Semantically-Governed Data-Aware Processes

Diego Calvanese[1], Giuseppe De Giacomo[2], Domenico Lembo[2],
Marco Montali[1], and Ario Santoso[1]

[1] Free University of Bozen-Bolzano, *lastname*@inf.unibz.it
[2] Sapienza Università di Roma, *lastname*@dis.uniroma1.it

Abstract. In this paper we consider processes that run over data stored in a relational database. Our setting is that of ontology-based data access (OBDA), where the information in the database is conceptually represented as an ontology and is declaratively mapped to it through queries. We are interested in verifying temporal logic formulas on the evolution of the information at the conceptual level, taking into account the knowledge present in the ontology, which allows for deducing information that is only implicitly available. Specifically, we show how, building on first-order rewritability of queries over the system state that is typical of ontology languages for OBDA, we are able to reformulate the temporal properties into temporal properties expressed over the underlying database. This allows us adopt notable decidability results on verification of evolving databases that have been established recently.

1 Introduction

Recent work in business processes, services and databases brought the necessity of considering both data and processes simultaneously while designing the system. This holistic view of considering data and processes together has given rise to a line of research under the name of *artifact-centric business processes* [16, 14, 19, 1] that aims at avoiding the notorious discrepancy of traditional approaches where these aspects are considered separately [7]. Recently, interesting decidability results for verification of temporal properties over such systems have been obtained in the context of so-called Data-centric Dynamic Systems (DCDSs) based on relational technology [12, 6, 4, 5]. In a DCDS, processes operate over the data of the system and evolve it by executing actions that may issue calls to external services. The data returned by such external services is injected into the system, effectively making it infinite state. There has been also some work on a form of DCDS based on ontologies, where the data layer is represented in a rich ontology formalism, and actions perform a form of instance level update of the ontology [3]. The use of an ontology allows for a high-level conceptual view of the data layer that is better suited for a business level treatment of the manipulated information.

Here we introduce Semantically-Governed Data-Aware Processes (SGDAP), in which we merge these two approaches by enhancing a *relational layer* constituted by a DCDS based system, with an ontology, constituting a *semantic layer*. The ontology captures the domain in which the SGDAP is executed, and allows for seeing the data and their manipulation at a conceptual level through an ontology-based data access (OBDA) system [8, 18]. Hence it provides us with a way of semantically governing

the underlying DCDS. Specifically, an SGDAP is constituted by two main components: *(i)* an *OBDA system* [8] which includes (the intensional level of) an ontology, a relational database schema, and a mapping between the ontology and the database; *(ii)* a *process component*, which characterizes the evolution of the system in terms of a process specifying preconditions and effects of action execution over the relational layer.

The ontology is represented through a Description Logic (DL) TBox [2], expressed in a lightweight ontology language of the *DL-Lite* family [10], a family of DLs specifically designed for efficiently accessing to large amounts of data. The mapping is defined in terms of a set of assertions, each relating an arbitrary (SQL) query over the relational layer to a set of atoms whose predicates are the concepts and roles of the ontology, and whose arguments are terms built using specific function symbols applied to the answer variables of the SQL query. Such mappings specify how to populate the elements of the ontology from the data in the database, and function symbols are used to construct (abstract) objects (*object terms*) from the concrete values retrieved from the database.

When an SGDAP evolves, each snapshot of the system is characterized by a database instance at the relational layer, and by a corresponding virtual ABox, which together with the TBox provides a conceptual view of the relational instance at the semantic layer. When the system is progressed by the process component, we assume that at every time the current instance can be arbitrarily queried, and can be updated through action executions, possibly involving external service calls to get new values from the environment. Hence the process component relies on three main notions: *actions*, which are the atomic progression steps for the data layer; *external services*, which can be called during the execution of actions; and a *process*, which is essentially a non-deterministic program that uses actions as atomic instructions. During the execution, the snapshots of the relational layer can be virtually mapped as ABoxes in the semantic layer. This enables to: *(i) understand* the evolution of the system at the conceptual level, and *(ii) govern* it at the semantic level, rejecting those actions that, executed at the relational layer, would lead to a new semantic snapshot that is inconsistent with the semantic layer's TBox.

In this work, we are interested in verifying dynamic properties specified in a variant of μ-calculus [15], one of the most powerful temporal logics, expressed over the semantic layer of an SGDAP. We consider properties expressed as μ-calculus formulae whose atoms are queries built over the semantic layer. By relying on techniques for query answering in *DL-Lite* OBDA systems, which exploit FOL rewritability of query answering and of ontology satisfiability, we reformulate the temporal properties expressed over the semantic layer into analogous properties over the relational layer. Given that our systems are in general infinite-state, verification of temporal properties is undecidable. However, we show how we can adapt to our setting recent results on the decidability of verification of DCDSs based on suitable finite-state abstractions [5].

2 Preliminaries

In this section we introduce the description logic (DL) *DL-Lite*$_{\mathcal{A},id}$ and describe the ontology-based data access (OBDA) framework.

DL-Lite$_{\mathcal{A},id}$ [11, 8] allows for specifying *concepts*, representing sets of objects, *roles*, representing binary relations between objects, and *attributes*, representing binary relations between objects and values. The syntax of concept, role and attribute *expressions*

in *DL-Lite*$_{\mathcal{A},id}$ is as follows:

$$B \longrightarrow N \mid \exists R \mid \delta(U) \qquad\qquad R \longrightarrow P \mid P^{-}$$

Here, N, P, and U respectively denote a *concept name*, a *role name*, and an *attribute name*, P^{-} denotes the *inverse of a role*, and B and R respectively denote *basic concepts* and *basic roles*. The concept $\exists R$, also called *unqualified existential restriction*, denotes the *domain* of a role R, i.e., the set of objects that R relates to some object. Similarly, the concept $\delta(U)$ denotes the *domain* of an attribute U, i.e., the set of objects that U relates to some value. Note that we consider here a simplified version of *DL-Lite*$_{\mathcal{A},id}$ where we distinguish between between objects and values, but do not further deal with different datatypes; similarly, we consider only a simplified version of identification assertions.

A *DL-Lite*$_{\mathcal{A},id}$ *ontology* is a pair $(\mathcal{T}, A)$, where $\mathcal{T}$ is a TBox, i.e., a finite set of *TBox assertions*, and A is an Abox, i.e., a finite set of *ABox assertions*. *DL-Lite*$_{\mathcal{A},id}$ TBox assertions have the following form:

$B_1 \sqsubseteq B_2$	$R_1 \sqsubseteq R_2$	$U_1 \sqsubseteq U_2$
$B_1 \sqsubseteq \neg B_2$	$R_1 \sqsubseteq \neg R_2$	$U_1 \sqsubseteq \neg U_2$
$(\mathsf{id}\ B\ Z_1, \ldots, Z_n)$	$(\mathsf{funct}\ R)$	$(\mathsf{funct}\ U)$

From left to right, assertions of the first row denote *inclusions* between basic concepts, basic roles, and attributes; assertions of the second row denote *disjointness* between basic concepts, basic roles, and attributes; assertions of the last row denote *identification (assertions)* (IdA), and global *functionality* on roles and attributes. In the IdA, each Z_i denotes either an attribute or a basic role. Intuitively, an IdA of the above form asserts that for any two different instances o, o' of B, there is at least one Z_i such that o and o' differ in the set of their Z_i-fillers, that is the set of objects (if Z_i is a role) or values (if Z_i is an attribute) that are related to o by Z_i. As usual, in *DL-Lite*$_{\mathcal{A},id}$ TBoxes we impose that roles and attributes occurring in functionality assertions or IdAs cannot be specialized (i.e., they cannot occur in the right-hand side of inclusions).

DL-Lite$_{\mathcal{A},id}$ ABox assertions have the form $N(t_1)$, $P(t_1, t_2)$, or $U(t_1, v_1)$, where t_1 and t_2 denote individual objects and v_1 denotes a value.

The semantics of *DL-Lite*$_{\mathcal{A},id}$ is given in [11]. We only recall here that we interpret objects and values over distinct domains, and that for both we adopt the Unique Name Assumption, i.e., different constants denote different objects (or values). The notions of *entailment*, *satisfaction*, and *model* are as usual [11]. We also say that A is *consistent wrt* $\mathcal{T}$ if $(\mathcal{T}, A)$ is satisfiable, i.e., admits at least one model.

Next we introduce queries. As usual (cf. OWL 2), answers to queries are formed by terms denoting individuals appearing in the ABox. The *domain of an ABox* A, denoted by $\textsc{adom}(A)$, is the (finite) set of terms appearing in A. A *union of conjunctive queries* (UCQ) q over a TBox $\mathcal{T}$ is a FOL formula of the form $\exists \vec{y}_1 . conj_1(\vec{x}, \vec{y}_1) \vee \cdots \vee \exists \vec{y}_n . conj_n(\vec{x}, \vec{y}_n)$, with free variables $\vec{x}$ and existentially quantified variables $\vec{y}_1, \ldots, \vec{y}_n$. Each $conj_i(\vec{x}, \vec{y}_i)$ in q is a conjunction of atoms of the form $N(z)$, $P(z, z')$, $U(z, z')$ where N, P and U respectively denote a concept, role and attribute name of $\mathcal{T}$, and z, z' are constants in a set $\mathcal{C}$ or variables in $\vec{x}$ or $\vec{y}_i$, for some $i \in \{1, \ldots, n\}$. The *(certain) answers* to q over an ontology $(\mathcal{T}, A)$ is the set $ans(q, \mathcal{T}, A)$ of substitutions[3]

[3] As customary, we can view each substitution simply as a tuple of constants, assuming some ordering of the free variables of q.

σ of the free variables of q with constants in $\text{ADOM}(A)$ such that $q\sigma$ evaluates to true in every model of $(\mathcal{T}, A)$. If q has no free variables, then it is called *boolean*, and its certain answers are true or false. Computing $ans(q, \mathcal{T}, A)$ of a UCQ q over a *DL-Lite*$_{\mathcal{A},id}$ ontology $(\mathcal{T}, A)$ is in AC^0 in the size of A [11]. This is actually a consequence of the fact that *DL-Lite*$_{\mathcal{A},id}$ enjoys the *FOL rewritability* property, which in our setting says that for every UCQ q, $ans(q, \mathcal{T}, A)$ can be computed by evaluating the UCQ $\text{REW}(q, \mathcal{T})$ over A considered as a database. $\text{REW}(q, \mathcal{T})$ is the so-called perfect reformulation of q w.r.t. $\mathcal{T}$ [11]. We also recall that, in *DL-Lite*$_{\mathcal{A},id}$, ontology satisfiability is FOL rewritable. In other words, we can construct a boolean FOL query $\mathsf{q}_{\mathsf{unsat}}(\mathcal{T})$ that evaluates to true over an ABox A iff the ontology $(\mathcal{T}, A)$ is unsatisfiable.

In our framework, we consider an extension of UCQs, called ECQs, which are queries of the query language *EQL-Lite*(UCQ) [9]. Formally, an *ECQ* over a TBox $\mathcal{T}$ is a possibly open *domain independent* formula of the form:

$$Q \longrightarrow [q] \mid \neg Q \mid Q_1 \wedge Q_2 \mid \exists x.Q \mid x = y$$

where q is a UCQ over $\mathcal{T}$ and $[q]$ denotes that q is evaluated under the (minimal) knowledge operator (cf. [9]). To compute the certain answers $\text{ANS}(Q, \mathcal{T}, A)$ to an ECQ Q over an ontology $(\mathcal{T}, A)$, we can compute the certain answers over $(\mathcal{T}, A)$ of each UCQ embedded in Q, and evaluate the first-order part of Q over the relations obtained as the certain answers of the embedded UCQs. Hence, also computing $\text{ANS}(Q, \mathcal{T}, A)$ of an ECQ Q over a *DL-Lite*$_{\mathcal{A},id}$ ontology $(\mathcal{T}, A)$ is in AC^0 in the size of A [9].

Ontology-Based Data Access (OBDA). In an OBDA system, a relational database is connected to an ontology that represents the domain of interest by a mapping, which relates database values with values and (abstract) objects in the ontology (c.f. [8]). In particular, we make use of a countably infinite set $\mathcal{V}$ of values and a set Λ of function symbols, each with an associated arity. We also define the set $\mathcal{C}$ of constants as the union of $\mathcal{V}$ and the set $\{f(d_1, \ldots, d_n) \mid f \in \Lambda \text{ and } d_1, \ldots, d_n \in \mathcal{V}\}$ of *object terms*.

Formally, an OBDA system is a structure $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$, where: *(i)* $\mathcal{R} = \{R_1, \ldots, R_n\}$ is a database schema, constituted by a finite set of relation schemas; *(ii)* $\mathcal{T}$ is a *DL-Lite*$_{\mathcal{A},id}$ TBox; *(iii)* $\mathcal{M}$ is a set of mapping assertions, each of the form: $\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{y}, \vec{t})$, where: (a) $\vec{x}$ is a non-empty set of variables, (b) $\vec{y} \subseteq \vec{x}$, (c) $\vec{t}$ is a set of object terms of the form $f(\vec{z})$, with $f \in \Lambda$ and $\vec{z} \subseteq \vec{x}$, (d) $\Phi(\vec{x})$ is an arbitrary SQL query over $\mathcal{D}$, with $\vec{x}$ as output variables, and (e) $\Psi(\vec{y}, \vec{t})$ is a conjunctive query over $\mathcal{T}$ of arity $n > 0$ without non-distinguished variables, whose atoms are over the variables $\vec{y}$ and the object terms $\vec{t}$.

Example 1. As a running example, we consider a simple university information system that stores and manipulates data concerning students and their degree. In particular, we define an OBDA system $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ to capture the conceptual schema of such a domain, how data are concretely maintained in a relational database, and how the two information levels are linked through mappings. The conceptual schema is depicted in Figure 1, and formalized as the following *DL-Lite*$_{\mathcal{A},id}$ TBox $\mathcal{T}$:

$$\begin{array}{rlr} \mathsf{Bachelor} \sqsubseteq \mathsf{Student} & \delta(\mathsf{MNum}) \sqsubseteq \mathsf{Student} & (\mathsf{funct\ MNum}) \\ \mathsf{Master} \sqsubseteq \mathsf{Student} & \mathsf{Student} \sqsubseteq \delta(\mathsf{MNum}) & (\mathsf{id\ Student\ MNum}) \\ \mathsf{Graduated} \sqsubseteq \mathsf{Student} & & \end{array}$$

The conceptual schema states that Bachelor and Master are subclasses of Student, that some Students could be already Graduated, and that MNum (representing the matriculation number) is

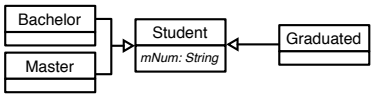

Fig. 1. UML conceptual schema for our running example.

an attribute relating individuals of type Student (domain of the attribute) to corresponding Codes (range of the attribute). The conceptual schema also expresses that each Student has *exactly one* matriculation number, and we assume that matriculation numbers can be used to *identify* Students (i.e., each MNum is associated to *at most one* Student). Data related to students are maintained in a concrete underlying data source that obeys the database schema $\mathcal{R}$, constituted by the following relation schemas: *(i)* ENROLLED(id, name, surname, type, endDate) stores information about students that are currently (endDate=NULL) or were enrolled in a bachelor (type="Bachelor") or master (type="Master") course. *(ii)* GRAD(id, mark, type) stores data of former students who have been graduated. *(iii)* TRANSF_M(name, surname) is a temporary relation used to maintain information about master students that have been recently transferred from another university, and must still complete the enrollment process. The interconnection between the database schema $\mathcal{R}$ and the conceptual schema $\mathcal{T}$ is specified through the following set $\mathcal{M}$ of mappings:

m_1 : SELECT name, surname, type FROM ENROLLED WHERE type ="Bachelor"
$\rightsquigarrow$ Bachelor(stu$_1$(name, surname, type))

m_2 : SELECT name, surname, type FROM ENROLLED WHERE type ="Master"
$\rightsquigarrow$ Master(stu$_1$(name, surname, type))

m_3 : SELECT name, surname, type, id FROM ENROLLED $\rightsquigarrow$ MNum(stu$_1$(name, surname, type), *val*(id))

m_4 : SELECT name, surname FROM TRANSF_M $\rightsquigarrow$ Master(stu$_1$(name, surname, "Master"))

m_5 : SELECT e.name, e.surname, e.type FROM ENROLLED e, GRAD g WHERE e.id = g.id
$\rightsquigarrow$ Graduated(stu$_1$(name, surname, type))

Intuitively, m_1 (m_2 resp.) maps every id in ENROLLED with type "Bachelor" ("Master") to a bachelor (master) student. Such a student is constructed by "objectifying" the name, surname and course type using variable term stu$_1/3$. In m_3, the MNum attribute is instead created using directly the value of id to fill in the target of the attribute. Notice the use of the *val* function symbol for mapping id to the range of MNum. Mapping m_4 leads to create further master students by starting from the temporary TRANSF_M table. Since such students are not explicitly associated to course type, but it is intended that they are "Master", objectification is applied to students' name and surname, adding "Master" as a constant in the variable term. Notice that, according to the TBox $\mathcal{T}$, such students have a matriculation number, but its value is not known (and, in fact, no mapping exists to generate their MNum attribute). Finally, m_5 generates graduated students by selecting only those students in the ENROLLED table whose matriculation number is also contained in the GRAD table. □

Given a database instance D made up of values in $\mathcal{V}$ and conforming to schema $\mathcal{R}$, and given a mapping $\mathcal{M}$, the *virtual ABox* generated from D by a mapping assertion $m = \Phi(x) \rightsquigarrow \Psi(y,t)$ in $\mathcal{M}$ is $m(D) = \bigcup_{v \in eval(\Phi,D)} \Psi[x/v]$, where $eval(\Phi, D)$ denotes the evaluation of the SQL query Φ over D, and where we consider $\Psi[x/v]$ to be a set of atoms (as opposed to a conjunction). Then, the ABox generated from D by the mapping $\mathcal{M}$ is $\mathcal{M}(D) = \bigcup_{m \in \mathcal{M}} m(D)$. Notice that $\text{ADOM}(\mathcal{M}(D)) \subseteq \mathcal{C}$. As for ABoxes, the active domain $\text{ADOM}(D)$ of a database instance D is the set of values occurring in D. Notice that $\text{ADOM}(D) \subseteq \mathcal{V}$. Given an OBDA system $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ and a database instance D for $\mathcal{R}$, a *model* for $\mathcal{O}$ wrt D is a model of the ontology $(\mathcal{T}, \mathcal{M}(D))$. We say that $\mathcal{O}$ wrt D is satisfiable if it admits a model wrt D.

Example 2. Consider a database instance $D = \{\mathsf{ENROLLED}(123, \mathsf{john}, \mathsf{doe}, \mathsf{Bachelor}, \mathsf{NULL})\}$. The corresponding virtual ABox obtained from the application of the mapping $\mathcal{M}$ is $\mathcal{M}(D) = \{\mathsf{Bachelor}(\mathsf{stu}_1(\mathsf{john}, \mathsf{doe}, \mathsf{Bachelor})), \mathsf{MNum}(\mathsf{stu}_1(\mathsf{john}, \mathsf{doe}, \mathsf{Bachelor}), \mathit{val}(123))\}$. □

An UCQ q over an OBDA system $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ is simply an UCQ over $\mathcal{T}$. To compute the certain answers of q over $\mathcal{O}$ wrt a database instance D for $\mathcal{R}$, we follow a three-step approach: *(i)* q is *rewritten* to compile away $\mathcal{T}$, obtaining $q_r = \text{REW}(q, \mathcal{T})$; *(ii)* the mapping $\mathcal{M}$ is used to *unfold* q_r into a query over $\mathcal{R}$, denoted by $\text{UNFOLD}(q_r, \mathcal{M})$, which turns out to be an SQL query [17]; *(iii)* such a query is executed over D, obtaining the certain answers. For an ECQ, we can proceed in a similar way, applying the rewriting and unfolding steps to the embedded UCQs. It follows that computing certain answers to UCQs/ECQs in an OBDA system is FOL rewritable. Applying the unfolding step to $\mathsf{q}_{\mathsf{unsat}}(\mathcal{T})$, we obtain also that satisfiability in $\mathcal{O}$ is FOL rewritable.

3 Semantically-Governed Data-Aware Processes

A Semantically-Governed Data-Aware Process (SGDAP) $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ is formed by an OBDA System $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ by a process component $\mathcal{P}$, and by an initial database instance D_0 that conforms to the relational schema $\mathcal{R}$ in $\mathcal{O}$. Intuitively, the OBDA system keeps all the data of interest, while the process component modifies and evolves such data, starting from the initial database D_0.

The process component $\mathcal{P}$ constitutes the progression mechanism for the SGDAP. Formally, $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \pi \rangle$, where: *(i)* $\mathcal{F}$ is a finite set of *functions* representing calls to *external services*, which return values; *(ii)* $\mathcal{A}$ is a finite set of *actions*, whose execution progresses the data layer, and may involve external service calls; *(iii)* π is a finite set of *condition-action rules* that form the specification of the overall *process*, which tells at any moment which actions can be executed.

An *action* $\alpha \in \mathcal{A}$ has the form $\alpha(p_1, \ldots, p_n) : \{e_1, \ldots, e_m\}$, where: *(i)* $\alpha(p_1, \ldots, p_n)$ is the *signature* of the action, constituted by a name α and a sequence $p_1, \ldots, p_n$ of *input parameters* that need to be substituted with values for the execution of the action, and *(ii)* $\{e_1, \ldots, e_m\}$ is a set of *effect specifications*, whose specified effects are assumed to take place simultaneously. Each e_i has the form $q_i^+ \wedge Q_i^- \rightsquigarrow E_i$, where: *(a)* $q_i^+ \wedge Q_i^-$ is a query over $\mathcal{R}$ whose terms are variables $\vec{x}$, action parameters, and constants from $\text{ADOM}(D_0)$. The query q_i^+ is a UCQ, and the query Q_i^- is an arbitrary FO formula whose free variables are included in those of q_i^+. Intuitively, q_i^+ selects the tuples to instantiate the effect, and Q_i^- filters away some of them. *(b)* E_i is the effect, i.e., a set of facts for $\mathcal{R}$, which includes as terms: terms in $\text{ADOM}(D_0)$, input parameters, free variables of q_i^+, and in addition Skolem terms formed by applying a function $f \in \mathcal{F}$ to one of the previous kinds of terms. Such Skolem terms involving functions represent external service calls and are interpreted so as to return a value chosen by an external user/environment when executing the action.

The *process* π is a finite set of *condition-action rules* $Q \mapsto \alpha$, where α is an action in $\mathcal{A}$ and Q is a FO query over $\mathcal{R}$ whose free variables are exactly the parameters of α, and whose other terms can be quantified variables or values in $\text{ADOM}(D_0)$.

Example 3. Consider the OBDA system $\mathcal{O}$ defined in Example 1. We now define a process component $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \pi \rangle$ over the relational schema $\mathcal{R}$ of $\mathcal{O}$, so as to obtain a full SGDAP.

In particular, π is constituted by the following condition-action rules ('_' denotes existentially quantified variables that are not used elsewhere):

- $\mathsf{ENROLLED}(\mathsf{id}, \_, \_, \_, \mathtt{NULL}) \rightsquigarrow \textsc{graduate}(\mathsf{id})$
- $\mathsf{TRANSF\_M}(\mathsf{name}, \mathsf{surname}) \rightsquigarrow \textsc{compl-enr}(\mathsf{name}, \mathsf{surname})$

The first rule extracts a matriculation number id of a currently enrolled student (endDate=NULL) from the ENROLLED relation and graduates the student, whereas the second rule selects a pair name surname in TRANSF_M and use them to complete the enrollment of that student. In order to be effectively executed, the involved actions rely on the following set $\mathcal{F}$ of service calls: *(i)* today() returns the current date; *(ii)* getMark(id, type) returns the final mark received by student id; *(iii)* getID(name, surname, type) returns the matriculation number for the name-surname pair of a student. The two actions GRADUATE and COMPL-ENR are then defined as follows:

$$
\begin{array}{l}
\textsc{graduate}(\mathsf{id}) : \{\ \mathsf{GRAD}(\mathsf{id}_2, \mathsf{m}, \mathsf{t}) \rightsquigarrow \mathsf{GRAD}(\mathsf{id}_2, \mathsf{m}, \mathsf{t}),\\
\qquad\qquad \mathsf{TRANSF\_M}(\mathsf{n}, \mathsf{s}) \rightsquigarrow \mathsf{TRANSF\_M}(\mathsf{n}, \mathsf{s}),\\
\qquad\qquad \mathsf{ENROLLED}(\mathsf{id}_2, \mathsf{n}, \mathsf{s}, \mathsf{t}, \mathsf{d}) \wedge \mathsf{id}_2 \neq \mathsf{id} \rightsquigarrow \mathsf{ENROLLED}(\mathsf{id}_2, \mathsf{n}, \mathsf{s}, \mathsf{t}, \mathsf{d}),\\
\qquad\qquad \mathsf{ENROLLED}(\mathsf{id}, \mathsf{n}, \mathsf{s}, \mathsf{t}, \mathtt{NULL}) \rightsquigarrow \mathsf{ENROLLED}(\mathsf{id}, \mathsf{n}, \mathsf{s}, \mathsf{t}, \mathsf{today}()),\\
\qquad\qquad \mathsf{ENROLLED}(\mathsf{id}, \_, \_, \mathsf{t}, \mathtt{NULL}) \rightsquigarrow \mathsf{GRAD}(\mathsf{id}, \mathsf{getMark}(\mathsf{id}, \mathsf{t}), \mathsf{t})\ \};\\
\textsc{compl-enr}(\mathsf{n}, \mathsf{s}) : \{\ \mathsf{GRAD}(\mathsf{id}, \mathsf{m}, \mathsf{t}) \rightsquigarrow \mathsf{GRAD}(\mathsf{id}, \mathsf{m}, \mathsf{t}),\\
\qquad\qquad \mathsf{ENROLLED}(\mathsf{id}, \mathsf{n}_2, \mathsf{s}_2, \mathsf{t}, \mathsf{d}) \rightsquigarrow \mathsf{ENROLLED}(\mathsf{id}, \mathsf{n}_2, \mathsf{s}_2, \mathsf{t}, \mathsf{d}),\\
\qquad\qquad \mathsf{TRANSF\_M}(\mathsf{n}_2, \mathsf{s}_2) \wedge (\mathsf{n}_2 \neq \mathsf{n} \vee \mathsf{s}_2 \neq \mathsf{s}) \rightsquigarrow \mathsf{TRANSF\_M}(\mathsf{n}_2, \mathsf{s}_2),\\
\qquad\qquad \mathsf{TRANSF\_M}(\mathsf{n}, \mathsf{s}) \rightsquigarrow \mathsf{ENROLLED}(\mathsf{getID}(\mathsf{n}, \mathsf{s}, \texttt{"Master"}), \mathsf{n}, \mathsf{s}, \texttt{"Master"}, \mathsf{NULL})\}
\end{array}
$$

Given a matriculation number id, action GRADUATE inserts a new tuple for id in GRAD, updating at the same time the enrollment's end date for id in ENROLLED to the current date, while keeping all other entries in TRANSF_M, GRAD and ENROLLED. Given a name and surname, action COMPL-ENR has the effect of moving the corresponding tuple in TRANSF_M to a new tuple in ENROLLED, for which the matriculation number is obtained by interacting with the getID service call; all other entries TRANSF_M, GRAD and ENROLLED are preserved. □

4 Semantics of SGDAP

This work focuses on the semantics of SGDAP assuming that *external services behave nondeterministically*, i.e., two calls of a service with the same arguments may return different results during the same run. This captures both services that model a truly nondeterministic process (e.g., human operators), and services that model stateful servers.

Let $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ be a SGDAP where $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ and $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \pi \rangle$. The semantics of $\mathcal{S}$ is defined in terms of a possibly infinite transition system (TS), which represents all possible computations that the process component can do over the data starting from D_0. We start by defining the semantics of *action execution*. Let α be an action in $\mathcal{A}$ of the form $\alpha(\vec{p}) : \{e_1, \ldots, e_n\}$ with effects $e_i = q_i^+ \wedge Q_i^- \rightsquigarrow E_i$, and let σ be a substitution of $\vec{p}$ with values in $\mathcal{V}$. The evaluation of the effects of α on a database instance D using a substitution σ is captured by the following function:

$$\textsc{do}(D, \alpha, \sigma) = \bigcup_{q_i^+ \wedge Q_i^- \rightsquigarrow E_i \text{ in } \alpha} \bigcup_{\theta \in \textsc{ans}((q_i^+ \wedge Q_i^-)\sigma, D)} E_i \sigma \theta$$

which returns a database instance made up of values in $\mathcal{V}$ and Skolem terms representing service calls. We denote with $\textsc{calls}(\textsc{do}(D, \alpha, \sigma))$ such service calls, and with $\textsc{evals}(D, \alpha, \sigma)$ the set of substitutions that replace these service calls with values in $\mathcal{V}$:

$$\textsc{evals}(D, \alpha, \sigma) = \{\theta \mid \theta : \textsc{calls}(\textsc{do}(D, \alpha, \sigma)) \to \mathcal{V} \text{ is a total function}\}.$$

We then say that the database instance D' over $\mathcal{V}$ and conforming to $\mathcal{R}$ is *produced* from D by the application of action α using substitution σ if $D' = \text{DO}(D, \alpha, \sigma)\theta$, where $\theta \in \text{EVALS}(D, \alpha, \sigma)$.

Relational Layer Transition System (RTS). Let $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ be a SGDAP with $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$. The RTS $\Upsilon^{\text{R}}_{\mathcal{S}}$ of $\mathcal{S}$ is formally defined as $\langle \mathcal{R}, \Sigma, s_0, db, \Rightarrow \rangle$, where Σ is a (possibly infinite) set of states, s_0 is the initial state, db is a total function from states in Σ to database instances made up of values in $\mathcal{V}$ and conforming to $\mathcal{R}$, and $\Rightarrow \subseteq \Sigma \times \Sigma$ is a transition relation. Σ, $\Rightarrow$ and db are defined by simultaneous induction as the smallest sets such that $s_0 \in \Sigma$, with $db(s_0) = D_0$, and satisfying the following property: Given $s \in \Sigma$, for each condition-action rule $Q(\vec{p}) \mapsto \alpha(\vec{p}) \in \pi$, for each substitution σ of $\vec{p}$ such that $\sigma \in \text{ANS}(Q, D)$, consider every database instance D' produced from D by the application of α using σ. Then: *(i)* if there exists $s' \in \Sigma$ such that $db(s') = D'$, then $s \Rightarrow s'$; *(ii)* otherwise, if $\mathcal{O}$ is *satisfiable* wrt D', then $s' \in \Sigma$, $s \Rightarrow s'$ and $db(s') = D'$, where s' is a fresh state. We observe that the satisfiability check done in the last step of the RTS construction accounts for *semantic governance*.

Semantic Layer Transition System (STS). Given a SGDAP $\mathcal{S}$ with $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ and with RTS $\Upsilon^{\text{R}}_{\mathcal{S}} = \langle \mathcal{R}, \Sigma, s_0, db, \Rightarrow \rangle$, the STS $\Upsilon^{\text{S}}_{\mathcal{S}}$ of $\mathcal{S}$ is a "virtualization" of the RTS in the semantic layer. In particular, $\Upsilon^{\text{S}}_{\mathcal{S}}$ maintains the structure of $\Upsilon^{\text{R}}_{\mathcal{S}}$ unaltered, reflecting that the process component is executed over the relational layer, but it associates each state to a virtual ABox obtained from the application of the mapping $\mathcal{M}$ to the database instance associated by $\Upsilon^{\text{R}}_{\mathcal{S}}$ to the same state. Formally, $\Upsilon^{\text{S}}_{\mathcal{S}} = \langle \mathcal{T}, \Sigma, s_0, \mathit{abox}, \Rightarrow \rangle$, where abox is a total function from Σ to ABoxes made up of individual objects in $\mathcal{C}$ and conforming to $\mathcal{T}$, such that for each $s \in \Sigma$ with $db(s) = D$, $\mathit{abox}(s) = \mathcal{M}(D)$.

5 Dynamic Constraints Formalism

Let $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ be an SGDAP where $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ and $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \pi \rangle$. We are interested in the verification of *conceptual temporal properties* over $\mathcal{S}$, i.e., properties that constrain the dynamics of $\mathcal{S}$ understood at the semantic layer. Technically, this means that properties are verified over the SGDAP's STS $\Upsilon^{\text{S}}_{\mathcal{S}}$, combining temporal operators with queries posed over the ontologies obtained by combining the TBox $\mathcal{T}$ with the ABoxes associated to the states of $\Upsilon^{\text{S}}_{\mathcal{S}}$. More specifically, we adopt ECQs [9] to query the ontologies of $\Upsilon^{\text{S}}_{\mathcal{S}}$, and μ-calculus [15] to predicate over the dynamics of $\Upsilon^{\text{S}}_{\mathcal{S}}$.

We use a variant of μ-calculus [15], one of the most powerful temporal logics subsuming LTL, PSL, and CTL* [13], called $\mu\mathcal{L}^{\text{EQL}}_{\text{C}}$, whose formulae have the form:

$$\Phi ::= Q \mid Z \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists x \in \mathcal{C}_0.\Phi \mid \langle - \rangle \Phi \mid \mu Z.\Phi$$

where Q is an ECQ over $\mathcal{T}$, $\mathcal{C}_0 = \text{ADOM}(\mathcal{M}(D_0))$ is the set of object terms appearing in the initial virtual ABox (obtained by applying the mapping $\mathcal{M}$ over the database instance D_0), and Z is a predicate variable. As usual, syntactic monotonicity is enforced to ensure existence of unique fixpoints. Beside the usual FOL abbreviations, we also make use of the following ones: $[-]\Phi = \neg\langle - \rangle(\neg\Phi)$ and $\nu Z.\Phi = \neg\mu Z.\neg\Phi[Z/\neg Z]$. The subscript C in $\mu\mathcal{L}^{\text{EQL}}_{\text{C}}$ stands for "closed", and attests that ECQs are closed queries. In fact, $\mu\mathcal{L}^{\text{EQL}}_{\text{C}}$ formulae only support the limited form of quantification $\exists x \in \mathcal{C}_0.\Phi$, which

is a convenient, compact notation for $\bigvee_{c \in \text{ADOM}(\mathcal{M}(D_0))} \Phi[x/c]$. We make this assumption for simplicity, but actually, with some care, our result can be extended to a more general form of quantification over time [5].

In order to define the semantics of $\mu\mathcal{L}_{\text{C}}^{\text{EQL}}$ we resort to transition systems. Let $\Upsilon = \langle \mathcal{T}, \Sigma, s_0, abox, \Rightarrow \rangle$ be an STS. Let V be a predicate and individual variable valuation on Υ, i.e., a mapping from the predicate variables Z to subsets of the states Σ, and from individual variables to constants in $\text{ADOM}(\mathcal{M}(D_0))$. Then, we assign meaning to $\mu\mathcal{L}_{\text{C}}^{\text{EQL}}$ formulas by associating to Υ and V an *extension function* $(\cdot)_V^{\mathfrak{A}}$, which maps $\mu\mathcal{L}_{\text{C}}^{\text{EQL}}$ formulas to subsets of Σ. The extension function $(\cdot)_V^{\mathfrak{A}}$ is defined inductively as:

$$
\begin{array}{ll}
(Q)_V^{\mathfrak{A}} & = \{s \in \Sigma \mid \text{ANS}(QV, \mathcal{T}, abox(s)) = \mathsf{true}\} \\
(Z)_V^{\mathfrak{A}} & = V(Z) \subseteq \Sigma \\
(\neg\Phi)_V^{\mathfrak{A}} & = \Sigma - (\Phi)_V^{\mathfrak{A}} \\
(\Phi_1 \wedge \Phi_2)_V^{\mathfrak{A}} & = (\Phi_1)_V^{\mathfrak{A}} \cap (\Phi_2)_V^{\mathfrak{A}} \\
(\exists x \in \mathcal{C}_0.\Phi)_V^{\mathfrak{A}} & = \bigcup\{(\Phi)_{V\,[x/c]}^{\mathfrak{A}} \mid c \in \text{ADOM}(\mathcal{M}(D_0))\} \\
(\langle - \rangle \Phi)_V^{\mathfrak{A}} & = \{s \in \Sigma \mid \exists s'.\ s \Rightarrow s' \text{ and } s' \in (\Phi)_V^{\mathfrak{A}}\} \\
(\mu Z.\Phi)_V^{\mathfrak{A}} & = \bigcap\{\mathcal{E} \subseteq \Sigma \mid (\Phi)_{v[Z/\mathcal{E}],V}^{\mathfrak{A}} \subseteq \mathcal{E}\}
\end{array}
$$

When Φ is a closed formula, $(\Phi)_V^{\mathfrak{A}}$ does not depend on V, and we denote it by $(\Phi)^{\mathfrak{A}}$. We are interested in the *model checking* problem, i.e., verify whether a $\mu\mathcal{L}_{\mathcal{C}}^{EQL}$ *closed formula* Φ *holds for the SGDAP* $\mathcal{S}$. This problem is defined as checking whether $s_0 \in (\Phi)^{\Upsilon_{\mathcal{S}}^{\text{S}}}$, that is, whether Φ is true in the initial state s_0 of $\Upsilon_{\mathcal{S}}^{\text{S}}$. If it is the case, we write $\Upsilon_{\mathcal{S}}^{\text{S}} \models \Phi$.

Example 4. An example of dynamic property in our running example is $\Phi = \mu Z.((\forall s.[\mathsf{Student(s)}] \rightarrow [\mathsf{Graduated(s)}]) \vee [-]Z)$, which says that every evolution of the system leads to a state in which all students present in that state are graduated. □

6 Verification of Dynamic Properties over SGDAPs

We now describe how $\mu\mathcal{L}_{\text{C}}^{\text{EQL}}$ properties can be effectively verified over SGDAPs. Let $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ be an SGDAP where $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ and $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \pi \rangle$. Let Φ be a $\mu\mathcal{L}_{\text{C}}^{\text{EQL}}$ dynamic property specified over the $\mathcal{T}$, and let $\Upsilon_{\mathcal{S}}^{\text{S}}$ and $\Upsilon_{\mathcal{S}}^{\text{R}}$ respectively be the STS and RTS of $\mathcal{S}$. The main issue to be tackled is that $\Upsilon_{\mathcal{S}}^{\text{S}}$ and $\Upsilon_{\mathcal{S}}^{\text{R}}$ are in general infinite-state, and their verification undecidable. In [5], some decidability boundaries for the verification of Data-Centric Dynamic Systems (DCDSs) have been extensively studied. DCDSs are tightly related to SGDAPs, with some key differences in the data component: *(i)* the process component is identical in the two frameworks; *(ii)* DCDSs are only equipped with a relational layer, i.e., no ontology nor mapping are specified; *(iii)* while SGDAPs define constraints over the data at the semantic layer, DCDSs are equipped with denial constraints posed directly over the database schema. Given a $\mu\mathcal{L}_{\text{C}}^{\text{EQL}}$ property Φ, we therefore attack the verification problem $\Upsilon_{\mathcal{S}}^{\text{S}} \models \Phi$ in the following way: (1) We transform Φ into a corresponding $\mu\mathcal{L}_C$ property Φ', i.e., a $\mu\mathcal{L}$ property whose atoms are closed FO queries over $\mathcal{R}$, thus reducing $\Upsilon_{\mathcal{S}}^{\text{S}} \models \Phi$ to $\Upsilon_{\mathcal{S}}^{\text{R}} \models \Phi'$. (2) We show, again exploiting FOL rewritability in *DL-Lite*$_{\mathcal{A}}$, that the consistency check used to generate $\Upsilon_{\mathcal{S}}^{\text{R}}$ can be rewritten as denial constraints over $\mathcal{R}$. This means that $\Upsilon_{\mathcal{S}}^{\text{R}}$ can be generated by a purely relational DCDS. (3) We argue that Φ' belongs to the dynamic

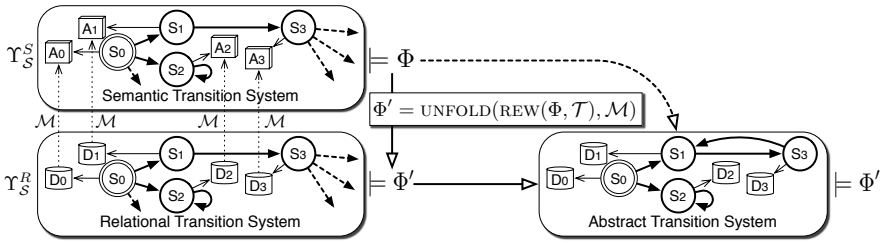

Fig. 2. Verification of dynamic $\mu\mathcal{L}_{\text{C}}^{\text{EQL}}$ properties over SGDAP

property language investigated in [5] for DCDSs under the nondeterministic semantics. (4) We can therefore reuse the decidability results of [5] to check whether $\Upsilon_{\mathcal{S}}^{\text{R}} \models \Phi'$ can be decided and, in the positive case, we apply the abstraction technique defined in [5] for reducing the verification problem to conventional finite-state model checking. Details are provided below. The idea of the approach is depicted in Figure 2.

Property Transformation. In order to transform the property, we separate the treatment of the dynamic part and of the embedded ECQs. Since the dynamics of an SGDAP is completely determined at the relational layer, the dynamic part is maintained unaltered. ECQs are instead manipulated as defined in Section 2. In particular, the rewriting of Φ wrt the TBox $\mathcal{T}$, denoted by $\Phi_r = \text{REW}(\Phi, \mathcal{T})$, is done by replacing each embedded ECQ with its corresponding rewriting wrt $\mathcal{T}$.

Example 5. Consider the $\mu\mathcal{L}_{\text{C}}^{\text{EQL}}$ property Φ described in Example 4, together with the TBox $\mathcal{T}$ introduced in Example 1. The rewriting of Φ wrt $\mathcal{T}$ produces $\Phi_r = \text{REW}(\Phi, \mathcal{T})$, which is:

$$\mu Z.(\forall s.[\mathsf{Student}(\mathsf{s}) \vee \mathsf{Bachelor}(\mathsf{s}) \vee \mathsf{Master}(\mathsf{s}) \vee \mathsf{MNum}(\mathsf{s}, \_)] \rightarrow [\mathsf{Graduated}(s)]) \vee [-]Z$$

□

Before unfolding the rewritten dynamic property Φ_r we translate each subformula of the form $\exists x \in \mathcal{C}_0.\Psi$ into the equivalent form $\bigvee_{c \in \text{ADOM}(\mathcal{M}(D_0))} \Psi[x/c]$. This means that when such a form of quantification is used, the initial ABox must be *materialized* in order to compute the initial active domain of the semantic layer. We then extend the UNFOLD() function defined in Section 2 to unfold a $\mu\mathcal{L}_{\text{C}}^{\text{EQL}}$ dynamic property over the semantic layer into a corresponding property over the relational layer. As for the rewriting, the temporal structure is maintained unaltered, reflecting that the dynamics of SGDAPs is determined at the relational layer. For what concerns the ECQs embedded in the property, the interesting case to be discussed is the one of (existential) quantification:

$$\begin{aligned}\text{UNFOLD}(\exists x.\varphi, \mathcal{M}) = \exists x.\text{UNFOLD}(\varphi, \mathcal{M}) \vee \\ \bigvee_{(f/n) \in \text{FS}(\mathcal{M})} \exists x_1, \ldots, x_n.\text{UNFOLD}(\varphi[x/f(x_1, \ldots, x_n)], \mathcal{M})\end{aligned}$$

where $\text{FS}(\mathcal{M})$ is the set of function symbols contained in $\mathcal{M}$. This unfolding reflects that quantification over individuals at the semantic layer must be properly rephrased as a corresponding quantification over those values in the relational layer that could lead to produce such individuals through the application of $\mathcal{M}$. This is done by unfolding $\exists x.\varphi$ into a disjunction of formulae, where: *(i)* the first formula corresponds to $\exists x.\varphi$ itself, and is used to tackle the case in which x appears in the range of an attribute, which is in fact a value; *(ii)* Each of the other formulae is obtained from φ by replacing x with one of the possible variable terms produced by $\mathcal{M}$, and quantifying over the existence of values used to construct the corresponding object term.

Example 6. Let us consider the $\mu\mathcal{L}_{\text{C}}^{\text{EQL}}$ property Φ_r of Example 5, together with the mapping $\mathcal{M}$ defined in Example 1. We get that $\text{UNFOLD}(\Phi_r, \mathcal{M})$ corresponds to:

$$\mu Z.\Big(\forall x_1, x_2, x_3.\text{AUX}_{m_3}(x_1, x_2, x_3, \_) \rightarrow \text{AUX}_{m_5}(x_1, x_2, x_3)\Big) \vee [-]Z$$

where $\text{AUX}_{m_3}(\mathsf{name}, \mathsf{surname}, \mathsf{type}, \mathsf{id})$ and $\text{AUX}_{m_5}(\mathsf{name}, \mathsf{surname}, \mathsf{type})$ represent the auxiliary view predicates of mapping assertions m_3 and m_5 respectively, whose defining queries are the SQL queries in the left-hand side of the mapping assertion themselves. When unfolding the UCQ $\mathsf{Student}(\mathsf{stu}_1(x_1, x_2, x_3)) \vee \mathsf{Bachelor}(\mathsf{stu}_1(x_1, x_2, x_3)) \vee \mathsf{Master}(\mathsf{stu}_1(x_1, x_2, x_3)) \vee \mathsf{MNum}(\mathsf{stu}_1(x_1, x_2, x_3), \_)$, we notice that the involved mapping assertions are m_1, m_2, and m_3. However, we only consider m_3, because the query on its left-hand side contains the ones on the left-hand side of m_1 and m_2.

Reduction to Data-Centric Dynamic Systems. The connection between SGDAPs and DCDSs is straightforward (see [5] for the definition of DCDS). Given a SGDAP $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ with $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$, we can construct a corresponding DCDS with nondeterministic services $\mathcal{S}_{\text{REL}} = \langle \mathcal{D}, \mathcal{P} \rangle$, where $\mathcal{D} = \langle \mathcal{V}, \mathcal{R}, \{\mathsf{q}_{\mathsf{unsat}}(\mathcal{T}) \rightarrow \mathsf{false}\}, D_0 \rangle$. Thanks to this encoding, we obtain $\Upsilon_{\mathcal{S}}^{\text{R}} \equiv \Upsilon_{\mathcal{S}_{\text{REL}}}^{\text{DCDS}}$, where $\Upsilon_{\mathcal{S}_{\text{REL}}}^{\text{DCDS}}$ is the RTS constructed for the DCDS $\mathcal{S}_{\text{REL}}$ following the definition in [5].

Verification. Leveraging on the parallel between SGDAPs and DCDSs, verification of a $\mu\mathcal{L}_{\text{C}}^{\text{EQL}}$ property over a SGDAP can be reduced to the verification of a $\mu\mathcal{L}_C$ property over the corresponding DCDS. In fact, $\mu\mathcal{L}_C$ (μ-calculus over closed FOL queries) is contained in the fragments of FO μ-calculus studied for DCDSs in [5], namely $\mu\mathcal{L}_A$ and $\mu\mathcal{L}_P$. Both $\mu\mathcal{L}_A$ and $\mu\mathcal{L}_P$ support FOL queries over the DCDS, allowing for controlled forms of FO quantification across states, and therefore they clearly support FO sentences.

Let $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ be a SGDAP with $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$, STS $\Upsilon_{\mathcal{S}}^{\text{S}}$ and $\Upsilon_{\mathcal{S}}^{\text{R}} = \langle \mathcal{R}, \Sigma, s_0, db, \Rightarrow \rangle$. We say that $\Upsilon_{\mathcal{S}}^{\text{R}}$ is *state-bounded* if there exists a bound b such that for each $s \in \Sigma$, $|\text{ADOM}(db(s))| < b$. Let Φ be a $\mu\mathcal{L}_{\text{C}}^{\text{EQL}}$ property, and let $\Phi' = \text{UNFOLD}(\text{REW}(\Phi, \mathcal{T}), \mathcal{M})$. Since *(i)* $\Upsilon_{\mathcal{S}}^{\text{S}} \models \Phi$ can be reduced to $\Upsilon_{\mathcal{S}}^{\text{R}} \models \Phi'$, *(ii)* Φ' belongs to $\mu\mathcal{L}_C$ (which is contained in $\mu\mathcal{L}_P$), *(iii)* $\Upsilon_{\mathcal{S}}^{\text{R}}$ can be generated by a DCDS with nondeterministic services, we can reuse the decidability results presented in [5]. In particular, we obtain that *$\Upsilon_{\mathcal{S}}^{\text{S}} \models \Phi$ is decidable if $\Upsilon_{\mathcal{S}}^{\text{R}}$ is state bounded*. Verification can in this case be reduced to conventional finite-state model checking.

Example 7. Consider the SGDAP $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$, where $\mathcal{O}$ is the OBDA system defined in Example 1, $\mathcal{P}$ the process component defined in Example 3. It is easy to see that the resulting RTS $\Upsilon_{\mathcal{S}}^{\text{R}}$ is state-bounded. Intuitively, this follows from the facts that the actions of $\mathcal{S}$ either move tuples from the TRANSF_M table to the ENROLLED one, or copy tuples from the ENROLLED table to the GRAD one. Hence, the size of each database instance appearing in $\Upsilon_{\mathcal{S}}^{\text{R}}$ is at most twice the size of D_0, thus verification of $\mu\mathcal{L}_{\text{C}}^{\text{EQL}}$ properties over the STS $\Upsilon_{\mathcal{S}}^{\text{S}}$ is decidable. □

Acknowledgements. This research has been partially supported by the ICT Collaborative Project ACSI (Artifact-Centric Service Interoperation), funded by the EU under FP7 ICT Call 5, 2009.1.2, grant agreement No. FP7-257593.

References

1. S. Abiteboul, P. Bourhis, A. Galland, and B. Marinoiu. The AXML artifact model. In *Proc. of TIME 2009*, pages 11–17, 2009.
2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
3. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, and R. De Masellis. Verification of conjunctive-query based semantic artifacts. In *Proc. of DL 2011*, volume 745 of *CEUR,* `ceur-ws.org`, 2011.
4. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, R. De Masellis, and P. Felli. Foundations of relational artifacts verification. In *Proc. of BPM 2011*, volume 6896 of *LNCS*, pages 379–395. Springer, 2011.
5. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of relational data-centric dynamic systems with external services. CoRR Technical Report arXiv:1203.0024, arXiv.org e-Print archive, 2012. Available at `http://arxiv.org/abs/1203.0024`.
6. F. Belardinelli, A. Lomuscio, and F. Patrizi. Verification of deployed artifact systems via data abstraction. In *Proc. of ICSOC 2011*, 2011.
7. K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *Proc. of BPM 2007*, volume 4714 of *LNCS*, pages 288–234. Springer, 2007.
8. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodríguez-Muro, and R. Rosati. Ontologies and databases: The *DL-Lite* approach. In S. Tessaris and E. Franconi, editors, *Semantic Technologies for Informations Systems – 5th Int. Reasoning Web Summer School (RW 2009)*, volume 5689 of *LNCS*, pages 255–356. Springer, 2009.
9. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. EQL-Lite: Effective first-order query processing in description logics. In *Proc. of IJCAI 2007*, 2007.
10. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.
11. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Path-based identification constraints in description logics. In *Proc. of KR 2008*, pages 231–241, 2008.
12. P. Cangialosi, G. De Giacomo, R. De Masellis, and R. Rosati. Conjunctive artifact-centric services. In *Proc. of ICSOC 2010*, volume 6470 of *LNCS*, pages 318–333. Springer, 2010.
13. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. The MIT Press, Cambridge, MA, USA, 1999.
14. D. Cohn and R. Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Bull. on Data Engineering*, 32(3):3–9, 2009.
15. E. A. Emerson. Automated temporal reasoning about reactive systems. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *LNCS*, pages 41–101. Springer, 1996.
16. A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.
17. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.
18. M. Rodríguez-Muro and D. Calvanese. Dependencies: Making ontology based data access work in practice. In *Proc. of AMW 2011*, volume 749 of *CEUR,* `ceur-ws.org`, 2011.
19. W. M. P. van der Aalst, P. Barthelmess, C. A. Ellis, and J. Wainer. Proclets: A framework for lightweight interacting workflow processes. *Int. J. of Cooperative Information Systems*, 10(4):443–481, 2001.