Modeling and Managing Temporal Obligations in GUCON Using SPARQL-star and RDF-star

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 times 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 SPAQRL 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-star and SPARQL-star and propose an Obligation State Manager that monitors obligation states and assess their compliance with respect to usage traces. Finally, we evaluate both the extended model and its prototype implementation.

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 [34].

A central mechanism for operationalization these conditions and ensuring continuous policy enforcement is the concept of obligations [23]. Obligations specify the required actions that must be performed before, during, or after data usage thereby ensuring accountability, compliance, and trust throughout the entire data usage process.

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

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 [17], as time passes. The enforcement of obligations follows two main strategies [37]: 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. [23], 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) [34] introduced obligations, decision continuity, and attribute mutability, yet despite various formalizations and integration attempts [11, 33], no standard specification or widely adopted implementation exists. Policy languages such as the Obligation Specification Language (OSL) [23], Rei [29], Ponder [12], Proteus [48], KAoS [49], and the Dynamic Spectrum Access Policy Framework (DSA policy framework) [44] 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) [27], 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-star and SPARQL-star. 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-star and SPARQL-star 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-star, SPARQL-star, 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

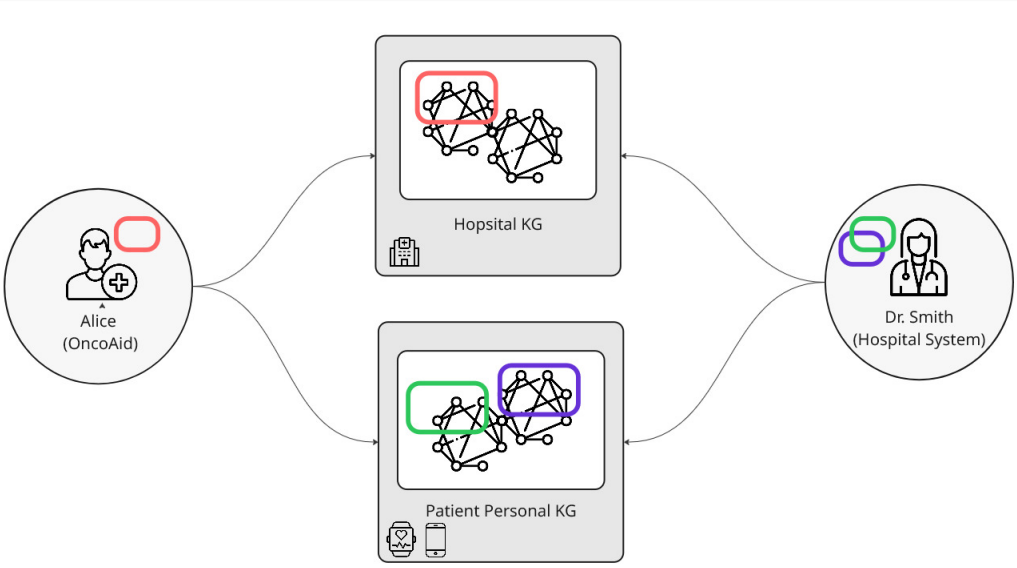


Figure 1. A Depiction of the OncoAid Use Case

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.

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[1]. These tasks are defined based on the concepts of state of affairs and usage control policies, 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.

[1] In this paper, we focus primarily on compliance and requirements checking, as our emphasis is solely on obligations.

3.1. RDF & SPARQL

In our work, we rely on the syntax and semantics of graph patterns expressions presented in [35]. Throughout the paper, we assume two pairwise disjoint and 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 [38] 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. Throughout this paper, the state of affairs will be referred to as the knowledge base (KB).

Definition 3 (Knowledge Base)**.** A KB, denoted as K, is an RDF graph describing the set of actual knowledge.

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 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 (e.g., `?doctor ?sign ?diagnosisReport`), to allow generally applicable restrictions to be expressed. We refer to such a tuple as an *action pattern*.

Definition 4 (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.

An obligation pattern can be defined as follows.

Definition 5 (Obligation Pattern)**.** Let **O** denotes 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 a defined as (n, c, r), $\mathbf{O}(n, c, r)$: indicates that an entity n is obliged to perform an action c over a resource r. 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, and 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 6 (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 $var(a) \subseteq var(cond)$.

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

Definition 7 (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.

Definition 8 (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 9 (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[2]. 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.

[2]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.

Definition 10 (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 11 (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 [21], we extend the GUCON model of obligation rules to encompass temporal properties. Consequently, the KB is also extended to represent these temporal attributes.

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 store two kinds of triples: (1) facts that do not change over time. For example, *Dr. Smith is a doctor*. Although facts may evolve over time, 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. [21], an EKB is defined as follows.

Definition 12 (Event Knowledge Base)**.** An event is a tuple (n, c, r, t_{exec}). An EKB is a finite set of events.

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

Definition 13 (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: $K_t = K^s \cup K^e_t$.

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.

Definition 14 (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.

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

Definition 15 (Extended Obligation Pattern)**.** An *extended obligation pattern* is of the form $\mathbf{O}ea$, where $\mathbf{O} \in \mathcal{D}$ and $ea \in \mathcal{EAP}$.

Similarly, an obligation rule is extended as follows.

Definition 16 (Extended Obligation Rule)**.** An extended obligation rule is of the form: $cond \rightsquigarrow \mathbf{O}ea$, where *cond* is a graph pattern, and $\mathbf{O}ea$ is an extended deontic pattern. We denote by $\mathcal{EO}$ the set of all extended obligation rules.

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 [18].

Generally, given a snapshot K_t of a KB at time t, an extended obligation rule *OR* of the general form $cond \rightsquigarrow \mathbf{O}(np, cp, rp, tp_{start}, tp_{deadline})$, *OR* can either be *active*, *fulfilled*, *violated*, *expired*, or *not satisfied*. Figure 2 illustrates the states of obligations with regards 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. We extend the definition of satisfied condition by including the time aspect as follows.

Definition 17 (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 $\mu \in [[cond]]_{K_t}$.

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 18 (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.

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.

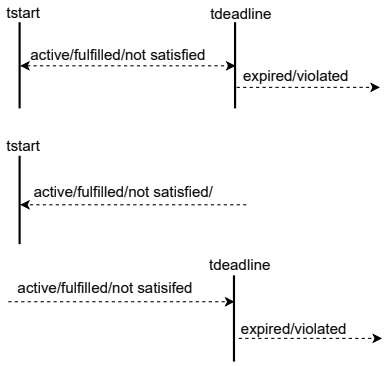


Figure 2. States Of Obligations.

Definition 19 (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*.

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 20 (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*.

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 21 (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*.

A not satisifed 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 22 (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 satisifed obligations at time *t*.

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

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[3].

[3]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.

5.1. Implementing Temporal GUCON using RDF-star and SPARQL-star

In what follows, we present an overview of SPARQL-star and RDF-star. Then, we demonstrate how to represent our GUCON temporal obligations using SPARQL-star and RDF-star syntax.

5.1.1. RDF-star and SPARQL-star

The RDF-star data model extends RDF by allowing arbitrarily deep nesting of triples as subject or object arguments. Any RDF triple $tr \in TR$ is an RDF-star triple. Given RDF-star triples tr and tr', and RDF terms $s \in I$, $p \in I$, and $o \in (I \cup L)$, then the triples (tr, p, o), (s, p, tr) and (tr, p, tr') are also RDF-star triples.

Definition 24 (SPARQL-star Triple Pattern)**.** A SPARQL-star triple pattern is a 3-tuple that is defined recursively as follows:

- Every SPARQL triple pattern is a SPARQL-star triple pattern;
- If tr and tr' are SPARQL-star triple patterns, $x \in (IL \cup V)$, $p \in (I \cup V)$, then (tr, p, x), (x, p, tr), and (tr, p, tr') are SPARQL-star triple patterns.

Definition 25 (SPARQL-star Graph Pattern)**.** A SPARQL-star graph pattern is defined recursively as follows:

- A SPARQL-star triple pattern is a SPARQL-star graph pattern.
- If $G1$ and $G2$ are SPARQL-star graph pattern, then (G1 AND G2), (G1 OPT G2), (G1 UNION G1), (G1 MINUS G2) are SPARQL-star graph pattern.
- If G is a SPARQL-star graph pattern and R is a filter expression, then (G FILTER R) is a SPARQL-star 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 [38] for a complete list).

The semantics of SPARQL-star graph pattern expressions are similar to the semantics of standard SPARQL graph patterns [35]. The notion of a SPARQL-star solution mapping extends the notion of a standard SPARQL solution mapping; that is, every SPARQL solution mapping is a SPARQL-star solution mapping. However, in contrast to SPARQL solution mappings, SPARQL-star solution mappings may map variables also to RDF-star triples.

5.1.2. Syntax for Temporal GUCON

In order to represent the temporal aspects of GUCON rules and the temporal KB, we adopt SPARQL-star and RDF-star that enable the use of embedded triples to annotate statements with temporal 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 SPARQL-star triple pattern[4]:

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

where $?n\ ?a\ ?r$ represents an embedded triple pattern and tp_{start}, tp_{deadline} map to literals of type *xsd:dateTime*. Rule conditions may also contain SPARQL-star triple pattern, making both the action pattern and its condition SPARQL-star graph patterns.

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 thus an RDF-star graph, capable of expressing both static and temporal information.

[4]RDF-star and SPARQL-star, https://w3c.github.io/rdf-star/cg-spec/2021-07-01.html

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`)`**:**
  $K_t \leftarrow$ `GET_SNAPSHOT(`K, t`)`
  `states` $\leftarrow$ new `ObligationStates()`
  foreach *rule* $r \in P$ **do**
    $\omega \leftarrow$ `GET_MAPPINGS(`*r.condition*, K_t`)`
    if $\omega \neq \emptyset$ **then**
      foreach *binding* $\mu \in \omega$ **do**
        mappedRule $\leftarrow \mu(r)$
        if *mappedRule.startTime* $\leqslant t \leqslant$ *mappedRule.deadline* **then**
          `states.active0` $\leftarrow$ `states.active0` $\cup$ {mappedRule}
          if *mappedRule.executionTime* $=$ `null` **then**
            `states.notSatisifed0` $\leftarrow$ `states.notSatisifed0` $\cup$ {mappedRule}
          end
        end
        else if $t >$ *mappedRule.deadline* **then**
          `states.expired0` $\leftarrow$ `states.expired0` $\cup$ {mappedRule}
          if *mappedRule.executionTime* $=$ `null` ***or*** *mappedRule.executionTime* $>$ *mappedRule.deadline* **then**
            `states.violated0` $\leftarrow$ `states.violated0` $\cup$ {mappedRule}
          end
        end
        else if *mappedRule.executionTime* $\neq$ `null` **then**
          if *mappedRule.startTime* $\leqslant$ *mappedRule.executionTime* $\leqslant$ *mappedRule.deadline* **then**
            `states.fulfilled0` $\leftarrow$ `states.fulfilled0` $\cup$ {mappedRule}
          end
        end
      end
    end
  end
  return `states`

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

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

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

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

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 RDF-star 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-star. While *?startTime* and *?deadline* are not always explicitly provided, they can be computed dynamically using the `BIND`

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.expiredO ∩ states.violatedO ≠ ∅ then
4         return NON_COMPLIANT
5     end
6     return COMPLIANT
```

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 KB and a policy, both provided as Turtle files, and a time instant t, expressed using the W3C XML Schema Definition Language (XSD)[5].

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 in Algorithm 1 (line 2). If the sets of violated and expired rules are disjoint (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.

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

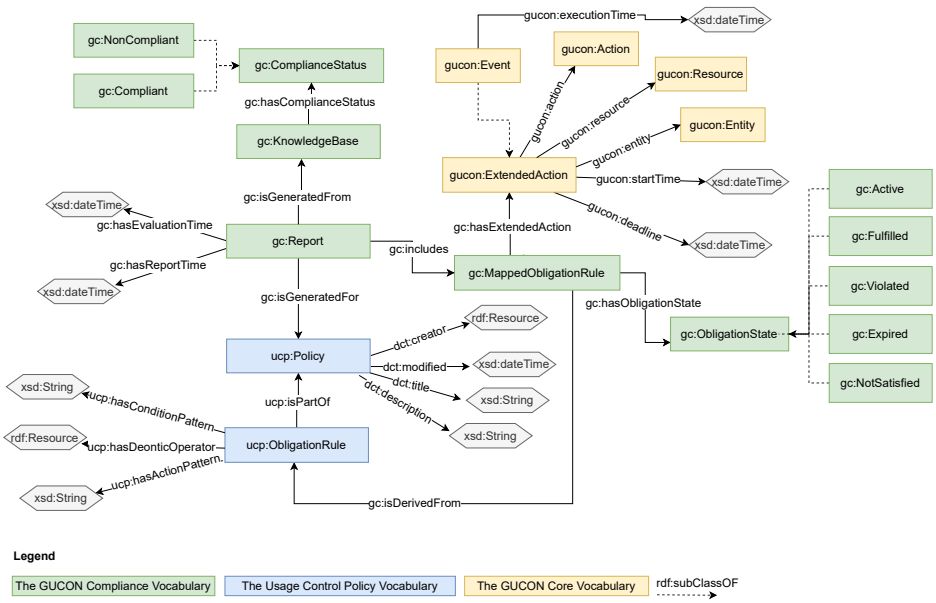


Figure 3. An ontology for a Compliance Report

An RDF Vocabulary for GUCON Policies. To ensure interoperability and facilitate automated processing, GUCON policies are represented in RDF. 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)[6].

An RDF 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 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. An *gucon:ExtendedAction*, connects to an *gucon:Entity*, an *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[7].

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 Hospital Inpatient Care (HIC) ontology[8]. Specifically, we demonstrate how the GUCON State Manager processes input data and generates compliance outcomes

[6]Data Catalog Vocabulary (DCAT), https://www.w3.org/TR/vocab-dcat-3/
[7]https://github.com/Ines-Akaichi/Temporal-GUCON/tree/main/ontologies
[8]The HIC ontology is accessible on our GitHub repository, https://github.com/Ines-Akaichi/Temporal-GUCON/tree/main/ontologies/domain

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 .
?diagnosisReport a hc:DiagnosisReport .
?admission a hc:Admission .
?admission hc:hasPatient ?patient .
?admission hc:hasActualAdmissionEndDate
    ?actualAdmissionEndDate .
?diagnosisReport hc:hasAdmission  ?admission .
?patient hc:hasResponsibleDoctor ?doctor .
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

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

6. Evaluation

This section presents our evaluation strategy, which comprises two complementary assessment components: the GUCON model and the GUCON State Manager. The GUCON model is evaluated in accordance with Thörn's quality assessment criteria [47], providing a structured analysis of its conceptual and design quality. In parallel, the GUCON State Manager is assessed using a benchmark design inspired by the methodology developed within the HOBBIT 2020 project[43]. For this second evaluation, we also describe the experimental setup used to conduct the benchmark and present the corresponding results.

6.1. Thörn's Criteria for Model Quality

Inspired by previous work that applied Thörn's quality assessment criteria [47] to evaluate policy models [44], we use the criteria to assess our extended GUCON model. Thörn's Criteria for model quality provide a systematic approach for evaluating the quality of modeled artifacts and encompass the following factors: *changeability*, *reusability*, *formalness*, *correctness*, *mobility*, and *usability*.

Changeability. This criteria 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 [34]. Other constraints, such as spatial, purpose-based, and event-driven restrictions are also important for capturing realistic usage control scenarios [23]. 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-star 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 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[9]. Other ontologies could similarly be employed to capture domain knowledge in mobility [31], energy [6], or agriculture [26].

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

[9] https://github.com/Ines-Akaichi/Temporal-GUCON/tree/main/use-case

Table 1
Test Cases Representing the Hospital Inpatient Care Scenarios

Sceanrio	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

the well-established formal semantics underpinning SPARQL [35]. 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. In each case, the expected states are determined based on 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[10].

Mobility. This criteria 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-star syntax. Beyond SPARQL-star, our model can also be instantiated with other widely adopted W3C languages, such as the Shapes Constraint Language (SHACL)[11] and the Web Ontology Language (OWL)[12]. SHACL was originally designed for validating RDF graphs, but it has also attracted attention for expressing policies [39, 40], particularly through its advanced features[13]. 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 [29, 44]. In OWL, an obligation can capture the entity, resource, and action using `owl:equivalentClass` together with `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

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

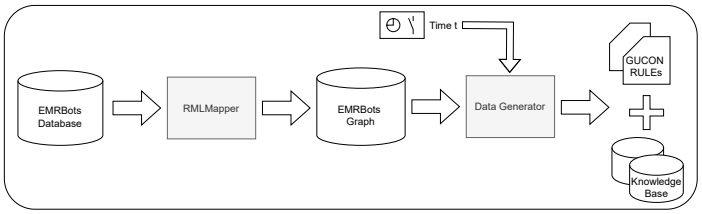


Figure 4. Data Generation Pipeline

GitHub repository[14], 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 criteria concerns the aspects of the model that affect a user's ability to define and work with it effectively. The syntax of GUCON rules is deliberately concise: both the condition and the extended action of a rule follow the SPARQL-star syntax. We have also demonstrated how this syntax can be seamlessly mapped to the Turtle language, a widely adopted and human-readable serialization. Both syntax are advantageous for Semantic Web engineers who are already familiar with existing tools. Looking ahead, we envision the development of dedicated policy editors that allow users to define policies in a more intuitive and user-friendly manner, with automatic translation into either Turtle or the GUCON syntax.

6.2. Performance & Scalability

To evaluate the performance and scalability of our prototype, 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 is aimed at identifying 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.

6.2.1. Data Generation

For our experimental evaluation, we use a dataset that closely reflects our target use case. The EMRBots dataset[15] 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 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)[16] and the RMLMapper tool[17]. 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

[14]https://github.com/Ines-Akaichi/Temporal-GUCON/tree/main/instantiation
[15]EMRBots, https://github.com/kartoun/emrbots
[16]RML, https://rml.io/specs/rml/
[17]RMLMapper, https://github.com/RMLio/rmlmapper-java

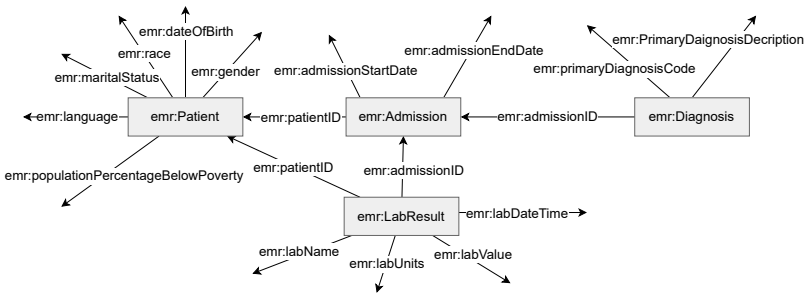


Figure 5. The EMRBots Schema

The template consists of two triple patterns that incorporate two distinct predicates selected from the EMRBots graph. Up to 56 unique rules can be 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 2400000 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.

6.2.2. 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. Table 2 outlines the different tasks associated with each choke point and their respective scaling strategy. Specifically, the policy size is incremented in steps of 4 rules, and the KB size is scaled in increments of 108,000 triples.

Selectivity. We first analyzed the selectivity of the generated rules against the KB and grouped them according to the number of matches they return:

1. Low selectivity: 21 rules return 300-400 matches.
2. Medium selectivity: 14 rules return 1116–1488 matches.
3. High selectivity: 21 rules return 330K–551K matches.

From this distribution, benchmark rules were selected from one 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 high-selectivity group. Likewise, subsets of the KB were generated from the initial dataset while preserving proportional match counts for each query.

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 empty at the start of each process). Figure 6 presents the results of our experiments. Specifically, Figure 6a and 6b shows 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.

Table 2
Benchmark Tasks

Task	Policy	Knowledge Base	Choke Point
1	5	406000	Increasing number of rules
	9		
	13		
	17		
	21		
2	13	100000	Increasing data volume
		208000	
		406000	
		604000	
		802000	
		1000000	

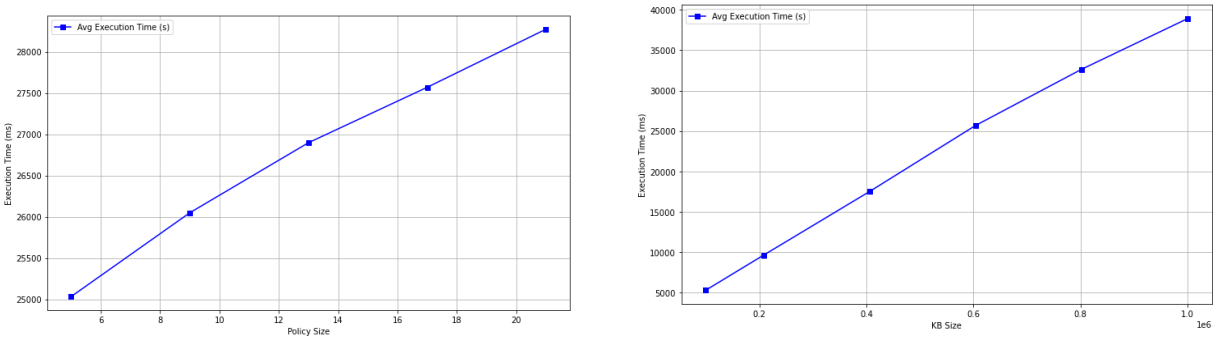
7. Related Work

There are several works that focus on usage control in general and obligation monitoring in particular. UCON [34] 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 [11, 33], 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. Another language, OSL, formalized in Z [23], 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 [29], Ponder [12], KAoS [49], and the DSA policy framework [44]. 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.

In terms of policy standards, ODRL[18] 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 [20] or by defining operational semantics through rule-based reasoning [17, 46]. Notably, Fornara et al. [18] 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. Our extended GUCON model builds upon the obligation states formally defined by Fornara et al. [18]. However, we simplify the GUCON model by including only temporal metaproperties and define the formal semantics of obligation states separately. Currently, a W3C group[19] 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 [8] 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. In contrast, the state of the affairs, represented as a KB, is a fundamental component of our GUCON framework. GUCON also leverages graph patterns, enabling straightforward and dynamic policy re-evaluation against the current state of the world. SHACL is another standard that has been explored

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



(a) Task 1: Execution Time in Terms of Policy Size

(b) Task 2: Execution Time in Terms of KB Size

Figure 6. Experiments Results

for expressing policies. Robaldo et al. [39] propose using SHACL 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 [15], originally developed for managing access control policies. Several studies have extended XACML with UCON capabilities [10, 30]. Another related standard is the Abbreviated Language for Authorization (ALFA)[20], a domain-specific language for specifying authorization policies. However, both XACML (including its extensions) and ALFA lack formal foundations. 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.

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-star and SPARQL-star and introduced an Obligation State Manager that tracks obligation states and checks compliance against the state of affairs.

To assess our approach, we evaluated the extended model using Thörn's criteria for model quality. The evaluation highlighted several strengths: (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 formal semantics; (iv) correctness, as demonstrated through the use case and corresponding test scenarios; (v) mobility, by showing different ways of instantiating the model using Semantic Web languages; and (vi) usability, as GUCON deliberately employs concise syntax to reduce verbosity. We also 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 strengthen interoperability, we aim to develop systematic mappings between different instantiations of GUCON, such as OWL and SHACL. Finally, we plan to address usability by designing intuitive interfaces that enable users to craft and manage GUCON policies more effectively.

[20] ALFA, https://alfa.guide/

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] 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.

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

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

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

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

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

[23] 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.

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

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

[26] 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.
[27] Iannella, Renato and Villata, Serena. Odrl information model 2.2, 2018. URL https://www.w3.org/TR/odrl-model/. Online; accessed 28 April 2025.
[28] 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.
[29] 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.
[30] 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.
[31] Megan Katsumi and Mark Fox. Ontologies for transportation research: A survey. *Transportation Research Part C: Emerging Technologies*, 89, 2018.
[32] Timotej Knez and Slavko Žitnik. Event-centric temporal knowledge graph construction: A survey. *Mathematics*, 11, 12 2023.
[33] Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. Usage control in computer security: A survey. *Computer Science Review*, 4(2), 2010.
[34] Jaehong Park and Ravi Sandhu. The uconabc usage control model. *ACM Transactions on Information and System Security*, 7, 2004.
[35] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *The Semantic Web - ISWC 2006*, 2006.
[36] Rajesh Piryani, Nathalie Aussenac-Gilles, and Nathalie Hernandez. Comprehensive survey on ontologies about event. *CEUR Workshop Proceedings*, 3443, 2023.
[37] Alexander Pretschner, Manuel Hilty, Florian Schütz, Christian Schaefer, and Thomas Walter. Usage control enforcement: Present and future. *IEEE Security & Privacy*, 6(4), 2008.
[38] 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.
[39] 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.
[40] 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.
[41] 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.
[42] 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.
[43] Michael Röder, Denis Kuchelev, and Axel-Cyrille Ngonga Ngomo. Hobbit: A platform for benchmarking big linked data. *Data Science*, 3 (1), 2020.
[44] 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.
[45] 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.
[46] Simon Steyskal and Axel Polleres. Towards formal semantics for odrl policies. In *International Web Rule Symposium*, 2015.
[47] 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.
[48] 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.
[49] 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.
[50] Lluís Vila and Han Reichgelt. The token reification approach to temporal reasoning. *Artificial Intelligence*, 83, 1996.