaichi/Temporal-GUCON/tree/main/instantiation

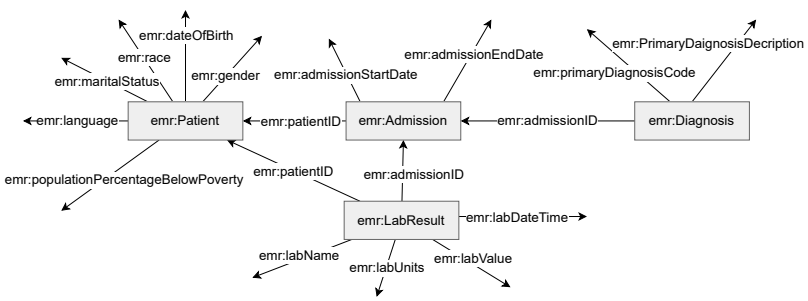


Figure 5. The EMRBots Schema

generating GUCON rules and constructing KBs, both of which are employed by our GUCON Obligation Manager. The data generation pipeline is illustrated in Figure 4. Originally provided as a relational database in plain text files, the EMRBots dataset was transformed into an RDF graph using the RDF Mapping Language (RML) and the RMLMapper tool [34]. The schema of the resulting EMRBots graph is shown in Figure 5. This RDF graph is then processed by a Data Generator, which produces GUCON rules and the associated KB. The rules are generated using a predefined template, as shown in Listing 5.

```
{?e    <random predicate>    ?r . ?e    <random predicate>    ?v .
BIND (<random startTime> as ?startTime)
BIND (<random deadline> as ?deadline)}
-> O <<?e <gucon:random action>  ?r>> gucon:startTime ?startTime  gucon:deadline ?deadline
```

Listing 5: GUCON Template Rule

The template consists of two triple patterns that incorporate two distinct predicates selected from the EMRBots graph. Up to 56 unique rules are generated based on these predicate combinations. Additionally, the template uses BIND to bind the start time and the deadline of the rule to generated date-times. The resulting KB comprises two parts: the DKB, which consists of the EMRBots graph itself, and the AKB, which includes generated events corresponding to the action triples defined by the rules. The full KB comprises around 2545342 triples. The Data Generator also requires an input timestamp to ensure that the temporal components of both the rules and execution traces are generated in a meaningful and coherent manner.

Selectivity. The 56 generated rules differ in the number of matches they return. To make our experiments more meaningful, we first analyzed the selectivity of the generated rules against the DKB and grouped them according to the number of matches they return:

1. Low selectivity: 21 rules return 100 matches, each.
2. Medium selectivity: 14 rules return 372 matches, each.
3. High selectivity: 21 rules return 110107 matches, each.

6.2. Benchmark Tasks

The benchmark tasks address two primary scalability choke points: increasing data volume and increasing rule count. An increase in data volume corresponds to a higher number of RDF triples in the KB, while an increase in rule count reflects a growing policy size.

Benchmark rules were selected from one selectivity group at a time. This ensures that rules added at each scaling step exhibit comparable match counts, thereby making the benchmark more controlled and meaningful. For the reported experiments, we used rules from the different selectivity groups. Likewise, subsets of the KB were generated from the initial full KB while preserving proportional match counts for each rule. Table 2 outlines the different tasks associated with each choke point and their respective scaling strategy. Specifically, for the first task, the policy

Table 2

Benchmark Tasks.Benchmark Tasks.*The selectivity groups categorize rules based on the number of matches they return over the input DKB: Low corresponds to rules with approximately 100 matches, Medium to around 372 matches, and High to about 110,107 matches. The policy size refers to the number of rules in a policy, while the knowledge base size denotes the number of triples in the knowledge base.

Task	Selectivity Group*	Policy Size*	Knowledge Base Size*	Choke Point
1	High & Low	5, 9, 13, 17, 21	1318136	Increasing number of rules
	Medium	6, 8, 10, 12, 14		
2	High & Low	13	500000, 909068, 1318136,	Increasing KB volume
	Medium	10	1727204, 2136272, 2545342	

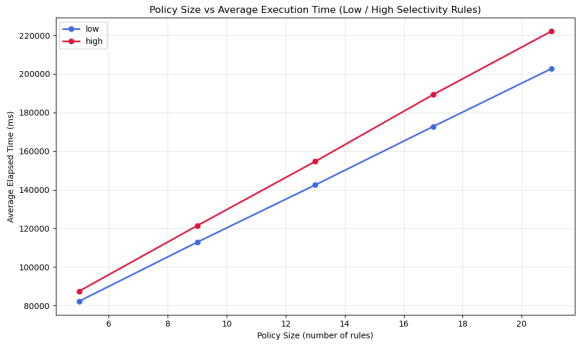
size is increased in steps of 4 rules for the high- and low-selectivity groups, whereas it is increased in steps of 2 rules for the medium-selectivity group. The reason for using different scales across selectivity groups is the number of unique rules that could be generated from our input dataset. Meanwhile, the KB size is scaled in increments of approximately 400000 triples.

6.3. Experimental Setup & Result

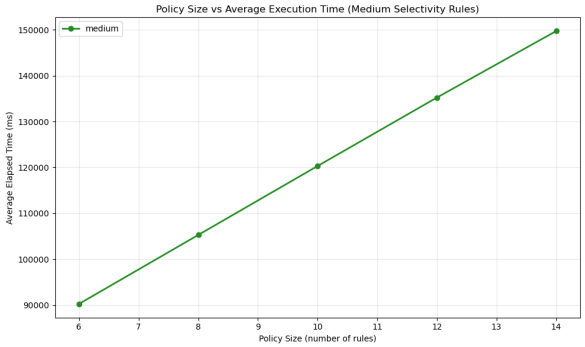
Experiments were run on a virtual machine with 125GB of RAM, equipped with an Intel® Xeon® Silver 4114 CPU@2.20GHz, featuring 10 physical cores on Fedora Linux 42 (Adams). The TDB2 was parameterized to use an IRI that labels the loaded KB. All the subsequent processing steps are performed in memory. All calculations presented were based on an average of 10 response times excluding the two slowest and fastest times in a cold cache scenario (caches are emptied at the start of each process). Figures 6 and 7 present the results of our experiments. Specifically, they show a steady linear increase in execution time (in ms) as the size of policy and KB increases. This indicates that the prototype maintains predictable performance behavior with respect to larger policies and knowledge bases. Additionally, Figures 6a and 7a show that high selectivity rules take more time than the medium selectivity rules, which demonstrates that the selectivity of the policy rules has an effect on the execution time. Figures 6b and 7b show that the medium selectivity rules take on average similar execution time as the low selectivity rules as the medium and low rules are close in terms of selectivity. We plot medium-selectivity results separately due to their differing scale, to ensure clearer visual comparison.

6.4. Discussion

The performance and scalability evaluation demonstrates a largely linear and steady increase in execution time with respect to both knowledge base size and policy size. In the proposed approach, evaluating the system at a given time instant t requires constructing a temporal snapshot containing all triples whose timestamps are valid up to t. As a result, larger values of ttt produce progressively larger snapshots, increasing evaluation costs due to the growing amount of temporal data that must be processed. This trade-off is inherent to snapshot-based temporal reasoning approaches, where preserving correctness and temporal completeness requires considering the full relevant history of the graph rather than only recent updates. Similar challenges have been discussed in temporal RDF and stream reasoning research [44], where approaches such as incremental maintenance and window-based processing have been proposed to improve scalability over evolving knowledge graphs. In the context of the proposed GUCON extension, incremental maintenance would avoid reconstructing the temporal snapshot from scratch for each evaluation. Instead, when moving from a snapshot at time t1 to a later time t2, the system would reuse the previously computed snapshot and incorporate only the newly added or modified triples whose timestamps fall within (t1,t2]. This would reduce redundant processing and mitigate the impact of the continuously growing knowledge base. Conversely, a window-based approach would restrict evaluation to a bounded temporal interval instead of considering the complete history up to t. While this would reduce the size of the temporal snapshot and improve performance, it would do so at the cost of no longer guaranteeing complete historical coverage of the graph. Investigating and integrating optimization approaches that reduce the impact of increasing time instants t on the processing of the KB is left as an important direction for future work.

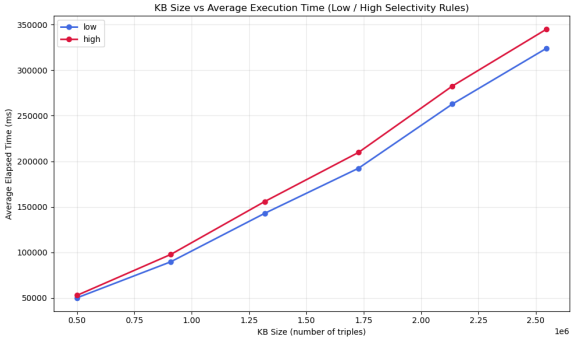


(a) Task 1: Execution Time in Terms of Policy Size with high & low Selectivity Rules

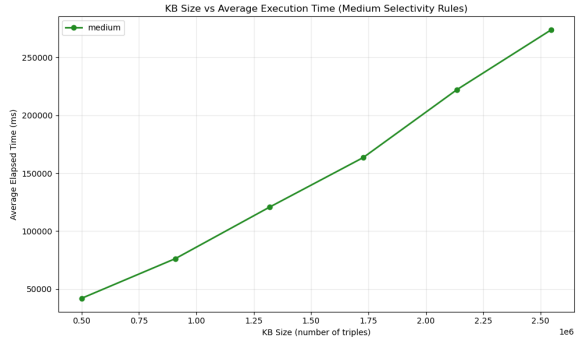


(b) Task 1: Execution Time in Terms of Policy Size with Medium Selectivity Rules

Figure 6. Experiments Results of Task 1



(a) Task 2: Execution Time in Terms of KB Size with High & Low Selectivity Rules



(b) Task 2: Execution Time in Terms of KB Size with Medium Selectivity Rules

Figure 7. Experiments Results of Task 2

7. Related Work

In the following, we review related work on representing and reasoning over obligations in usage control. Additionally, as our work evolves around representing temporal obligations using SPARQL and RDF, we touch upon existing approaches for modeling and querying temporal information in RDF.

7.1. Usage Control Obligations

There are several works that focus on usage control in general and obligation monitoring in particular. UCON [47] is an abstract model that extends access control with the concepts of obligations, decision continuity and attribute mutability. Although several formalisms have been suggested for UCON, and attempts have been made to include it in standard representation languages [13, 42], there is currently no established reference or standard policy specification and implementation for UCON. As a result, UCON has not gained widespread adoption in industry. Moreover, it does not explicitly model a state of affairs, which limits its suitability for runtime reasoning. Another language, OSL, formalized in Z [29], is utilized to express conditional prohibitions and obligations. Although one can express temporal constraints using OSL, the obligations in OSL do not support the notion of obligation states. Additionally, it is unclear how said obligations should be enforced.

In the Semantic Web community, several general policy languages and frameworks have been proposed, including Protune [5], Rei [37], Ponder [14], KAoS [64], and the DSA policy framework [58]. Protune focuses on access

control and trust management, but lacks support for expressing obligations. While, Rei, Ponder, KAoS, and the DSA framework can express obligations, their design lacks the states of obligations, which limits their ability to monitor obligation lifecycles. Furthermore, it remains unclear how obligations expressed in these languages are enforced in practice. Additionally, Ponder lacks formal semantics, which limits its applicability in policy reasoning. Moreover, most of these approaches are rooted in OWL and thus exhibit limited temporal reasoning, as OWL cannot natively represent or update time-evolving states.

In terms of policy standards, ODRL[17] is a W3C recommendation that provides a model and vocabulary for describing policies, including obligations. Although ODRL does not have yet an official formal semantics, several efforts have attempted to formalize it, either by using web ontologies [22] or by defining operational semantics through rule-based reasoning [19, 60]. Notably, Fornara et al. [20] focused on enriching ODRL by extending the model with the notions of permission and obligation states. The operational semantics of this extended model are implemented using a production rule system. Currently, a W3C community group[18] is working on defining the formal semantics of ODRL; which is currently only described informally in plain English, without any formal specification. Cimmino and Fornara [10] highlight several limitations of ODRL. Although ODRL supports expressing various types of constraints; such as temporal and spatial constraints; they lack formal semantics. In particular, it remains unclear how to reason effectively about obligation start times and deadlines. Furthermore, ODRL does not incorporate the notion of the state of affairs, which complicates the enforcement and/ or policy re-evaluation whenever the policy or context changes. SHACL is another standard that has been explored for expressing policies. Robaldo et al. [53] propose using first order logic to represent norms and leveraging SHACL implementations for compliance checking. However, it remains unclear how these implementations can be used to reason over the states of obligations. Another widely adopted standard is the XACML policy language and framework [17], originally developed for managing access control policies. Several studies have extended XACML with UCON capabilities [12, 39]. Another related standard is the Abbreviated Language for Authorization (ALFA)[19], a domain-specific language for specifying authorization policies. However, both XACML (including its extensions) and ALFA lack formal foundations. Despite strong industrial adoption, these languages do not provide formal semantics and do not natively support obligation state modeling or temporal lifecycle reasoning.

Our extended GUCON model builds upon the obligation state formalization introduced by Fornara et al. [20]. While we retain the core notion of temporally evolving obligations, we simplify the original model by focusing on temporal metaproperties and defining the formal semantics of obligation states independently. The state of affairs, represented as a KB, constitutes a central component of the GUCON framework. GUCON further exploits the formal semantics of SPARQL graph patterns, enabling efficient and dynamic re-evaluation of policies with respect to the dynamic state of the world. From an implementation perspective, GUCON is realized using RDF 1.2 and SPARQL 1.2. RDF 1.2 is currently a W3C Candidate Recommendation, reflecting its ongoing standardization status. Table 3 summarizes the main characteristics of existing approaches and highlights the differences with our extended GUCON model. For a more in-depth analysis of the state of the art in usage control solutions, the survey in [1] offers a comprehensive overview of the existing gaps in this field.

7.2. Modeling Time in RDF

The management of highly dynamic RDF data in a timely fashion has led to the development of RDF stream processing, a computation paradigm in which data is processed in motion, i.e., on the fly and as soon as it becomes available [6]. Another important use case for handling temporal RDF is provenance management (e.g., versioning) and tracking changes in RDF datasets [43]. Both directions have motivated a wide range of temporal extensions to RDF data models and query languages [6, 67]. Representing temporal information in RDF has been widely studied in the context of extending the standard entity-based model with explicit notions of time. Existing approaches can be broadly classified according to the level at which temporality is introduced [9]: graph-level, triple-level, and

[17] ODRL, https://www.w3.org/TR/odrl-model/
[18] ODRL Formal Semantics, https://w3c.github.io/odrl/formal-semantics/
[19] ALFA, https://alfa.guide/

Table 3
Comparison of policy languages and frameworks with respect to obligation handling and formal reasoning capabilities.

Language / Framework	Formal Semantics	Temporal Constraints	Obligation Support	Obligation States	State of Affairs	Standard / Recommendation	Industrial Adoption
UCON [47]	Partial	✓	✓	✓	×	×	Limited
OSL [29]	✓	✓	✓	×	×	×	Limited
Protune [5]	✓	Limited	×	×	×	×	Limited
Rei [37]	✓	Limited	✓	×	×	×	Limited
Ponder [14]	×	Limited	✓	×	×	×	Limited
KAoS [64]	✓	Limited	✓	×	×	×	Limited
DSA Framework [58]	✓	✓	✓	×	×	×	Limited
ODRL [19, 60]	Partial	✓	✓	Partial	×	W3C	Moderate
First Order Logic / SHACL [53]	✓	×	✓	×	✓	×	Limited
XACML extensions [12, 17, 39]	×	Limited	✓	×	×	OASIS	High
ALFA [17]	×	Limited	×	×	×	OASIS	High
Extended GUCON	✓	✓	✓	✓	✓	×	Prototype

component-level extensions. Graph-level approaches, such as Named Graphs and n-quads, group RDF triples into named contexts, where temporal validity can be associated with entire sets of statements. Similar to our work, triple-level approaches, including reification, tRDF [23], aRDF [63], and RDF* [26] (currently RDF-star), represent temporal information at the statement level by either explicitly reifying triples or implicitly annotating them with temporal labels. In contrast, component-level approaches embed temporal information directly into elements of a triple; for example, 4D fluents [66] model entities as time-dependent slices, while singleton properties [45] and n-ary relations [46] associate predicates or relations with specific temporal contexts. These approaches rely on explicit or implicit temporal semantics to represent notions such as instants, intervals, and durations, and to support reasoning over them. Some frameworks, such as tRDF and 4D fluents, extend RDF semantics directly and introduce dedicated temporal constructs, while more general-purpose approaches (e.g., Named Graphs and RDF*) typically depend on external vocabularies, such as the OWL-Time ontology[20], to provide formal temporal grounding.

8. Conclusion

In this paper, we presented an extension of the GUCON model that integrates temporal properties into obligations, enabling their continuous monitoring and reasoning over time. By incorporating start times and deadlines, our approach supports the formal assessment of obligation states and allows the evaluation of compliance with respect to a temporal knowledge graph (state of affairs). We further demonstrated how the extended model can be instantiated using RDF 1.2 and SPARQL 1.2 and introduced an Obligation State Manager that tracks obligation states and checks compliance against the state of affairs. The extended model highlights several properties: (i) adaptability, as the model can be easily extended with new constraints; (ii) reusability, since the model's generality allows it to be applied across multiple domains; (iii) formalness, due to its grounding in SPARQL formal semantics; (iv) correctness, and in particular its abilitiy to cover test cases for temporal obligations; (v) mobility, by showing different ways of instantiating the model using Semantic Web languages; and (vi) usability, as GUCON deliberately employs syntax that is aligned with Semantic Web standards, which benefits users who are already familiar with RDF and SPARQL-based technologies. To assess our approach, we evaluated the performance and scalability of the Obligation State Manager prototype, with results showing that the system scales well with increasing numbers of obligation policies and larger knowledge graphs.

Future work will focus on extending the model with additional constraints, such as spatial properties, and investigating the interplay between temporal and spatial aspects for capturing realistic usage control scenarios. To

[20]Time Ontology in OWL, https://www.w3.org/TR/owl-time/

strengthen interoperability, we aim to develop systematic mappings between different instantiations of GUCON, such as OWL and SHACL. Additionally, we also aim to address usability by designing intuitive interfaces that enable users to craft and manage GUCON policies more effectively. Finally, we intend to investigate and integrate optimization approaches to mitigate the impact of increasing time instants t on the processing performance of the knowledge base. We also plan to compare our approach with ODRL, which is one of the few W3C recommendations for policy models and has attracted a growing body of extensions aimed at enabling more expressive policy representation and enforcement.

Acknowledgements

This work is funded by the European Union Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 860801 and the FWF Austrian Science Fund [10.55776/COE12]. Sabrina Kirrane is also 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] Ines Akaichi and Sabrina Kirrane. A comprehensive review of usage control frameworks. *Computer Science Review*, 56, 2025.

[2] Ines Akaichi, Giorgos Flouris, Irini Fundulaki, and Sabrina Kirrane. Gucon: A generic graph pattern based policy framework for usage control enforcement. In *Rules and Reasoning: 7th International Joint Conference, RuleML+RR 2023, Oslo, Norway, September 18–20, 2023, Proceedings*, 2023.

[3] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26, 1983.

[4] Zubaria Asma, Daniel Hernández, Luis Galárraga, Giorgos Flouris, Irini Fundulaki, and Katja Hose. Npcs: Native provenance computation for sparql. *WWW 2024 - Proceedings of the ACM Web Conference*, 2024.

[5] Piero Bonatti, J.L. De Coi, Daniel Olmedilla, and Luigi Sauro. A rule-based trust negotiation system. *IEEE Transactions on Knowledge and Data Engineering*, 22(11), 2010.

[6] Pieter Bonte, Christophe Callé, Olivier Curé, Haridimos Kondylakis, and Riccardo Tommasini. Languages and systems for rdf stream processing, a survey: Languages and systems for rdf stream processing, a survey. *The VLDB Journal*, 34(4), June 2025. ISSN 1066-8888. . URL https://doi.org/10.1007/s00778-025-00927-7.

[7] Meisam Booshehri, Lukas Emele, Simon Flügel, Hannah Förster, Johannes Frey, Ulrich Frey, Martin Glauer, Janna Hastings, Christian Hofmann, Carsten Hoyer-Klick, Ludwig Hülk, Anna Kleinau, Kevin Knosala, Leander Kotzur, Patrick Kuckertz, Till Mossakowski, Christoph Muschner, Fabian Neuhaus, Michaja Pehl, Martin Robinius, Vera Sehn, and Mirjam Stappel. Introducing the open energy ontology: Enhancing data interpretation and interfacing in energy systems analysis. *Energy and AI*, 5, 2021.

[8] Jm Jeffrey M Bradshaw, Stewart Dutfield, Pete Benoit, and John D Jd Woolley. Kaos: toward an industrial-strength open agent architecture. *Software Agents*, 1997.

[9] Oleksandra Bruns. The persistence of temporality: Representation of time in cultural heritage knowledge graphs. In *2023 Doctoral Consortium at International Semantic Web Conference, ISWC-DC co-located with 22nd International Semantic Web Conference {(ISWC}, Athens, 6th-10th November 2023*, volume 3678 of *CEUR Workshop Proceedings*. CEUR-WS, 2023.

[10] Andrea Cimmino and Nicoletta Fornara. Improving odrl 2.2: current limitations and theoretical solutions. In *Joint Proceedings of the ESWC 2025 Workshops and Tutorials co-located with 22nd Extended Semantic Web Conference (ESWC 2025)*, 2025.

[11] Andrea Cimmino, Juan Cano-Benito, and Raúl García-Castro. Open digital rights enforcement framework (odre): From descriptive to enforceable policies. *Comput. Secur.*, 150, 2025.

[12] Maurizio Colombo, Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. A proposal on enhancing xacml with continuous usage control features. In *Grids, P2P and Services Computing*, 2010.

[13] Maurizio Colombo, Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. A proposal on enhancing xacml with continuous usage control features. In *Grids, P2P and Services Computing*, 2010.

[14] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1995, 2001.

[15] Marina De Vos, Sabrina Kirrane, Julian Padget, and Ken Satoh. Odrl policy modelling and compliance checking. In *Rules and Reasoning: Third International Joint Conference, RuleML+RR 2019, Bolzano, Italy, September 16–19, 2019, Proceedings*, 2019.

[16] Ana Dimishkovska. Deontic logic and legal rules. *Encyclopedia of the Philosophy of Law and Social Philosophy*, 2017.

[17] Erik Rissanen. extensible access control markup language (xacml) version 3.0., 2013. URL http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf. Online; accessed 14 February 2023.

[18] Javier D. Fernández, P.A. Bonatti, U. Milosevic, and Jonathan Langens. Scalability and Robustness testing report V2. Technical report, H2020 Project, 2019. URL https://specialprivacy.ercim.eu/images/documents/SPECIAL_D35_M27_V10.pdf.

[19] Nicoletta Fornara and Marco Colombetti. Operational semantics of an extension of odrl able to express obligations. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10767 LNAI, 2018.

[20] Nicoletta Fornara, Alessia Chiappa, and Marco Colombetti. Using semantic web technologies and production rules for reasoning on obligations and permissions. In Marin Lujak, editor, *Agreement Technologies*, 2019.

[21] Antony Galton. Reified temporal theories and how to unreify them. *12th International Joint Conference on Artificial Intelligence*, 88, 1991.

[22] Roberto García, Rosa Gil, Isabel Gallego, and Jaime Delgado. Formalising odrl semantics using web ontologies. In *Proc. 2nd Intl. ODRL Workshop*, 2005.

[23] Claudio Gutierrez, Carlos Hurtado, and Alejandro Vaisman. Temporal rdf. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *The Semantic Web: Research and Applications*, 2005.

[24] Claudio Gutierrez, Carlos A. Hurtado, and Alejandro Vaisman. Introducing time into rdf. *IEEE Transactions on Knowledge and Data Engineering*, 19(2), 2007.

[25] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. Sparql 1.1 query language. W3c recommendation, World Wide Web Consortium (W3C), March 2013. URL http://www.w3.org/TR/2013/REC-sparql11-query-20130321/. Editors: Steve Harris and Andy Seaborne; previous editor: Eric Prud'hommeaux.

[26] Olaf Hartig. Foundations of rdf* and sparql*: (an alternative approach to statement-level metadata in rdf). In Juan Reutter and Divesh Srivastava, editors, *Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web (AMW 2017)*, volume 1912 of *CEUR Workshop Proceedings*, Montevideo, Uruguay, June 2017. June 7–9, 2017.

[27] Olaf Hartig, Pierre-Antoine Champin, Andy Seaborne, and Gregg Kellogg. Rdf 1.2 concepts and abstract data model. W3c candidate recommendation snapshot, World Wide Web Consortium (W3C), April 2026. URL https://www.w3.org/TR/2026/CR-rdf12-concepts-20260407/. Editors: Olaf Hartig, Pierre-Antoine Champin, Andy Seaborne, Gregg Kellogg (in memoriam).

[28] Olaf Hartig, Andy Seaborne, Ruben Taelman, Gregory Williams, and Thomas Pellissier Tanon. Sparql 1.2 query language. W3c working draft, World Wide Web Consortium (W3C), April 2026. URL https://www.w3.org/TR/2026/WD-sparql12-query-20260423/. Editors: Olaf Hartig, Andy Seaborne, Ruben Taelman, Gregory Williams, Thomas Pellissier Tanon.

[29] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. A policy language for distributed usage control. In Joachim Biskup and Javier López, editors, *Computer Security – ESORICS 2007*, 2007.

[30] Jerry R. Hobbs and Feng Pan. An ontology of time for the semantic web. *ACM Transactions on Asian Language Information Processing*, 3, 2004.

[31] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194, 2013.

[32] Siquan Hu, Haiou Wang, Chundong She, and Junfeng Wang. Agont: Ontology for agriculture internet of things. In Daoliang Li, Yande Liu, and Yingyi Chen, editors, *Computer and Computing Technologies in Agriculture IV*, 2011.

[33] Iannella, Renato and Villata, Serena. Odrl information model 2.2, 2018. URL https://www.w3.org/TR/odrl-model/. Online; accessed 28 April 2025.

[34] Ana Iglesias-Molina, Dylan Van Assche, Julián Arenas-Guerrero, Ben De Meester, Christophe Debruyne, Samaneh Jozashoori, Pano Maria, Franck Michel, David Chaves-Fraga, and Anastasia Dimou. The rml ontology: A community-driven modular redesign after a decade of experience in mapping heterogeneous data to rdf. In *The Semantic Web – ISWC 2023: 22nd International Semantic Web Conference, Athens, Greece, November 6–10, 2023, Proceedings, Part II*, page 152–175, Berlin, Heidelberg, 2023. Springer-Verlag. ISBN 978-3-031-47242-8. . URL https://doi.org/10.1007/978-3-031-47243-5_9.

[35] Tomayess Issa and Pedro T. Isaías. Usability and human–computer interaction (hci). *Sustainable Design*, 2022. URL https://api.semanticscholar.org/CorpusID:64142469.

[36] L. Kagal and T. Berners-Lee. Trein: Where policies meet rules in the semantic web. Technical report, CSAIL, Massachusetts Institute of Technology, 2005. URL http://groups.csail.mit.edu/dig/2005/05/rein/rein-paper.pdf.

[37] Lalana Kagal. Rei 1 : A policy language for the me-centric project rei £ : A policy language for the me-centric project. Technical report, HP Laboratories, 2002. URL https://ebiquity.umbc.edu/_file_directory_/papers/57.pdf.

[38] Uri Kartoun. Emrbots: Synthetic electronic medical records dataset. https://github.com/kartoun/emrbots, 2019. Accessed: 2026-05-13.

[39] Donia Kateb, Yehia Elrakaiby, Tejeddine Mouelhi, Iram Rubab, and Yves Le Traon. Towards a full support of obligations in xacml. In *Risks and Security of Internet and Systems*, 08 2014.

[40] Megan Katsumi and Mark Fox. Ontologies for transportation research: A survey. *Transportation Research Part C: Emerging Technologies*, 89, 2018.

[41] Timotej Knez and Slavko Žitnik. Event-centric temporal knowledge graph construction: A survey. *Mathematics*, 11, 12 2023.

[42] Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. Usage control in computer security: A survey. *Computer Science Review*, 4(2), 2010.

[43] Arcangelo Massari, Silvio Peroni, Francesca Tomasi, and Ivan Heibi. Representing provenance and track changes of cultural heritage metadata in rdf: a survey of existing approaches. *Digital Scholarship in the Humanities*, 41:i196–i212, 04 2026. ISSN 2055-7671. . URL https://doi.org/10.1093/llc/fqaf076.

[44] Wafaa Mebrek. *Stream processing and incremental reasoning over RDF streams*. PhD thesis, Institut Polytechnique de Paris, 2024.

[45] Vinh Nguyen, Olivier Bodenreider, and Amit Sheth. Don't like rdf reification? making statements about statements using singleton property. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, page 759–770, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450327442. . URL https://doi.org/10.1145/2566486.2567973.

[46] Natasha Noy, Alan Rector, Patrick Hayes, and Christopher Welty. Defining n-ary relations on the semantic web. W3C Working Group Note 12, World Wide Web Consortium (W3C), April 2006.

[47] Jaehong Park and Ravi Sandhu. The uconabc usage control model. *ACM Transactions on Information and System Security*, 7, 2004.
[48] Peter Patel-Schneider, Dörthe Arndt, and Enrico Franconi. Rdf 1.2 semantics. W3c candidate recommendation snapshot, World Wide Web Consortium (W3C), April 2026. URL https://www.w3.org/TR/2026/CR-rdf12-semantics-20260407/. Editors: Peter Patel-Schneider, Dörthe Arndt, Enrico Franconi.
[49] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *The Semantic Web - ISWC 2006*, 2006.
[50] Rajesh Piryani, Nathalie Aussenac-Gilles, and Nathalie Hernandez. Comprehensive survey on ontologies about event. *CEUR Workshop Proceedings*, 3443, 2023.
[51] Alexander Pretschner, Manuel Hilty, Florian Schütz, Christian Schaefer, and Thomas Walter. Usage control enforcement: Present and future. *IEEE Security & Privacy*, 6(4), 2008.
[52] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. https://www.w3.org/TR/rdf-sparql-query/, 2008. W3C Recommendation 15 January 2008.
[53] L. Robaldo, S. Batsakis, R. Calegari, and et al. Compliance checking on first-order knowledge with conflicting and compensatory norms: a comparison among currently available technologies. *Artificial Intelligence and Law*, 2023.
[54] Livio Robaldo. Towards compliance checking in reified i/o logic via shacl. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law*, 2021.
[55] Livio Robaldo. Towards compliance checking in reified i/o logic via shacl. *Proceedings of the 18th International Conference on Artificial Intelligence and Law, ICAIL 2021*, 2021.
[56] Livio Robaldo and Xin Sun. Reified input/output logic: Combining input/output logic and reification to represent norms coming from existing legislation. *Journal of Logic and Computation*, 27, 2017.
[57] Michael Röder, Denis Kuchelev, and Axel-Cyrille Ngonga Ngomo. Hobbit: A platform for benchmarking big linked data. *Data Science*, 3 (1), 2020.
[58] Henrique Santos, Alice Mulvehill, John S. Erickson, Jamie P. McCusker, Minor Gordon, Owen Xie, Samuel Stouffer, Gerard Capraro, Alex Pidwerbetsky, John Burgess, Allan Berlinsky, Kurt Turck, Jonathan Ashdown, and Deborah L. McGuinness. A semantic framework for enabling radio spectrum policy management and evaluation. In *The Semantic Web – ISWC 2020: 19th International Semantic Web Conference, Athens, Greece, November 2–6, 2020, Proceedings, Part II*, 2020.
[59] Henrique Santos, Jamie P Mccusker, John S Erickson, Alice M Mulvehill, Oshani Seneviratne, and Deborah L Mcguinness. Towards computable and explainable policies using semantic web standards. In *15th Workshop on Ontology Design and Patterns co-located with ISWC 2024*, 2024.
[60] Simon Steyskal and Axel Polleres. Towards formal semantics for odrl policies. In *International Web Rule Symposium*, 2015.
[61] Christer Thörn. *On the Quality of Feature Models*. Ph.d. thesis, Linköping University, 2010. URL https://liu.diva-portal.org/smash/get/diva2:313940/FULLTEXT01.pdf.
[62] Alessandra Toninelli, Jeffrey Bradshaw, Lalana Kagal, Rebecca Montanari, et al. Rule-based and ontology-based policies : Toward a hybrid approach to control agents in pervasive environments. *Informatica*, 2005.
[63] Octavian Udrea, Diego Reforgiato Recupero, and V. S. Subrahmanian. Annotated rdf. *ACM Trans. Comput. Logic*, 11(2), January 2010. ISSN 1529-3785. . URL https://doi.org/10.1145/1656242.1656245.
[64] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. *Proceedings - POLICY 2003: IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, 2003.
[65] Lluís Vila and Han Reichgelt. The token reification approach to temporal reasoning. *Artificial Intelligence*, 83, 1996.
[66] Chris Welty and Richard Fikes. A reusable ontology for fluents in owl. In *Proceedings of the 2006 Conference on Formal Ontology in Information Systems: Proceedings of the Fourth International Conference (FOIS 2006)*, page 226–236, NLD, 2006. IOS Press. ISBN 1586036858.
[67] Marcin Wylot, Philippe Cudré-Mauroux, Manfred Hauswirth, and Paul Groth. Storing, tracking, and querying provenance in linked data. *IEEE Transactions on Knowledge and Data Engineering*, 29(8):1751–1764, 2017. .