# Representing and Querying Norm States Using Temporal Ontology-Based Data Access

Evellin Cardoso
*KRDB Research Centre*
*Free University of Bozen-Bolzano*
Bolzano, Italy
ecardoso@unibz.it

Marco Montali
*KRDB Research Centre*
*Free University of Bozen-Bolzano*
Bolzano, Italy
montali@inf.unibz.it

Diego Calvanese
*KRDB Research Centre*
*Free University of Bozen-Bolzano*
Bolzano, Italy
calvanese@inf.unibz.it

***Abstract*—The prominent role of normative primitives for regulating interactions in enterprise systems makes their representation and inspection a crucial task. In particular, querying the states of norms is essential to assess the accountability and compliance of interacting parties, as well as to reconstruct the historical evolution of their mutual contractual relationships. The main issue, however, is that the temporal data stored during the operation of the enterprise system are typically at a low level of abstraction, and do not directly refer to norms and their states. To overcome this problem, we propose the QUEN framework for querying norm states on top of legacy relational databases. QUEN builds on the temporal extension of the well-established framework of Ontology-Based Data Access (OBDA) to semantically represent normative primitives and their temporal states, and to map them to the underlying legacy data. The distinctive feature of QUEN is that mappings are not arbitrarily expressed by the modeler, but are instead automatically synthesized starting from a specification that explicitly indicates how the constitutive components of a normative primitive, as well as its lifecycle, can be reconstructed from the legacy data.**

## I. Introduction

Modern software systems are part of large enterprise systems, themselves part of even larger socio-technical systems (STS) that involves social entities (e.g., autonomous humans or organizations) together with technical components (e.g., IT resources). In order to coordinate the interactions between multiple actors, STS rely on *norms* as the main design construct to specify desired properties to govern such interactions [1]. For example, in a hospital environment, a norm could prohibit a physician to disclose sensitive information from patients, regulating the interactions among the principals involved.

As normative primitives provide a semantics for actors' interactions, their representation becomes crucial for providing guidance for accountability and compliance evaluation. Usually a propositional representation $N(D, C, qp, qd)$ is adopted, which captures a normative relationship $N$ between an expectee $D$ and an expector $C$. In this relationship, the expectee $D$ is accountable for the expector $C$ that, whenever condition $qp$ holds in the system, it will bring about condition $qd$ (i.e., the expectation). By checking the occurrence of events, one can in turn check whether conditions hold, and track how norms evolve through their constitutive states.

This poses a twofold challenge, which we tackle in this paper. The first challenge is about the *representation* of normative primitives. When representing and tracking normative primitives in an enterprise system, one has to consider the presence of complex objects and their interrelationships also when referring to normative primitives. For example, the notion of disclosure authorization of patient data cannot be just characterized propositionally, grounding upfront which are the involved agents: it must be characterized relationally, and then instantiated on actual individuals, tracking how such different instances evolve over time. E.g., a patient may have decided to authorize access to her own data to two different third parties, but with different temporal conditions. This has to be fully taken into account at the representation level. While most works adopt a propositional representation for norms, recent approaches have in fact brought forward a relational approach to commitments and other types of normative primitives [2], [3], [4], [5].

The second challenge is about the actual data reporting on the evolution of normative primitives. The main issue here is that the temporal data stored during the operation of the enterprise system are typically at a low level of abstraction, and do not directly refer to norms and their states. This poses the question on how to *access and query* such low-level data so as to obtain relevant information on normative primitives and their evolution. Such a question is not peculiar to normative aspects, but is in fact pervasive and relates, in general, to the problem of accessing and querying legacy data using conceptual, high-level abstractions that are in line with the knowledge and vocabulary of human experts.

Such a problem has given rise to the well-established paradigm of *Ontology-Based Data Access* (OBDA) [6], [7] (also known as Virtual Knowledge Graphs), which has been applied successfully within the Semantic Web community in a number of scenarios where the access to data poses a major challenge [8]. OBDA is based on the idea that end-users are exposed to the data through a high-level, conceptual representation of the domain of interest, given in the form of an *ontology*, that abstracts away low-level details about the organization and storage of the data itself. The ontology is expressed in terms of a domain vocabulary (of concepts and relations) that is familiar to the users, and moreover it is able to capture complex interrelationships between the conceptual elements of the domain by means of logical axioms. The

2325-6362/19/$31.00 ©2019 IEEE
DOI 10.1109/EDOC.2019.00024

ontology is linked to the underlying legacy data through a declarative *mapping* [6], which intuitively describes, by means of correspondences between queries, how the conceptual elements of the ontology are populated from the data. The mapping is exploited by an OBDA-system to automatically translate user queries formulated over the ontology into SQL queries over the relational data source(s), taking also into account the domain semantics encoded by the ontology axioms so as to enrich the set of provided answers [7].

In our work, we start from Custard [9], which is a framework for the specification of normative primitives (i.e., commitments, authorizations, prohibitions, and powers) equipped with a lifecycle, capturing the states in which these norms may be, and the transitions between states, triggered by punctual events. In Custard, norm states are maintained in a relational information store, and they can be manipulated through expressions that, on the one hand provide a declarative and very compact specification, but on the other hand, present several shortcomings. First, Custard adopts a relational representation of norms, a long advocated feature [2], but still does not represent the *target* of a norm, which would be needed to distinguish between different instances of the norm. Second, Custard supports only limited forms of joins based on equality, and this represents a potentially serious drawback especially when accessing legacy databases, where the structure of the data is not tailored towards the manipulation of norms. Third, the Custard event algebra does not support explicit expirations or violations of norms, determined by the semantics of the domain of interest.

To overcome the above mentioned problems, we propose the QUEN framework for querying norm states on top of legacy relational databases. QUEN builds on a temporal extension of OBDA [10] to semantically represent normative primitives and their temporal states, and to map them to the underlying legacy data. The distinctive feature of QUEN is that mappings are not arbitrarily expressed by the modeler, but are instead automatically synthesized starting from a specification that explicitly indicates how the constitutive components of a normative primitive, as well as its lifecycle, can be reconstructed from the legacy data.

By specifying normative primitives in a fully semantic framework that relies on the well established paradigm of (temporal) OBDA, we lay the foundations for verification, synthesis, monitoring, and query processing in normative systems, although we leave the specifics for future work. In particular, our long-term research goal is to monitor the achievement of normative primitives along their lifecycles in the presence of complex real-world data. As such, our work gives a first step towards a framework for normative and performance monitoring. More specifically, we provide the following contributions: 1) a novel framework for the representation of normative primitives that relies on temporal OBDA; 2) a methodology for lifting generic temporal data stored in a relational system to a fully semantic framework for the representation and querying of norms; 3) a technique, relying on query processing in (temporal) OBDA to automatically transform (semantic) queries expressed over a QUEN specification and execute them over the underlying legacy information sources.

The rest of the paper is structured as follows: Section II provides the research baseline for our work, which includes the Custard language and the temporal OBDA paradigm. Section III presents the QUEN framework through a sequence of methodological steps that start from the low-level temporal data and derive the fully semantic representation of norms. Section IV presents a translation from a specification in QUEN to a temporal OBDA system, in which it becomes possible to express semantic queries over the states of norms. Finally, in Section V we provide conclusions and outline future work.

## II. BASELINE

This section introduces the two main preliminary frameworks on which we rely, namely the QUEN normative primitives from the Custard language and the Ontology-Based Data Access (OBDA) paradigm.

### A. Normative Primitives in Custard

Custard [9] is a framework for the specification of information-based normative primitives (called simply *norms* hereafter), and the retrieval of the states of such primitives from a relational information store. A *norm schema* $\eta$ contains normative (norm) types that socially relate interacting parties and implicitly regulate their interactions: commitments, authorizations, prohibitions, and powers. Each norm instantiates its schema creating a contractual relationship between two agents, an *expector* and an *expectee*, where the expectee is accountable to the expector for the satisfaction of the expectation implied by the contractual relationship. More technically, each norm type comes with a canonical *lifecycle* that captures the different states in which a norm of that type can be in, as well as the possible transitions from one state to the other, and the conditions that induce those transitions. The lifecycles of the different Custard norm types are depicted in Fig. 1. The idea behind such lifecycles is borrowed from the well-established notion of commitment machine [11]. Specifically, once a norm is created, it corresponds to a conditional expectation that comes into force (i.e., is detached to an actual expectation) when the expectee brings about the *antecedent* condition associated to the norm. If the course of execution is so that such a condition cannot be realized, the norm expires. A created or detached norm becomes discharged when its associated *consequent* condition is made true by the expectee agent. The achievement of the consequent indeed witnesses that the expectation associated to the norm has been fulfilled. In the case of prohibitions and commitments, a detached norm may become violated if the course of execution is so that the consequent can never be achieved.

In Custard, conditions are specified with an algebra that employs punctual events as basic building blocks. Events that may occur in the system (e.g, register, sign up, send credentials) are stored in a relational *information schema* $I$, where each event corresponds to a dedicated table equipped with

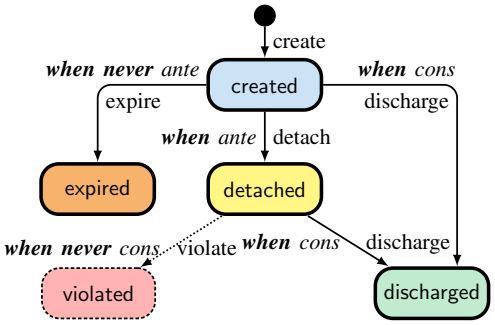


Figure 1: Lifecycle of norm types (from [9]). The violated state exists only for prohibition and commitment.

a timestamp column, and whose tuples record the different instances recorded in the system for that event.

**Example 1 (Inspired from [9]).** The following information schema captures three event types related to the request and access to patient data within a sanitary organization, where the primary key of each relation schema is underlined:

- $Allowed(pid, hid, \underline{discid}, tpid, t)$ – at time $t$, patient $pid$ accepts to disclose her data to third party $tpid$ through health vault provider $hid$. The id value used for $discid$ hence constitutes a "disclosure token" for the patient data.
- $SentCred(hid, tpid, \underline{discid}, t)$ – at time $t$, health vault provider $hid$ sends access credentials to third party $tpid$ using the disclosure token $discid$. The field $discid$ is a foreign key referencing the $Allowed$ relation. Since it is also a primary key for $SentCred$, credentials can be sent at most once for a given entry in $Allowed$.
- $ReqData(tpid, hid, \underline{reqid}, discid, t)$ – at time $t$ the third party entity $tpid$ requests patient data to the health vault provider $hid$, referring to the credentials obtained via $discid$ (which is in fact a foreign key referencing the $Allowed$ relation).
- $Accessed(tpid, hid, \underline{reqid}, discid, t)$ – at time $t$ the third party entity $tpid$ accesses, via health vault provider $hid$, the patient data related to the disclosure token $discid$ (which is a foreign key referencing the $Allowed$ relation). Similarly to the design pattern used for $SentCred$, here the field $reqid$ simultaneously acts as primary key for the $Accessed$ relation, and as a foreign key referencing the $ReqData$ relation. This guarantees that at most one access per request is actually recorded.

The different event tables have to be understood as log tables recording events triggered within the enterprise system. The *effect* induced by each of such events depends on the actual data. For example, credentials are effectively delivered only if the ids of the health vault provider and of the third party actually coincide with those obtained by inspecting the entry of the $Allowed$ relation with matching $discid$. Similarly, data are actually accessed only if the access request is done consistently with the parties involved in the corresponding request, and with the right disclosure token. As we will see, these implicit semantic constraints can be explicitly captured in QUEN when relating the information system to the ontology. ◁

The event algebra uses the tables in $\boldsymbol{I}$ as atomic events, and constructs complex event expressions using logical operators (AND, OR, EXCEPT), aggregation operators (SUM), set operators (COUNT, MIN, MAX, AVG), relative time intervals within which events should occur ([startEvent, endEvent]), and so on. In addition, an ad-hoc form of negation is used to determine when an event expression never holds (e.g., the negation of an event that should occur within a certain deadline corresponds to checking that the event did not occur before the deadline expiration time).

Specifically, the progression of a specific norm is defined in Custard by qualifying the *create*, *detach*, and *discharge* transitions with corresponding event expressions that instantiate the expector and expectee agents, and the corresponding transition timestamps. The *expire* and *violate* transitions are then implicitly qualified by negating the antecedent and consequent event expressions respectively associated to the detach and discharge transitions.

**Example 2 (Inspired from [9]).** Consider the information schema of Example 1. The following Custard specification defines an authorization norm regulating the possibility of accessing sensible patient data:

| | |
|---|---|
| **authorization** | $DisclosureAuth\ tpid$ **by** $hid$ |
| **create** | $SentCred$ |
| **detach** | $ReqData$ |
| **discharge** | $Accessed[ReqData + 1, ReqData + 10]$ |

In particular, the specification indicates that a disclosure authorization is created by $hid$ for $tpid$ at time $t$ whenever there exists an entry in the $SentCred$ relation relating $hid$, $tipd$, and $t$. A consequent detach of the authorization occurs at a consequent time $t_2$ whenever an entry in $ReqData$ exists with time $t_2$ and matching the common fields of $ReqData$ and $SentCred$. Finally, the authorization is discharged at time $t_3$ if a consequent matching entry in $Accessed$ exists at a time $t_3$, in such a way that $t_2 + 1 < t_3 < t_2 + 10$, where 1 and 10 are time units.

As dictated by the notion of negation used in Custard event patterns, the authorization is considered to be violated if, upon detachment, no access entry exists between $t_2+1$ and $t_2+10$.◁

The example shows that Custard is extremely compact and declarative when specifying the lifecycle of norms, but also highlights its main, three shortcomings:

- *Lack of an explicit norm target.* While Custard lifts the representation of norms from a propositional to a relational level, as long advocated in [2], [3], it still does not explicitly tackle what is the *target* of the norm: that is, the object—or combination of objects—that uniquely identifies an instance of that norm, and makes it possible to distinguish it from other instances of the same norm. E.g., the authorization captured in Example 2 only refers to the two involved actors, but does not capture that the authorization *is about* a disclosure token, in turn related to a patient. Consequently, it would not be possible to distinguish two separate instances of *DisclosureAuth*

relating *the same* third party and vault provider, but *for different* disclosure tokens.

- *Lack of flexibility in querying relations*. Custard assumes that when relating relations to each other, joins are implicitly done based on equality of field names. A more general approach relying on the full SQL query language is needed to handle the situation where the underlying information schema is a legacy one, and consequently require the power of complex queries so as to extract the required information. Already in Example 1, one would need more powerful queries than those supported by Custard to retrieve the disclosure tokens that have been potentially misused. For example, the SQL query

  **SELECT** $r.discid$ **FROM** $ReqData\ r, SentCred\ c$
  **WHERE** $r.discid = c.discid$ **AND** $r.tpid \neq c.tpid$

  can be used to retrieve potentially misused disclosure tokens, i.e., tokens used to request data by a third party that does not match the one to which the credentials for that token have been sent.
- *Lack of explicit expirations and violations*. While the Custard event algebra has the nice property of being able to express implicit expiration/violation of a norm, the specification of norms does not allow to complement such implicit transitions with explicit expirations/violations determined by the semantics of the domain under study, with conditions that cannot be simply ascribed to the "absence" of the norm antecedent or consequent. For example, one could use a variant of the SQL query shown in the previous bullet so as to induce a violation of an authorization for which the disclosure token has been potentially misused.

Given our final goal to create a framework for norm querying on top of legacy information systems, we take inspiration from Custard but lift it to a full, semantical setting where the aforementioned key limitations are resolved. Since our framework uses, as a technical basis, a temporal variant of the OBDA paradigm, we next provide a gentle introduction to temporal OBDA.

### B. Temporal Ontology-Based Data Access

The Ontology-Based Data Access (OBDA) paradigm consists of a novel philosophy of conceiving information systems in which conceptual schemas (ontologies) are used as an intermediate layer for accessing and querying the data stored in legacy information systems [7]. The ultimate goal of the OBDA paradigm is to provide a vocabulary for end-users based on domain notions, thus allowing a transparent access to legacy information systems, as it abstracts away from implementation details on how data is concretely stored.

Ontologies have been extensively used to provide the conceptualization of a domain of interest, and mechanisms for reasoning about it. The formal foundations for ontologies are provided by Description Logics (DLs) [12], which are logics specifically designed to represent structured knowledge and to reason upon it.

In DLs, the domain to represent is structured into classes of objects of interest that have properties in common, and these properties are explicitly represented through relevant relationships that hold among the classes. *Concepts* denote classes of objects, and *roles* denote (typically binary) *relations* between objects. Both are constructed, starting from atomic concepts and roles, by making use of various constructs, and the set of allowed constructs characterize a specific DL. The knowledge about the domain is then represented by means of a DL ontology, where a separation is made between general structural knowledge and specific extensional knowledge about individual objects. The structural knowledge is provided in a so-called *TBox* (for "Terminological Box"), which consists of a set of universally quantified assertions that state general properties about concepts and roles. The extensional knowledge is represented in an *ABox* (for "Assertional Box"), consisting of assertions on individual objects that state the membership of an individual in a concept, or the fact that two individuals are related by a role.

Among the various DLs, lightweight description logics, such as the OWL2QL profile of OWL 2, have been specifically designed to query and reason about huge amounts of data, trading off between expressiveness and tractability. Logics within such a family are able to capture front-end ontology design languages such as the core constructs of UML class diagrams (with the exception of covering) [13]. Thus, in the following we blur the distinction between UML ontologies and their corresponding OWL2QL TBoxes.

In this paper, we consider the widespread setting where data are not maintained in the form of an ontology ABox, but are instead stored inside legacy information systems. As storage mechanism, we consider in particular relational database with constraints (such as keys and foreign keys). This is fully compatible with the notion of information schema used in Custard (cf. Section II-A), without however posing any expectation on the structure of the relation schemas forming the database. Since our focus is on the evolution of norms implicitly represented in the data, the only assumption we do is that some of the relation schemas are equipped with timestamp fields.

In such a setting, the (TBox of the) ontology is used not only to capture relevant structural properties of the domain, but also acts as a conceptual data schema providing a high-level view over the underlying data.

With these premises, an OBDA specification is a tuple $\langle \mathcal{S}, \Sigma, \mathcal{O}, \mathcal{M} \rangle$, where:

- $\mathcal{S}$ is a *relational database schema* with constraints;
- $\Sigma$ is a vocabulary of concepts and relations;
- $\mathcal{O}$ is a lightweight ontology expressing constraints over $\Sigma$;
- $\mathcal{M}$ is a set of *mapping assertions* that conceptually link the concepts and relations in $\mathcal{O}$ to the underlying database schemas $\mathcal{R}$.

$\mathcal{M}$ serves two purposes. On the one hand, it declaratively specifies how to extract data from a database conforming to $\mathcal{R}$, using the standard SQL query language. On the other hand,

it indicates how the answers extracted using SQL queries are used to (virtually) populate the (ABox of) the ontology $\mathcal{O}$.

Technically, a mapping assertion (or simply mapping) has the form

$$Q(\vec{x}) \;\rightsquigarrow\; G(\vec{\mathbf{t}}(\vec{y}))$$

where

- $Q$, called the *source part* of the mapping, is an SQL query over the schema $\mathcal{S}$ with answer variables $\vec{x}$.
- $G(\vec{\mathbf{t}}(\vec{y}))$, called the *target part* of the mapping, is a conjunction of atoms whose predicate symbols are atomic concepts and roles in $\mathcal{O}$, and where $\vec{\mathbf{t}}(\vec{y})$ represents the arguments of the predicates in the atoms. The variables $\vec{y}$ are among the answer variables $\vec{x}$ of the query in the source part. $\vec{\mathbf{t}}(\vec{y})$ are terms obtained by applying function symbols $\vec{\mathbf{t}}$ to $\vec{y}$; such terms provide the basis to construct semantic objects in the ontology starting from values extracted from the underlying database, resolving the so-called *impedance mismatch*.

The main feature of an OBDA system is at query time. Here, users can submit conceptual queries over $\mathcal{O}$ (e.g., using the SPARQL query language) to the system, which uses $\mathcal{M}$ to automatically reformulate such queries into corresponding SQL queries that can directly be processed by an off-the-shelf relational database engine. For a survey of application domains in which this approach has been successfully deployed, see [7].

Recently, the OBDA paradigm has been applied in settings where the temporal dimension is predominant. In its standard form, however, the resulting framework cannot be used to describe temporal ontologies, formulate temporal queries, and obtain answers together with their temporal validity intervals. This is why the *temporal OBDA* (TOBDA) approach [10] has been brought forward. The TOBDA approach is exemplified in a use case from Siemens for monitoring steam and gas turbines. Treating time as a first-class citizen is essential in the context of this paper, to capture how norms evolve across their possible states.

A temporal ontology contains structural and temporal concepts and relations. As for the temporal part, the ABox assigns to corresponding facts their validity intervals, describing when those facts actually hold. A temporal fact is then represented using notation $F@I$, where $F$ is a fact and $I$ its validity interval. Formally, a TOBDA system [10][1] is a tuple $\langle \mathcal{S}, \Sigma_s, \Sigma_t, \mathcal{O}_s, \mathcal{T}, \mathcal{M}_s, \mathcal{M}_t \rangle$, where:

- $\langle \mathcal{S}, \Sigma_s, \mathcal{O}_s, \mathcal{M}_s \rangle$ is a classical OBDA system, where $\mathcal{R}$ contains timestamped relation schemas;
- $\Sigma_t$ is a vocabulary of temporal predicates;
- $\mathcal{T}$ is a set of temporal rules over $\Sigma_t$, defining the occurrence conditions of complex events, together with their temporal validity;
- $\mathcal{M}_t$ is a set of temporal mapping assertions linking $\mathcal{R}$ to elements in $\Sigma_t$, also specifying their validity intervals using timestamps retrieved from the database.

[1]For simplicity, we do not consider here static rules, but they can seamlessly be added to QUEN.

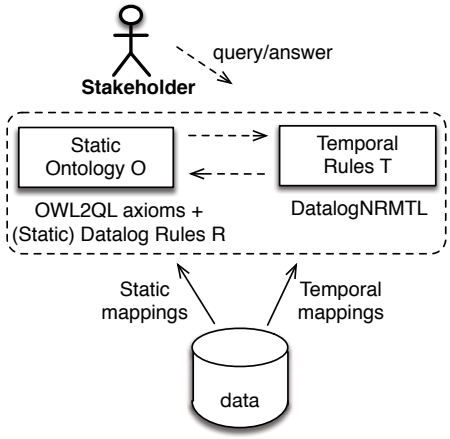


Figure 2: Schematic representation of TOBDA architecture [10]

The language used to formalize temporal rules is that of datalog$_{nr}$MTL, a non-recursive datalog program extended with metric temporal logic (MTL) operators interpreted over the reals.

The form of each temporal mapping assertion is

$$Q(\vec{x}, t_s, t_e) \rightsquigarrow\; G(\vec{\mathbf{t}}(\vec{y}))@\langle t_s, t_e \rangle$$

where $Q(\vec{x}) \rightsquigarrow G(\vec{\mathbf{t}}(\vec{y}))$ is a standard mapping assertion, $t_s$ matches with a timestamp, $t_e$ matches either with a timestamp of the special constant $\infty$, and $\langle / \rangle$ are meta-symbols to be instantiated with parenthesis or square brackets depending on whether the extremes of the temporal interval going from $t_s$ top $t_e$ are included.

Fig. 2 schematically depicts the elements of the TOBDA approach.

## III. The QUEN framework

This section presents the QUEN framework. The framework starts from the main Custard ontological entities and norm lifecycle specification as described in Section II-A. The main difference is that while Custard assumes a high-level information schema that explicitly describes events, QUEN starts from a generic database schema that stores timestamped data, in the style of TOBDA.

We introduce the main components of QUEN as methodological steps that lift the low-level temporal data to a fully semantic framework for norm representation. The querying functionality of the framework is then discussed in Section IV.

### A. Step 1: Building the Static Ontology

The first step of the methodology consists in the conceptual modeling of the domain knowledge capturing the main concepts and relationships necessary to define norms and their participants, as well as to understand the data contained in the legacy database.

Differently from the case of a generic (T)OBDA setting, QUEN is centered around normative primitives and their participants. Consequently, we can define once and for all an upper ontology containing those norm-related concepts and

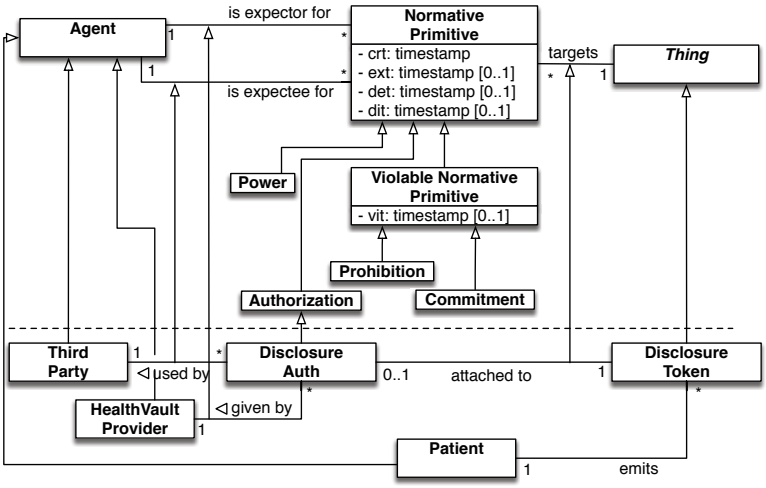


Figure 3: Static ontology in QUEN. The part above the dashed line is a fixed, upper ontology present in every QUEN system.

relationships that exist independently of the specific domain at hand.

The top part of Figure 3 shows such upper ontology, which we call in the following $\mathcal{O}_n$. This ontology is designed taking inspiration from the Custard structural elements. We make use of UML to graphically capture $\mathcal{O}_n$. With a slight abuse of notation, we use the same symbol to denote the UML class diagram of an ontology, and its corresponding formalization into lightweight DLs (cf. Section II-B).

Specifically, $\mathcal{O}_n$ captures the different types of normative primitives used in Custard: *authorization*, *prohibition*, *power* and *commitment*, all subclasses of the general notion of *normative primitive* (norm for short). A norm comes with various timestamp attributes, tracking their creation time ($crt$) and, optionally, expiration time ($ext$), detachment time ($det$), and discharge time ($dit$). Prohibitions and commitments also get, optionally, the information about their violation time ($vit$). While the creation timestamp is mandatory for the existence of a norm instance, the other timestamps are optional (for this reason we have the cardinality [0..1]), and get filled or not depending on how the instance evolved. Additional constraints (not shown in the UML diagram) can be imposed to ensure mutual exclusion between $ext$ and $det$, $vit$ and $dit$, and $ext$ and $dit$. Such attributes, in fact, cannot coexist, as they are associated to alternative evolutions of the norm. Recall that this ontology is static, so such timestamp attributes are treated as standard attributes, without attaching to them any temporal semantics.

In addition, $\mathcal{O}_n$ introduces the general notion of *agent*, as well as the two key relationships indicating that each normative primitive has one expector agent, and one expectee agent. Differently from Custard, we also include a further relationship to explicitly indicate what is the *target* of a normative primitive. The target class, together with the expector and expectee agent classes, provides an identification principle for the normative primitive, so that each instance of the normative primitive is uniquely identified by a corresponding triple of instances defining the actual expector, expectee, and target object.

When QUEN is used to capture norms and participants of a domain of interest, $\mathcal{O}_n$ is complemented by a specific, more concrete ontology, which refines the general upper notions through specialization of classes and relationships. This allows the modeler to conceptually capture the relevant, domain-specific (sub)types of norms, agents, and target objects.

**Example 3.** The lower part of Figure 3 shows how the upper ontology $\mathcal{O}_n$ is specialized to reflect the disclosure authorization from Example 2. ◁

### B. Step 2: Specifying the Lifecycle of Norms

This is the central step of the methodology. It is concerned with the specification of how the domain-specific norms introduced in the previous step evolve through their different states, depending on the patterns of data stored in the legacy database. To do so, we take again inspiration from Custard, and lift the specification of a norm lifecycle dealing with full SQL.

Let $D$ be the timestamped legacy database of interest. A QUEN lifecycle specification for norm $N$ makes use of SQL queries expressed over the schema of $D$ to retrieve data objects and timestamps to then indicate which instances of $N$ exist, and when and how they evolve. Such queries may indirectly make use of the answers produced by other queries (e.g., in Example 2 the discharge timestamp depends on the detach timestamp). When formulating such queries, we then employ "parameters" that act as placeholders for constant objects, and that will be bound to actual objects depending on the answers obtained via other queries. This is similar to the notion of prepared statement in SQL.

Notationally, we use $Q\vec{p}(\vec{x}, t)$ to indicate a SQL query with parameters $\vec{p}$ and answer variables $\vec{x}$ and $t$, where $t$ is an answer variable matching with timestamp values. This is a compact notation for a SQL query of the form

**SELECT** $\vec{x}, t$ **FROM** ... **WHERE** $\varphi(\vec{x}, t, \vec{p})$

where $\varphi$ is a (possibly nested) SQL condition over $\vec{x}$, $t$, and parameters in $\vec{p}$.

With this notation at hand, we are now ready to define our lifecycle specification. Let $N$ be a domain-specific norm class, and:

- $T \in \{Authorization, Power, Prohibition, Commitment\}$ be the norm type of $N$;
- $R_d$ be a domain-specific relations that is attached to $N$ and is a sub-relation of the expector relation in $\mathcal{O}_n$ (thus qualifying the domain-specific expector for $N$);
- $R_c$ be a domain-specific relation that is attached to $N$ and is a sub-relation of the expectee relation in $\mathcal{O}_n$ (thus qualifying the domain-specific expectee for $N$);
- $R_t$ be a domain-specific relationship that is attached to $N$ and is a sub-relation of the target relation in $\mathcal{O}_n$ (thus qualifying the domain-specific target for $N$).

A QUEN lifecycle specification for this combination of elements has the following form:

| | |
|---|---|
| **T** | $N\ R_d\ d\ R_c\ c\ R_t\ o$ |
| **create** | $Q^{cr}(d, c, o, t_{cr})$ |
| **expire** | $Q^{ex}_{d,c,o,t_{cr}}(t_{ex})$ |
| **detach** | $Q^{de}_{d,c,o,t_{cr}}(t_{de})$ |
| **discharge** | $Q^{di}_{d,c,o,t_{cr},t_{de}}(t_{di})$ |
| [**violate** | $Q^{vi}_{d,c,o,t_{cr},t_{de}}(t_{vi})$ ] |

where the last line is only present if $T \in \{Prohibition, Commitment\}$. The first line declares the norm $N$, by semantically identifying its expector, expectee, and target object using the vocabulary of the corresponding relations attached to $N$. These three data elements are obtained through the SQL query associated to the norm creation, and are used to identify instances of the norm. The expiration and detach queries may use such data elements as parameters, together also with the creation timestamp. The discharge and violate queries may also use as parameter the detach timestamp.

The similarities and differences with Custard should be apparent from the definition. In QUEN:

- the norm declaration uses the vocabulary of the ontology, and explicitly points to the target object on which the norm operates.
- Queries are arbitrary SQL queries that can make use of arbitrary SQL statements such as aggregations and the like, but that do not come with the surface, event-based interface syntax of Custard.
- Since transitions are defined via arbitrary SQL queries, it is not possible to implicitly define expiration and violation as the impossibility of realizing the detach/discharge queries; such transitions have to be explicitly defined.

**Example 4.** To clarify how QUEN lifecycle specifications work, let us redefine the authorization introduced in Example 2, considering as underlying database schema the one in Example 1. In particular, we want to refine the specification from Example 2 by:

- explicitly indicating that a disclosure authorization targets a disclosure token, as captured by the $Allowed$ relation schema;
- each disclosure authorization is created from an entry in the $SentCred$ relation, provided that the involved agents match those associated to the same disclosure token within $Allowed$. ◁

The same matching conditions over the involved agents may be applied also when detaching and discharging the disclosure authorization, but we do not do this and keep the specification compact.

The resulting QUEN lifecycle specification is shown in Figure 4. Notice the usage of the disclosure token as a correlation id to inspect all the different relation schemas of interest.

It is important to stress that a QUEN lifecycle specification may not be well-formed. This happens if, e.g., the detach query returns more than one answer, creating ambiguity on when the transition actually happened. This depends on how the specification has been modeled, not only considering the queries used therein, but also the actual data stored in the underling database. In our running example, the specification of the disclosure authorization would not be well-formed in case multiple data requests are issued for the same disclosure token: it would not be clear anymore when the authorization has been detached, becoming in place. We will come back to this important aspect in Section IV-C.

### C. Step 3: Devising Static Mappings

While QUEN lifecycle specifications focus on domain-specific norm classes and their surrounding relations, other elements present in the ontology may not be directly mentioned. Those elements consequently miss a linkage to the underlying data. This can be restored by introducing additional static mappings, in the style of OBDA. This can be done only if the underlying database actually contains data that could indeed provide the basis to describe the extension of some elements. If not, no explicit linkage between such elements in the ontology and the underlying data is established, which is perfectly fine given that ontologies work under incomplete information.

**Example 5.** Consider again the domain-specific part of the ontology shown in Figure 3. The lifecycle specification in Figure 4 focuses on the $DisclosureAuth$ class and surrounding relations (which implicitly includes also the endpoint classes attached to those relations, given that UML univocally assigns the endpoint classes to each binary relation). However, it does not mention directly the $Patient$ class, nor the corresponding $emits$ relation. The underlying database schema introduced in Example 1 actually provides us the raw data to characterize the extension of such elements: it is enough to inspect the $Allowed$ relation and filter it by retaining the $pid$ and $discid$ fields. We can then construct the following mapping,

**SELECT** $pid, discid$ **FROM** $Allowed$
$\rightsquigarrow$ $emits(\textbf{pat}(pid), \textbf{dtoken}(discid))$

| | |
|---|---|
| **authorization** | $DisclosureAuth$ **used by** $tp$ **given by** $h$ **attached to** $d$ |
| **create** | **SELECT** $c.tpid$ **AS** $tp$, $c.hid$ **AS** $h$, $c.discid$ **AS** $d$, $c.t$ **AS** $t_{cr}$ |
| | **FROM** $SentCred\ c, Allowed\ a$ **WHERE** $c.discid = a.discid$ **AND** $c.tpid = a.tpid$ **AND** $c.hid = a.hid$ |
| **detach** | **SELECT** $r.t$ **AS** $t_{de}$ **FROM** $ReqData\ r$ **WHERE** $r.discid = d$ **AND** $r.t > t_{cr}$ |
| **discharge** | **SELECT** $a.t$ **AS** $t_{di}$ **FROM** $Accessed\ a$ **WHERE** $a.discid = d$ **AND** $a.t \geq t_{de} + 1$ **AND** $a.t \leq t_{de} + 10$ |

Figure 4: QUEN lifecycle specification of the disclosure authorization on top of the database schema of Example 1.

where object constructors simply use (abbreviations of) the names of the corresponding endpoint classes. Notice that this mapping also implicitly populate the *Patient* class with **pat**$(pid)$, given that the domain of *emits* is *Patient* as dictated by the ontology. ◁

### D. Putting Everything Together

We are now ready to define a QUEN system, putting together the various elements introduced in Sections III-A, III-B, and III-C. A QUEN system is a tuple $\langle \mathcal{S}, \Sigma, \mathcal{O}, \mathcal{L}, \mathcal{M} \rangle$, where:

- $\mathcal{S}$ is a relational database schema;
- $\Sigma$ is a vocabulary,
- $\mathcal{O} = \mathcal{O}_n \uplus \mathcal{O}_{ds}$ is an ontology over $\Sigma$ constituted by the upper ontology $\mathcal{O}_n$, and its domain-specific specialization $\mathcal{O}_{ds}$;
- $\mathcal{L}$ is a set of QUEN lifecycle specifications with queries over $\mathcal{S}$, and containing at least one specification per class $N$ in $\mathcal{O}_{ds}$ that specializes a norm class;
- $\mathcal{M}$ is a set of static mappings linking $\mathcal{S}$ to $\mathcal{O}$.

A QUEN system semantically represents norms and their lifecycle by suitably inspecting a database schema. However, it cannot be directly used to formulate queries over norms and their states, and obtain answers computed starting from the raw data contained in the database. This is what we tackle next.

## IV. From QUEN to TOBDA

To use QUEN for querying legacy data and reconstruct the state of modeled norms, we provide a translation mechanism that, given a QUEN system $\mathcal{Q} = \langle \mathcal{S}, \Sigma, \mathcal{O}, \mathcal{L}, \mathcal{M} \rangle$, produces a corresponding TOBDA system $tobda(\mathcal{Q}) = \langle \mathcal{S}', \Sigma_s, \Sigma_t, \mathcal{O}_s, \mathcal{T}, \mathcal{M}_s, \mathcal{M}_t \rangle$ using the following approach:

- $\mathcal{S}' = \mathcal{S}$, $\Sigma_s = \Sigma$, $\mathcal{O}_s = \mathcal{O}$.
- $\Sigma_t$ contains unary temporal predicates created, expired, detached, discharged, and violated, whose single argument is a norm instance; these are the core temporal predicates that capture which norm instances are in which states, and when.
- $\mathcal{T} = \emptyset$;
- $\mathcal{M}_s$ contains $\mathcal{M}$ and additional static mappings extracted by processing the lifecycle specifications in $\mathcal{L}$, as described next;
- $\mathcal{M}_t$ contains temporal mappings extracted by processing the lifecycle specifications in $\mathcal{L}$, as described next.

The interesting part of the translation is how each specification in $\mathcal{L}$ is encoded into a set of static and temporal mappings. This is what we tackle next.

### A. From Lifecycle Specifications to Mappings

As a preliminary step for the translation, we need to define how a query with parameters can be suitably *merge* with a query providing those parameters, so as to obtain a standard, SQL query as result. This is done by simply computing their *join* (in the standard SQL sense).

Specifically, let $Q^1(\vec{x}, t_1)$ be a query without parameters of the form

**SELECT** $\vec{x}, t_1$ **FROM** $\psi_1$ **WHERE** $\varphi_1(\vec{x}, t_1)$

and $Q^2_{\vec{x},t_1}(\vec{y}, t_2)$ be a query using as parameters the answer variables of $Q_1$, and of the form

**SELECT** $\vec{y}, t_2$ **FROM** $\psi_2$ **WHERE** $\varphi_2(\vec{y}, t_2, \vec{x}, t_1)$

Then, the *join* of $Q^1$ into $Q^2$, written $Q^1 \bowtie Q^2$, is the query

**SELECT** $\vec{x}, t_1, \vec{y}, t_2$ **FROM** $\psi_1, \psi_2$ **WHERE** $\varphi_1(\vec{x}, t_1)$
**AND** $\varphi_2(\vec{y}, t_2, \vec{x}, t_1)$

**Example 6.** Consider the create and detach queries from the specification in Figure 4. Their join gives raise to the query:

**SELECT** $c.tpid$ **AS** $tp$, $c.hid$ **AS** $h$, $c.discid$ **AS** $d$,
$c.t$ **AS** $t_{cr}$, $r.t$ **AS** $t_{de}$
**FROM** $SentCred\ c, Allowed\ a, ReqData\ r$
**WHERE** $c.discid = a.discid$ **AND** $c.tpid = a.tpid$
**AND** $c.hid = a.hid$ **AND** $r.discid = d$ **AND** $r.t > t_{cr}$

◁

Consider a lifecycle specification for norm class $N$ and related relations, of the form:

| | |
|---|---|
| **T** | $N\ R_d\ d\ R_c\ c\ R_t\ o$ |
| **create** | $Q^{cr}(d, c, o, t_{cr})$ |
| **expire** | $Q^{ex}_{d,c,o,t_{cr}}(t_{ex})$ |
| **detach** | $Q^{de}_{d,c,o,t_{cr}}(t_{de})$ |
| **discharge** | $Q^{di}_{d,c,o,t_{cr},t_{de}}(t_{di})$ |
| [**violate** | $Q^{vi}_{d,c,o,t_{cr},t_{de}}(t_{vi})$ ] |

In addition, assume that the endpoint classes related to $N$ via $R_d$, $R_c$, and $R_t$ are respectively $A_d$, $A_c$, and $C$. Consistently with the static mappings in $\mathcal{M}$, defined as of Section III-C, we always use the very same name (or, consistently, an abbreviated version) of the class name to obtain the function symbol used to construct objects of such class in a mapping specification. We also use the abbreviated function symbol **t** for the timestamp class.

With this basis, the create entry provides the basis for extracting an instance of $N$ and its main constitutive components,

namely expector, expectee, and target object. In particular, we construct the static mapping

$$\begin{aligned} Q^{cr}(d,c,o,t_{cr}) \rightsquigarrow\ & R_t(\mathbf{n}(d,c,o),\mathbf{C}(o)),\\ & crt(\mathbf{n}(d,c,o),\mathbf{t}(t_{cr}))\\ & R_d(\mathbf{n}(d,c,o),\mathbf{a_d}(d)),\\ & R_c(\mathbf{n}(d,c,o),\mathbf{a_c}(c)), \end{aligned} \tag{1}$$

Recall that, also in this case, the mapping has the indirect effect of populating the endpoint classes of the relations involved in the mapping, that is, $N$, $A_d$, $A_c$, and $C$.

**Example 7.** By compactly indicating the create query of Figure 4 as $Q^{cr}(tp,h,d,t_{cr})$, the corresponding static mapping is:

$$\begin{aligned} Q^{cr}(tp,h,d,t_{cr}) \rightsquigarrow\ & crt(\mathbf{da}(tp,h,d),\mathbf{t}(t_{cr})),\\ & attachedTo(\mathbf{da}(tp,h,d),\mathbf{dtoken}(d)),\\ & usedBy(\mathbf{da}(tp,h,d),\mathbf{thParty}(tp)),\\ & givenBy(\mathbf{da}(tp,h,d),\mathbf{hvProv}(h)) \end{aligned}$$

The other entries in the specification give rise to two types of mappings: a static mapping populating the corresponding timestamp attribute, and a temporal mapping describing if and when the norm has been in a specific state of its lifecycle (cf. Figure 1). The left part of the mapping is formed by suitably joining the query used in the entry, with those in the other entries that provide parameters.

Let us specifically consider the detach entry (the expire entry works analogously). The static mapping is generated, as follows:

$$\begin{aligned} & Q^{de}_{d,c,o,t_{cr}}(t_{de}) \bowtie Q^{cr}(d,c,o,t_{cr})\\ & \quad \rightsquigarrow det(\mathbf{n}(d,c,o),\mathbf{t}(t_{de})) \end{aligned} \tag{2}$$

Notice that this mapping is applied only if the left-hand side returns an answer (i.e., if the norm instance $\mathbf{n}(d,c,o)$ has been actually subject to a detach, expressed according to $Q^{de}$).

As for the temporal part, we need instead two mappings. The first mapping accounts for the situation where the norm instance $\mathbf{n}(d,c,o)$ has been created, and eventually detached. This means that $\mathbf{n}(d,c,o)$ was in state created for the definite time interval going from the creation to the detachment time. In particular, we get:

$$\begin{aligned} & Q^{de}_{d,c,o,t_{cr}}(t_{de}) \bowtie Q^{cr}(d,c,o,t_{cr})\\ & \quad \rightsquigarrow \mathsf{created}(\mathbf{n}(d,c,o))@[t_{cr},t_{de}) \end{aligned} \tag{3}$$

The second mapping accounts instead for the situation where the norm instance has been created, but consequently never detached. The absence of the detachment can be captured by conjoining the WHERE part of $Q^{cr}$ (which extracts the norm instance and its creation time) with a SQL statement wrapping the join of $Q^{de}$ and $Q^{cr}$ inside the NOT EXISTS SQL statement (which witnesses that no detachment time can be extracted). If this is the case, then the norm instance indefinitely stays in the created state. Technically, we get:

$$\begin{aligned} & Q^{cr}(d,c,o,t_{cr})\\ & \textbf{AND NOT EXISTS}\\ & \quad \left(Q^{de}_{d,c,o,t_{cr}}(t_{de}) \bowtie Q^{cr}(d,c,o,t_{cr})\right)\\ & \quad\quad \rightsquigarrow \mathsf{created}(\mathbf{n}(d,c,o))@[t_{cr},\infty) \end{aligned} \tag{4}$$

A similar approach is followed to create static and temporal mappings from the discharge and violate entries. We consider only the case of discharge (that of violation is equivalent). The static mapping extracts the discharge timestamp as follows:

$$\begin{aligned} & Q^{di}_{d,c,o,t_{cr},t_{de}}(t_{di}) \bowtie \left(Q^{de}_{d,c,o,t_{cr}}(t_{de}) \bowtie Q^{cr}(d,c,o,t_{cr})\right)\\ & \quad \rightsquigarrow dit(\mathbf{n}(d,c,o),\mathbf{t}(t_{di})) \end{aligned} \tag{5}$$

The two temporal mappings focus on the detached state and the discharged one. If a discharge time can be retrieved, it means that the norm instance stayed detached for a while, eventually being discharged. Thus, we get:

$$\begin{aligned} & Q^{di}_{d,c,o,t_{cr},t_{de}}(t_{di}) \bowtie \left(Q^{de}_{d,c,o,t_{cr}}(t_{de}) \bowtie Q^{cr}(d,c,o,t_{cr})\right)\\ & \quad \rightsquigarrow \mathsf{detached}(\mathbf{n}(d,c,o))@[t_{de},t_{di}),\\ & \quad\quad \mathsf{discharged}(\mathbf{n}(d,c,o))@[t_{di},\infty). \end{aligned} \tag{6}$$

The other situation that has to be handled is the one in which the norm instance was detached, but never consequently discharged. This means that we can retrieve the detachment timestamp, but not the discharge one. Technically:

$$\begin{aligned} & \left(Q^{de}_{d,c,o,t_{cr}}(t_{de}) \bowtie Q^{cr}(d,c,o,t_{cr})\right)\\ & \textbf{AND NOT EXISTS}\\ & \left(Q^{di}_{d,c,o,t_{cr},t_{de}}(t_{di}) \bowtie \left(Q^{de}_{d,c,o,t_{cr}}(t_{de}) \bowtie Q^{cr}(d,c,o,t_{cr})\right)\right)\\ & \quad \rightsquigarrow \mathsf{detached}(\mathbf{n}(d,c,o))@[t_{de},\infty) \end{aligned} \tag{7}$$

We close by noticing that the temporal mappings are chained in such a way that from the moment where a norm instance is created, it is always associated, in each time point, to a normative state.

### B. Querying Norms

Now that we know how to translate a QUEN system $\mathcal{Q}$ into a corresponding TOBDA system $tobda(\mathcal{Q})$, we can leverage the full power of temporal TOBDA semantic queries to express complex queries on the states of norms over a legacy database (conforming to the database schema in $\mathcal{Q}$). Examples of queries are:

- Retrieve all norm instances that are violated at a given timestamp.
- Obtain all the detached commitments whose expectee is an agent of interest.
- Check if there is a detached authorization of a specific type, for a given pair of agents and a given target object. If so, retrieve the corresponding time interval.

The answers returned by such queries can be used by an auditor or business analyst to ascertain the normative state of a company starting from the actual data stored inside its information system(s). This, in turn, can be used to ascertain conformance and compliance.

### C. Debugging a QUEN System

As argued in Section III-B, a modeler may write queries that, once embedded into a QUEN specifications and applied on top of a specific database, lead to "wrong answers". Specifically, two errors may arise:

- *timestamp ambiguity*: a normative primitive gets associated to multiple timestamps for the same transition;
- *state superposition*: a normative primitive belongs to two states at the same time.

The first error may arise if the query associated to the corresponding transition in the lifecycle specification returns more than one answer. The second may arise if the queries associated to two transitions return the same timestamp.

To detect, and debug, such issues, we can again take advantage from the translation of QUEN into TOBDA. In the case of timestamp ambiguity, one of the mappings (1), (2), (5) would produce two facts for the corresponding timestamp role. However, this would lead to an inconsistent ontology: as shown in Figure 3, timestamps are in fact functional. Such an inconsistency can be checked using standard TOBDA reasoning services, but does not give any hint on the root cause for inconsistency. However, we can again leverage the TOBDA framework to have a fine-grained understanding of such a root cause, using standard techniques [13]. Specifically, it is possible to automatically construct a SQL query that, once submitted to the underlying database, returns those norm instances that have at least two creation times (and similarly for the other time attributes).

The case of state superposition can instead be simply handled by formulating suitable semantic queries that retrieve those norm instances that are simultaneously present in two states. By inspecting the temporal mappings, a case of state superposition can only arise if the norm instance *simultaneously* undergoes a transition to two different states. Hence, to retrieve all norm instances that experienced a superposition of state violated and discharged (and when this undesired superposition arose), we can issue the following query:

$$Q_{dv}(n,t) = \mathsf{violated}(n)@[t,t_1) \wedge \mathsf{discharged}(n)@[t,t_2)$$

A similar approach can be adopted to check different forms of superposition.

## V. Conclusion

We have presented QUEN, a framework for normative primitives, which starts from the norm lifecycle specification of Custard, and exploits temporal OBDA to lift generic temporal data to represent and query norms at a fully semantic level.

We have relied on temporal OBDA mainly to represent norms, agents, and target objects, and the elements corresponding to the different components of a QUEN system. We observe, however, that we can leverage the full power of the OBDA paradigm to encode additional knowledge about the domain of interest, and take it into account when querying norms and understanding their lifecycle. Also, one can rely on the inference services of ontology-based systems to support the debugging of a specification. In that respect, it is important to observe that we have presented a conceptual framework, but we can leverage existing implementations of (temporal) OBDA systems to immediately make this framework operational [14], [10]. As further future work, we plan to investigate how the framework and methodological approach presented here can be extended towards the problem of monitoring the lifecycle of normative states in a dynamic environment. This is a challenging problem that will have to rely on extending the (temporal) OBDA approach towards a setting of streaming data, a currently active area of research [15].


## Acknowledgement

This research has been supported by the unibz CRC Projects REKAP and PWORM.


## References


[1] M. Singh, "Norms as a basis for governing sociotechnical systems," *ACM Trans. on Intelligent Systems Technology*, vol. 5, no. 1, 2014.

[2] R. Ferrario and N. Guarino, "Commitment-based modeling of service systems," in *Proc. of the 3rd Int. Conf. on Exploring Services Science (IESS)*. Springer, 2012, pp. 170–185.

[3] M. Montali, D. Calvanese, and G. De Giacomo, "Verification of data-aware commitment-based multiagent systems," in *Proc. of the 13th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2014, pp. 157–164.

[4] F. Chesani, P. Mello, M. Montali, and P. Torroni, "Representing and monitoring social commitments using the event calculus," *J. of Autonomous Agents and Multi-Agent Systems*, vol. 27, no. 1, pp. 85–130, 2013.

[5] A. K. Chopra and M. P. Singh, "Cupid: Commitments in relational algebra," in *Proc. of the 29th AAAI Conf. on Artificial Intelligence (AAAI)*. AAAI Press, 2015, pp. 2052–2059.

[6] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati, "Linking data to ontologies," in *J. on Data Semantics*. Springer, 2008, vol. 10, pp. 133–173.

[7] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati, and M. Zakharyaschev, "Ontology-based data access: A survey," in *Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI)*. Int. Joint Conf. on Artificial Intelligence Org., 2018, pp. 5511–5519.

[8] G. Xiao, L. Ding, B. Cogrel, and D. Calvanese, "Virtual Knowledge Graphs: An overview of systems and use cases," *Data Intelligence*, vol. 1, no. 3, pp. 201–223, 2019.

[9] A. Chopra and M. Singh, "Custard: Computing norm states over information stores," in *Proc. of the 15th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2016, pp. 1096–1105.

[10] E. Güzel Kalayci, S. Brandt, D. Calvanese, V. Ryzhikov, G. Xiao, and M. Zakharyaschev, "Ontology-based access to temporal data with Ontop: A framework proposal," *Applied Mathematics and Computer Science*, vol. 29, no. 1, pp. 17–30, 2019.

[11] M. P. Singh, "Formalizing communication protocols for multiagent systems," in *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2007, pp. 1519–1524.

[12] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[13] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati, "Ontologies and databases: The *DL-Lite* approach," in *Reasoning Web: Semantic Technologies for Informations Systems – 5th Int. Summer School Tutorial Lectures (RW)*, ser. Lecture Notes in Computer Science, vol. 5689. Springer, 2009, pp. 255–356.

[14] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, and G. Xiao, "Ontop: Answering SPARQL queries over relational databases," *Semantic Web J.*, vol. 8, no. 3, pp. 471–487, 2017.

[15] R. Tommasini, P. Bonte, E. Della Valle, F. Ongenae, and F. De Turck, "A query model for ontology-based event processing over RDF streams," in *Proc. of the 21st Int. Conf. on Knowledge Engineering and Knowledge Management (EKAW)*, ser. Lecture Notes in Computer Science, vol. 11313. Springer, 2018, pp. 439–453.