Ontology-Based Governance of 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 show how one can use the technology developed recently for Ontology-Based Data Access (OBDA) to govern data-aware processes through ontologies. In particular, we consider processes executed over a relational database which issue calls to external services to acquire new information and update the data. We equip these processes with an OBDA system, in which an ontology modeling the domain of interest is connected through declarative mappings to the database, and that consequently allows one to understand and govern the manipulated information at the conceptual level. In this setting, we are interested in verifying first-order μ-calculus formulae specifying temporal properties over the evolution of the information at the conceptual level. Specifically, we show how, building on first-order rewritability of queries over the system state that is typical of OBDA, we are able to reformulate the temporal properties into temporal properties expressed over the underlying database. This allows us to 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* [19,16,1,2] that aims at avoiding the notorious discrepancy of traditional approaches where these aspects are considered separately [9]. 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 [14,7,5,6]. 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 [4]. 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.

[⋆] This research has been partially supported by the EU under the ICT Collaborative Project ACSI (Artifact-Centric Service Interoperation), grant agreement n. FP7-257593.

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 [10,21]. 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* [10] 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 [3], expressed in a lightweight ontology language of the *DL-Lite* family [12], 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. As an example, let us consider a fragment of a university information system in which data about students and their degree is stored and manipulated. An ontology records the fact that both bachelor and master students are students, and that some of the students are graduated. The actual data about students is maintained in a relational database containing, among others, a table storing for currently enrolled students their id, name, surname, type of degree, and for previously enrolled students also the graduation date. The mappings relating the database to the ontology specify that the concepts for bachelor and master students are populated using a simple query that extracts from the enrollment table name, surname, and degree type of each students stored therein, and uses this information to create student objects. This reflects the fact that the combination of these three properties is considered sufficient to identify a student in the modeled domain.

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. In our example, we might have an action to graduate a student with a given id, that extracts from the enrollment table the student, provided her graduation date is `NULL` (indicating that the student is not graduated yet), and after obtaining the graduation mark by calling an external service, inserts her in the table of graduated students. During the execution, the snapshots of the

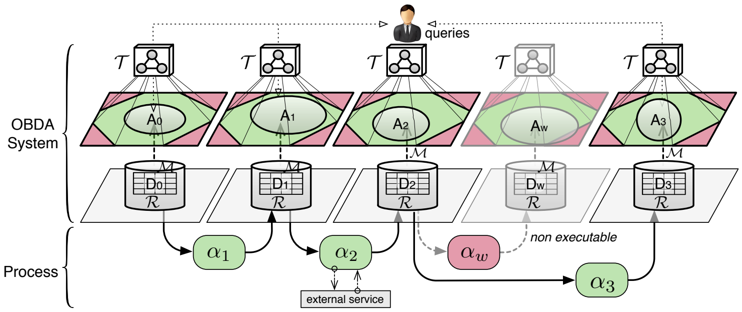

Fig. 1. Overview of a semantically-governed data-aware process

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. Figure 1 gives an intuition about the components of a SGDAP and the usages of the ontology to understand and govern the system execution. The subsequent technical development details the various components of the depicted framework, and the role they play in the system.

In this work, we are in particular interested in verifying dynamic properties specified in a variant of μ-calculus [18], 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. In our running example, we can verify a property stating that every evolution of the system leads to a state in which all students present in that state have graduated. 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 [6].

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*$_{\mathcal{A},id}$** [13,10] allows for specifying *concepts*, representing sets of objects, *roles*, representing binary relations between objects, and *attributes*, representing binary relations between objects and values. For simplicity, in this paper we consider a unique domain for all values used in the system. 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:

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

From left to right, assertions of the first row respectively 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 $U(t_1, v)$, where t_1 and t_2 denote individual objects and v denotes a value.

The semantics of *DL-Lite*$_{\mathcal{A},id}$ is given in [13]. 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 [13]. 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] σ of the free variables of q with constants in $\textsc{adom}(A)$ 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

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

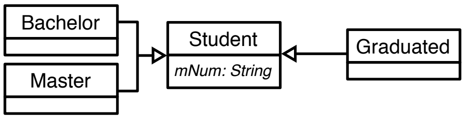

Fig. 2. UML conceptual schema for our running example

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 [13]. 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. $\mathcal{T}$ [13]. 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) [11]. 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. [11]). 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 [11].

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. [10]). 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 $\Phi(\vec{x}) \leadsto \Psi(\vec{y},\vec{t})$, 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 $\mathcal{R}$, with $\vec{x}$ as output variables, and (e) $\Psi(\vec{y},\vec{t})$ is a CQ over $\mathcal{T}$ of arity $n > 0$ without non-distinguished variables, whose atoms are over the variables $\vec{y}$ and the object terms $\vec{t}$. Without loss of generality, we use the special function symbol *val*/1 to map values from the relational layer to the range of attributes in the semantic layer.

Example 1. We formalize our running example presented in Section 1, and 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 TBox $\mathcal{T}$, shown in Figure 2 using the notation of UML class diagrams, is constituted by the following assertions:

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

The conceptual schema states that Bachelor, Master, and Graduated are subclasses of Student, and that MNum (representing the matriculation number) is an attribute of Student. The conceptual schema also expresses that: *(i)* each Student has *exactly one* matriculation number (by composing the assertion stating that each Student must be in the domain of MNum with the assertion stating that MNum is functional); *(ii)* 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)* $\mathsf{ENROLLED}(\mathsf{id}, \mathsf{name}, \mathsf{surname}, \mathsf{type}, \mathsf{endDate})$ stores information about students that are currently (endDate=NULL) or were enrolled in a bachelor (type="BSc") or master (type="MSc") course. *(ii)* $\mathsf{GRAD}(\mathsf{id}, \mathsf{mark}, \mathsf{type})$ stores data of former students who have been graduated. *(iii)* $\mathsf{TRANSF\_M}(\mathsf{name}, \mathsf{surname})$ 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 $\mathcal{R}$ and $\mathcal{T}$ is specified through the following set $\mathcal{M}$ of mapping assertions:

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

m_2 : SELECT name, surname, type FROM ENROLLED WHERE type ="MSc"
$\rightsquigarrow$ 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,"MSc"))

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 (resp., m_2) maps every id in ENROLLED with type "BSc" ("MSc") to a bachelor (master) student. Such a student is constructed by "objectifying" the name, surname and course type using variable term $\mathsf{stu}_1/3$. 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 "MSc", objectification is applied to students' name and surname, adding "MSc" 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 $\textsc{adom}(\mathcal{M}(D)) \subseteq \mathcal{C}$. 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 DB instance $D = \{\mathsf{ENROLLED}(123, \mathsf{john}, \mathsf{doe}, \texttt{"BSc"}, \texttt{NULL})\}$. The corresponding virtual ABox obtained from the application of the mapping $\mathcal{M}$ is $\mathcal{M}(D) = \{\mathsf{Bachelor}(\mathsf{stu_1}(\mathsf{john}, \mathsf{doe}, \texttt{"BSc"})), \mathsf{MNum}(\mathsf{stu_1}(\mathsf{john}, \mathsf{doe}, \texttt{"BSc"}), \mathit{val}(123))\}$. □

A 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 [20]; *(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 $\mathsf{q_{unsat}}(\mathcal{T})$, 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 $q_i^+ \land Q_i^- \rightsquigarrow E_i$, where: *(a)* $q_i^+ \land Q_i^-$ 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 FOL 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[4]. *(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

[4] To convey this intuition, we use the "$^-$" superscript.

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 FOL query over $\mathcal{R}$ whose free variables are exactly the parameters of α, and whose other terms can be quantified variables or values in $\textsc{adom}(D_0)$.

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):

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

The first rule extracts a matriculation number id of a currently enrolled student ($\mathsf{endDate}$=$\mathtt{NULL}$) from the $\mathsf{ENROLLED}$ relation and graduates the student, whereas the second rule selects a pair name $\mathsf{surname}$ in $\mathsf{TRANSF\_M}$ 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)* $\mathsf{today}()$ returns the current date; *(ii)* $\mathsf{getMark}(\mathsf{id}, \mathsf{type})$ returns the final mark received by student id; *(iii)* $\mathsf{getID}(\mathsf{name}, \mathsf{surname}, \mathsf{type})$ returns the matriculation number for the name-surname pair of a student. The two actions $\textsc{graduate}$ and $\textsc{compl-enr}$ are then defined as follows:

$$
\begin{aligned}
\textsc{graduate}(\mathsf{id}) : \{\ & \mathsf{GRAD}(\mathsf{id}_2, \mathsf{m}, \mathsf{t}) \leadsto \mathsf{GRAD}(\mathsf{id}_2, \mathsf{m}, \mathsf{t}),\\
& \mathsf{TRANSF\_M}(\mathsf{n}, \mathsf{s}) \leadsto \mathsf{TRANSF\_M}(\mathsf{n}, \mathsf{s}),\\
& \mathsf{ENROLLED}(\mathsf{id}_2, \mathsf{n}, \mathsf{s}, \mathsf{t}, \mathsf{d}) \land \mathsf{id}_2 \neq \mathsf{id} \leadsto \mathsf{ENROLLED}(\mathsf{id}_2, \mathsf{n}, \mathsf{s}, \mathsf{t}, \mathsf{d}),\\
& \mathsf{ENROLLED}(\mathsf{id}, \mathsf{n}, \mathsf{s}, \mathsf{t}, \mathtt{NULL}) \leadsto \mathsf{ENROLLED}(\mathsf{id}, \mathsf{n}, \mathsf{s}, \mathsf{t}, \mathsf{today}()),\\
& \mathsf{ENROLLED}(\mathsf{id}, \_, \_, \mathsf{t}, \mathtt{NULL}) \leadsto \mathsf{GRAD}(\mathsf{id}, \mathsf{getMark}(\mathsf{id}, \mathsf{t}), \mathsf{t})\ \}\\
\textsc{compl-enr}(\mathsf{n}, \mathsf{s}) : \{\ & \mathsf{GRAD}(\mathsf{id}, \mathsf{m}, \mathsf{t}) \leadsto \mathsf{GRAD}(\mathsf{id}, \mathsf{m}, \mathsf{t}),\\
& \mathsf{ENROLLED}(\mathsf{id}, \mathsf{n}_2, \mathsf{s}_2, \mathsf{t}, \mathsf{d}) \leadsto \mathsf{ENROLLED}(\mathsf{id}, \mathsf{n}_2, \mathsf{s}_2, \mathsf{t}, \mathsf{d}),\\
& \mathsf{TRANSF\_M}(\mathsf{n}_2, \mathsf{s}_2) \land (\mathsf{n}_2 \neq \mathsf{n} \lor \mathsf{s}_2 \neq \mathsf{s}) \leadsto \mathsf{TRANSF\_M}(\mathsf{n}_2, \mathsf{s}_2),\\
& \mathsf{TRANSF\_M}(\mathsf{n}, \mathsf{s})\\
& \quad \leadsto \mathsf{ENROLLED}(\mathsf{getID}(\mathsf{n}, \mathsf{s}, \texttt{"MSc"}), \mathsf{n}, \mathsf{s}, \texttt{"MSc"}, \mathsf{NULL})\}
\end{aligned}
$$

Given a matriculation number id, action $\textsc{graduate}$ inserts a new tuple for id in GRAD, updating at the same time the enrollment's end date for id in $\mathsf{ENROLLED}$ to the current date, while keeping all other entries in $\mathsf{TRANSF\_M}$, GRAD and $\mathsf{ENROLLED}$. Given a name and surname, action $\textsc{compl-enr}$ has the effect of moving the corresponding tuple in $\mathsf{TRANSF\_M}$ to a new tuple in $\mathsf{ENROLLED}$, for which the matriculation number is obtained by interacting with the getID service call; all other entries $\mathsf{TRANSF\_M}$, GRAD and $\mathsf{ENROLLED}$ are preserved. □

4 Execution Semantics

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). More specifically, two possible transition systems can be constructed to describe the execution semantics of $\mathcal{S}$: *(i)* a *relational layer transition system* (RTS), representing all allowed computations that, starting from D_0, the process component can do over the data of the relational layer, according to the constraints imposed at the semantic layer (semantic governance); *(ii)* a *semantic layer transition system* (STS), representing the same computations at the semantic layer. To construct these transition systems, we first define 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' is *produced* from D by the application of action α using substitution σ if $D' = \textsc{do}(D, \alpha, \sigma)\theta$, where $\theta \in \textsc{evals}(D, \alpha, \sigma)$.

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 $\Upsilon_{\mathcal{S}}^{\mathsf{R}}$ 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 $\sigma \in \textsc{ans}(Q, D)$, 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 $\Upsilon_{\mathcal{S}}^{\mathsf{R}} = \langle \mathcal{R}, \Sigma, s_0, db, \Rightarrow \rangle$, the STS $\Upsilon_{\mathcal{S}}^{\mathsf{S}}$ of $\mathcal{S}$ is a "virtualization" of the RTS in the semantic layer. In particular, $\Upsilon_{\mathcal{S}}^{\mathsf{S}}$ maintains the structure of $\Upsilon_{\mathcal{S}}^{\mathsf{R}}$ 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 $\Upsilon_{\mathcal{S}}^{\mathsf{R}}$ to the same state. Formally, $\Upsilon_{\mathcal{S}}^{\mathsf{S}} = \langle \mathcal{T}, \Sigma, s_0, \textit{abox}, \Rightarrow \rangle$, 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 $db(s) = D$, $\textit{abox}(s) = \mathcal{M}(D)$.

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 $\Upsilon^{\mathbb{S}}_{\mathcal{S}}$, combining temporal operators with queries posed over the ontologies obtained by combining the TBox $\mathcal{T}$ with the ABoxes associated to the states of $\Upsilon^{\mathbb{S}}_{\mathcal{S}}$. More specifically, we adopt ECQs [11] to query the ontologies of $\Upsilon^{\mathbb{S}}_{\mathcal{S}}$, and μ-calculus [18] to predicate over the dynamics of $\Upsilon^{\mathbb{S}}_{\mathcal{S}}$.

We use a variant of μ-calculus [18], one of the most powerful temporal logics subsuming LTL, PSL, and CTL* [15], called $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$, whose formulae have the form:

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

where Q is an ECQ over $\mathcal{T}$, $\mathcal{C}_0 = \textsc{adom}(\mathcal{M}(D_0))$ 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 $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ stands for "closed", and attests that ECQs are closed queries. In fact, $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ formulae only support the limited form of quantification $\exists x \in \mathcal{C}_0.\Phi$, which is a convenient, compact notation for $\bigvee_{c \in \textsc{adom}(\mathcal{M}(D_0))} \Phi[x/c]$. 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 [6].

In order to define the semantics of $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ we resort to STSs. Let $\Upsilon = \langle \mathcal{T}, \Sigma, s_0, \mathit{abox}, \Rightarrow \rangle$ 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 $\textsc{adom}(\mathcal{M}(D_0))$. Then, we assign meaning to $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ formulas by associating to Υ and V an *extension function* $(\cdot)^{\Upsilon}_{V}$, which maps $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ formulas to subsets of Σ. The extension function $(\cdot)^{\Upsilon}_{V}$ is defined inductively as:

$$\begin{array}{ll}
(Q)^{\Upsilon}_{V} & = \{s \in \Sigma \mid \textsc{ans}(QV, \mathcal{T}, \mathit{abox}(s)) = \mathsf{true}\} \\
(Z)^{\Upsilon}_{V} & = V(Z) \subseteq \Sigma \\
(\neg\Phi)^{\Upsilon}_{V} & = \Sigma - (\Phi)^{\Upsilon}_{V} \\
(\Phi_1 \vee \Phi_2)^{\Upsilon}_{V} & = (\Phi_1)^{\Upsilon}_{V} \cup (\Phi_2)^{\Upsilon}_{V} \\
(\exists x \in \mathcal{C}_0.\Phi)^{\Upsilon}_{V} & = \bigcup\{(\Phi)^{\Upsilon}_{V[x/c]} \mid c \in \textsc{adom}(\mathcal{M}(D_0))\} \\
(\langle - \rangle\Phi)^{\Upsilon}_{V} & = \{s \in \Sigma \mid \exists s'.\ s \Rightarrow s' \text{ and } s' \in (\Phi)^{\Upsilon}_{V}\} \\
(\mu Z.\Phi)^{\Upsilon}_{V} & = \bigcap\{\mathcal{E} \subseteq \Sigma \mid (\Phi)^{\Upsilon}_{V[Z/\mathcal{E}]} \subseteq \mathcal{E}\}
\end{array}$$

Intuitively, the extension function $(\cdot)^{\Upsilon}_{V}$ assigns to the various $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ constructs the following meanings. The boolean connectives have the expected meaning, while quantification is restricted to constants in $\textsc{adom}(\mathcal{M}(D_0))$. The extension of $\langle - \rangle\Phi$ consists of the states s such that for *some* state s' with $s \Rightarrow s'$, we have that Φ holds in s', The extension of $\mu Z.\Phi$ is the *smallest subset* $\mathcal{E}_\mu$ of Σ such that, assigning to Z the extension $\mathcal{E}_\mu$, the resulting extension of Φ is contained in $\mathcal{E}_\mu$. When Φ is a closed formula, $(\Phi)^{\Upsilon}_{V}$ does not depend on V, and we denote it by $(\Phi)^{\Upsilon}$. We are interested in the *model checking* problem, i.e., verify whether a $\mu\mathcal{L}_{C}^{EQL}$ *closed formula* Φ *holds for the SGDAP* $\mathcal{S}$. This problem is defined as checking whether $s_0 \in (\Phi)^{\Upsilon^{\mathbb{S}}_{\mathcal{S}}}$, that is, whether Φ is true in the initial state s_0 of $\Upsilon^{\mathbb{S}}_{\mathcal{S}}$. If it is the case, we write $\Upsilon^{\mathbb{S}}_{\mathcal{S}} \models \Phi$.

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 expresses that every evolution of the system leads to a state in which all students present in that state are graduated. □

6 Rewriting μ-Calculus Formulae

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 $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \pi \rangle$. In this section, we show how verification of $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ properties over the STS $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$ can be reduced to verification of $\mu\mathcal{L}_{C}$ properties over the corresponding RTS $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$.

$\mu\mathcal{L}_{C}$ properties are μ-calculus properties whose atoms are closed, domain-independent FO queries over a database schema. More specifically, the semantics of $\mu\mathcal{L}_{C}$ is defined over an RTS $\Upsilon^{\mathrm{R}} = \langle \mathcal{R}, \Sigma, s_0, db, \Rightarrow \rangle$, following exactly the same line given in Section 5 for $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ and STSs, except for local queries, whose semantics is:

$$(Q)_V^{\Upsilon^{\mathrm{R}}} = \{s \in \Sigma \mid eval(QV, db(s)) = \mathsf{true}\}$$

where $eval(QV, db(s)) = \mathsf{true}$ iff QV is true in $db(s)$, considered as a FOL interpretation. Let Φ be a $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ dynamic property specified over the TBox $\mathcal{T}$ of $\mathcal{S}$. The reduction is realized by providing a translation mechanism from Φ into a corresponding $\mu\mathcal{L}_{C}$ property Φ' specified over $\mathcal{R}$, and then showing that $\Upsilon_{\mathcal{S}}^{\mathrm{S}} \models \Phi$ if and only if $\Upsilon_{\mathcal{S}}^{\mathrm{R}} \models \Phi'$.

Before dealing with the translation of Φ, we substitute each subformula of the form $\exists x \in \mathcal{C}_0.\Psi$ into the equivalent form $\bigvee_{c \in \textsc{adom}(\mathcal{M}(D_0))} \Psi[x/c]$. This means that when such a form of quantification is used, the initial ABox must be *materialized* in order to compute the active domain of the initial ABox in the semantic layer. We then deal with the translation of Φ, by separating 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, performing in particular the following two-step translation: (1) the TBox $\mathcal{T}$ used by the property is compiled away, rewriting the original formula into a "self-contained" equivalent formula $\Phi_r = \textsc{rew}(\Phi, \mathcal{T})$, obtained by replacing each embedded ECQ with its corresponding rewriting wrt $\mathcal{T}$ [20]; (2) by using the information contained in the mapping $\mathcal{M}$, Φ_r is unfolded to the relational layer into $\textsc{unfold}(\Phi_r, \mathcal{M})$, by replacing each embedded ECQ with its corresponding unfolding wrt $\mathcal{M}$ [20].

As for the unfolding, the interesting case to be discussed is hence the one of (existential) quantification: the other cases are simply managed by pushing the unfolding down to the subformula(e). Given an ECQ of the form $\exists x.Q$, we have:

$$\textsc{unfold}(\exists x.Q, \mathcal{M}) = \bigvee_{(f/n) \in \textsc{fs}(\mathcal{M})} \exists x_1, \ldots, x_n.\textsc{unfold}(Q[x/f(x_1, \ldots, x_n)], \mathcal{M})$$

where $\textsc{fs}(\mathcal{M})$ is the set of function symbols contained in $\mathcal{M}$, including the special function symbol *val* used for attribute values. This unfolding reflects that quantification over object terms and values in the ontology must be properly rephrased as a corresponding quantification over those values in the relational layer that could lead to produce such object terms and values through the application of $\mathcal{M}$. This is done by unfolding $\exists x.Q$ into a disjunction of formulae, where each of the formulae is obtained from Q by replacing x with one of the possible variable terms constructed from function symbols in $\mathcal{M}$, and quantifying over the existence of values that could form a corresponding object term. We observe that one of the formulae, namely the one using the *val* function symbol, tackles the case in which x appears in the range of an attribute.

When the unfolding is applied to a UCQ of the query, the atoms in the UCQ are unified with the heads of the mapping assertions in $\mathcal{M}$. For each possible unifier, each atom is replaced with an auxiliary view predicate, which corresponds to a view defined in terms of the SQL query that constitutes the body of the matching mapping assertion. The UCQ's unfolding is then obtained as the union of all queries obtained in this way.

Example 5. Consider the $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ property Φ described in Example 4, together with the TBox $\mathcal{T}$ and mapping $\mathcal{M}$ in Example 1. $\Phi_r = \textsc{rew}(\Phi, \mathcal{T})$ results in $\mu Z.(\forall s.Q_r(s)) \vee [-]Z$, where

$$Q_r(s) = [\mathsf{Student}(s) \vee \mathsf{Bachelor}(s) \vee \mathsf{Master}(s) \vee \mathsf{MNum}(s, \_)] \supset [\mathsf{Graduated}(s)]$$

As for the unfolding, we first observe that $\textsc{fs}(\mathcal{M}) = \{\mathsf{stu}_1/3, \mathit{val}/1\}$. This means that $\textsc{unfold}(\forall s.Q_r(s), \mathcal{M})$ results in

$$\forall v.\textsc{unfold}(Q_r(\mathit{val}(v)), \mathcal{M}) \wedge \forall x_1, x_2, x_3.\textsc{unfold}(Q_r(\mathsf{stu}_1(x_1, x_2, x_3)), \mathcal{M})$$

The first conjunct corresponds to true, because there are no matching mapping assertions for the UCQ $\mathsf{Student}(\mathit{val}(x)) \vee \mathsf{Bachelor}(\mathit{val}(x)) \vee \mathsf{Master}(\mathit{val}(x)) \vee \mathsf{MNum}(\mathit{val}(x), \_)$, which is on the left-hand side of the implication in $Q_r(\mathit{val}(v))$. As for the second conjunct, 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, but 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. The unfolding then results in:

$$\mu Z.\big(\forall x_1, x_2, x_3.\textsc{aux}_{m_3}(x_1, x_2, x_3, \_) \supset \textsc{aux}_{m_5}(x_1, x_2, x_3)\big) \vee [-]Z$$

where m_3 and m_5 are the mapping assertions whose right-hand side respectively matches with $\mathsf{MNum}(\mathsf{stu}_1(x_1, x_2, x_3), \_)$ and $\mathsf{Graduated}(\mathsf{stu}_1(x_1, x_2, x_3))$, and where $\textsc{aux}_{m_3}(\mathsf{name}, \mathsf{surname}, \mathsf{type}, \mathsf{id})$ and $\textsc{aux}_{m_5}(\mathsf{name}, \mathsf{surname}, \mathsf{type})$ 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 assertions themselves. □

We are now ready to state our main result: verification of $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ properties over an STS can be faithfully reduced to verification of $\mu\mathcal{L}_C$ properties over the corresponding RTS.

Theorem 1. *Let $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ be an SGDAP with $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$, and let Φ be a $\mu\mathcal{L}_C^{EQL}$ dynamic property specified over $\mathcal{T}$. Then:*

$$\Upsilon_{\mathcal{S}}^{S} \models \Phi \quad \textit{if and only if} \quad \Upsilon_{\mathcal{S}}^{R} \models \textsc{unfold}(\textsc{rew}(\Phi, \mathcal{T}), \mathcal{M})$$

Proof. From Section 4, we know that $\Upsilon_{\mathcal{S}}^{\mathrm{R}} = \langle \mathcal{R}, \Sigma, s_0, \mathit{db}, \Rightarrow \rangle$ and $\Upsilon_{\mathcal{S}}^{\mathrm{S}} = \langle \mathcal{T}, \Sigma, s_0, \mathit{abox}, \Rightarrow \rangle$, where $\mathit{abox}(\cdot)$ is defined as follows: for every $s \in \Sigma$, $\mathit{abox}(s) = \mathcal{M}(\mathit{db}(s))$. We prove the following more general result: given a state $s \in \Sigma$, we have

$$s \in (\Phi)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}} \quad \text{if and only if} \quad s \in (\textsc{unfold}(\textsc{rew}(\Phi, \mathcal{T}), \mathcal{M}))^{\Upsilon_{\mathcal{S}}^{\mathrm{R}}}$$

For simplicity, below use $\textsc{ur}(\Phi)$ as an abbreviation for $\textsc{unfold}(\textsc{rew}(\Phi, \mathcal{T}), \mathcal{M})$.

We start by observing that we can drop first-order quantification from the language replacing $\exists x \in \mathcal{C}_0.\Phi$ with $\bigvee_{c \in \textsc{adom}(\mathit{abox}(s_0))} \Phi[x/c]$. This allows us to consider valuations only for the predicate variables, used in fixpoint formulae. The proof is then organized in three parts: (1) We prove the theorem for formulae of $\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$, obtained from $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ by dropping the predicate variables and the fixpoint constructs. $\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ corresponds to a first-order variant of the Hennessy Milner logic, and its semantics does not depend on the second-order valuation. (2) We extend the results to the infinitary logic obtained by extending $\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ with arbitrary countable disjunction. (3) We recall that fixpoints can be translated into this infinitary logic, thus proving that the theorem holds for $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$.

Proof for $\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$. We proceed by induction on the structure of Φ, without considering the case of predicate variable and of fixpoint constructs, which are not part of $\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$.

(Base case: $\Phi = Q$) We have to show that $\textsc{ans}(Q, \mathcal{T}, \mathit{abox}(s)) = \mathtt{true}$ if and only if $\textsc{ans}(\textsc{ur}(Q), \mathit{db}(s)) = \mathtt{true}$. By definition, $\mathit{abox}(s) = \mathcal{M}(\mathit{db}(s))$, hence the proof is obtained from the soundness and completeness of ECQ rewriting [11].

(Inductive step: $\Phi = \neg\Psi$) By induction hypothesis, for every $s \in \Sigma$ we have $s \in (\Psi)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$ if and only if $s \in (\textsc{ur}(\Psi))^{\Upsilon_{\mathcal{S}}^{\mathrm{R}}}$. Hence, $s \notin (\Psi)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$ if and only if $s \notin (\textsc{ur}(\Psi))^{\Upsilon_{\mathcal{S}}^{\mathrm{R}}}$, which in turn implies that $s \in (\neg\Psi)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$ if and only if $s \in (\neg\textsc{ur}(\Psi))^{\Upsilon_{\mathcal{S}}^{\mathrm{R}}}$. The proof is then obtained by observing that, by definition, $\neg\textsc{ur}(\Psi) = \textsc{ur}(\neg\Psi)$.

(Inductive step: $\Phi = \Phi_1 \vee \Phi_2$) By induction hypothesis, for every $s \in \Sigma$ we have $s \in (\Phi_1)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$ if and only if $s \in (\textsc{ur}(\Phi_1))^{\Upsilon_{\mathcal{S}}^{\mathrm{R}}}$, and $s \in (\Phi_2)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$ if and only if $s \in (\textsc{ur}(\Phi_2))^{\Upsilon_{\mathcal{S}}^{\mathrm{R}}}$. Hence, $s \in (\Phi_1)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$ or $s \in (\Phi_2)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$ if and only if $s \in (\textsc{ur}(\Phi_1))^{\Upsilon_{\mathcal{S}}^{\mathrm{R}}}$ or $s \in (\textsc{ur}(\Phi_2))^{\Upsilon_{\mathcal{S}}^{\mathrm{R}}}$, which in turn implies that $s \in (\Phi_1 \vee \Phi_2)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$ if and only if $s \in (\textsc{ur}(\Phi_1) \vee \textsc{ur}(\Phi_2))^{\Upsilon_{\mathcal{S}}^{\mathrm{R}}}$. The proof is then obtained by observing that, by definition, $\textsc{ur}(\Phi_1) \vee \textsc{ur}(\Phi_2) = \textsc{ur}(\Phi_1 \vee \Phi_2)$.

(Inductive step: $\Phi = \langle-\rangle\Psi$) By induction hypothesis, for every $s' \in \Sigma$ we have $s' \in (\Psi)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$ if and only if $s' \in (\textsc{ur}(\Psi))^{\Upsilon_{\mathcal{S}}^{\mathrm{R}}}$. Now consider that, by definition, $s \in (\langle-\rangle\Psi)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$ if and only if there exists a transition $s \Rightarrow s'$ such that $s' \in (\Psi)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$. By construction, $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$ and $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ have the same transition relation. Therefore, we have that $s \in (\langle-\rangle\Psi)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$ if and only if $s \in (\langle-\rangle\textsc{ur}(\Psi))^{\Upsilon_{\mathcal{S}}^{\mathrm{R}}}$. The proof is then obtained by observing that, by definition, $\langle-\rangle\textsc{ur}(\Psi) = \textsc{ur}(\langle-\rangle\Psi)$.

Extension to arbitrary countable disjunction. Let Ψ be a countable set of $\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ formulae. Given an STS $\Upsilon^{\mathrm{S}} = \langle\mathcal{T}, \Sigma, s_0, \mathit{abox}, \Rightarrow\rangle$, the semantics of $\bigvee\Psi$ is $(\bigvee\Psi)^{\Upsilon^{\mathrm{S}}} = \bigcup_{\psi\in\Psi}(\psi)^{\Upsilon^{\mathrm{S}}}$ (similarly for RTSs). Therefore, given a state $s \in \Sigma$ we have $s \in (\bigvee\Psi)^{\Upsilon^{\mathrm{S}}}$ if and only if there exists $\psi \in \Psi$ such that $s \in (\psi)^{\Upsilon^{\mathrm{S}}}$. Arbitrary countable conjunction is obtained for free because of negation.

Let $\Upsilon_{\mathcal{S}}^{\mathrm{R}} = \langle\mathcal{R}, \Sigma, s_0, \mathit{db}, \Rightarrow\rangle$ and $\Upsilon_{\mathcal{S}}^{\mathrm{S}} = \langle\mathcal{T}, \Sigma, s_0, \mathit{abox}, \Rightarrow\rangle$. By induction hypothesis, we can assume that for every $s \in \Sigma$ and formula $\psi \in \Psi$, we have $s \in (\psi)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$ if and only if $s \in (\textsc{ur}(\psi))^{\Upsilon_{\mathcal{S}}^{\mathrm{R}}}$. Given the semantics of $\bigvee\Psi$ above, this implies that $s \in (\bigvee\Psi)^{\Upsilon_{\mathcal{S}}^{\mathrm{S}}}$ if and only if $s \in (\bigvee\textsc{ur}(\Psi))^{\Upsilon_{\mathcal{S}}^{\mathrm{R}}}$, where $\textsc{ur}(\Psi) = \{\textsc{ur}(\psi) \mid \psi \in \Psi\}$. The proof is then obtained by observing that $\bigvee\textsc{ur}(\Psi) = \textsc{ur}(\bigvee\Psi)$.

Extension to full $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$. In order to extend the result to the whole $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$, we resort to the well-known result stating that fixpoints of the μ-calculus can be translated into the infinitary Hennessy Milner logic by iterating over *approximants*, where the approximant of index α is denoted by $\mu^{\alpha}Z.\Phi$ (resp. $\nu^{\alpha}Z.\Phi$). This is a standard result that also holds for $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$. In particular, approximants are built as follows:

$$\begin{array}{rlrl} \mu^0 Z.\Phi &= \mathsf{false} & \nu^0 Z.\Phi &= \mathsf{true} \\ \mu^{\beta+1} Z.\Phi &= \Phi[Z/\mu^\beta Z.\Phi] & \nu^{\beta+1} Z.\Phi &= \Phi[Z/\nu^\beta Z.\Phi] \\ \mu^\lambda Z.\Phi &= \bigvee_{\beta<\lambda} \mu^\beta Z.\Phi & \nu^\lambda Z.\Phi &= \bigwedge_{\beta<\lambda} \nu^\beta Z.\Phi \end{array}$$

where λ is a limit ordinal, and where fixpoints and their approximants are connected by the following properties: given an STS or RTS Υ and a state s of Υ

- $s \in (\mu Z.\Phi)_V^\Upsilon$ if and only if there exists an ordinal α such that $s \in (\mu^\alpha Z.\Phi)_V^\Upsilon$ and, for every $\beta < \alpha$, it holds that $s \notin (\mu^\beta Z.\Phi)_V^\Upsilon$;
- $s \notin (\nu Z.\Phi)_V^\Upsilon$ if and only if there exists an ordinal α such that $s \notin (\nu^\alpha Z.\Phi)_V^\Upsilon$ and, for every $\beta < \alpha$, it holds that $s \in (\nu^\beta Z.\Phi)_V^\Upsilon$. □

7 Decidability Results

Given an SGDAP $\mathcal{S}$, in Section 6 we have shown how to reduce verification of $\mu\mathcal{L}_{\mathrm{C}}^{\mathrm{EQL}}$ properties over $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$ into verification of $\mu\mathcal{L}_C$ properties over $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$. However, due to the injection of new, fresh data into the system due to call to external services, $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ (as well as $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$) is in general infinite-state. This causes verification to be undecidable in general, even for the very simple case of an SGDAP in which the TBox contains no assertions and directly reflects the database schema via simple one-to-one mappings, and where the temporal formula to be verified is a propositional reachability property [6].

An extensive study concerning some decidability boundaries for the verification of Data-Centric Dynamic Systems (DCDSs) with non-deterministic external services has been provided in [6]. One of the most interesting conditions for decidability that have been studied so far is *state-boundedness*. Let $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$ be an SGDAP with $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$ and RTS $\Upsilon_{\mathcal{S}}^{\mathrm{R}} = \langle \mathcal{R}, \Sigma, s_0, db, \Rightarrow \rangle$. We say that $\mathcal{S}$ is *state-bounded* if there exists a bound b such that for each $s \in \Sigma$, $|\textsc{adom}(db(s))| < b$. Intuitively, state-boundedness imposes that the database associated to the state of the RTS $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ remains bounded, although it may acquire arbitrarily many new values in the course of the evolution of the system (forgetting old ones, to keep the bound on the state). Leveraging on the result on state-boundedness, we can exploit our rewriting result above to get the following theorem.

Theorem 2. *Verification of $\mu\mathcal{L}_C^{EQL}$ properties over state-bounded SGDAPs is decidable, and can be reduced to conventional finite-state model checking.*

Proof (sketch). The proof is based on a reduction to DCDSs. For a formal definition of a DCDS, the interested reader can refer to [6]. Intuitively, 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 constituted by a relational layer (i.e., no ontology nor mapping are present); *(iii)* while SGDAPs define constraints over the data at the semantic layer, DCDSs are equipped with denial constraints posed directly over the database schema.

Let $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$, with $\mathcal{O} = \langle \mathcal{R}, \mathcal{T}, \mathcal{M} \rangle$. By exploiting FOL rewritability in *DL-Lite*$_{\mathcal{A}}$, the consistency check used to generate $\Upsilon_{\mathcal{S}}^{\mathrm{S}}$ can be rewritten as a denial constraint over $\mathcal{R}$. This means that $\Upsilon_{\mathcal{S}}^{\mathrm{R}}$ can be generated by a purely relational DCDS. Technically, starting from $\mathcal{S}$ we can construct a corresponding DCDS with nondeterministic services $\mathcal{S}_{\text{REL}} = \langle \mathcal{D}, \mathcal{P} \rangle$, where $\mathcal{D} = \langle \mathcal{V}, \mathcal{R}, \{\textsc{unfold}(\mathsf{q}_{\mathsf{unsat}}(\mathcal{T}), \mathcal{M}) \rightarrow \mathsf{false}\}, D_0 \rangle$, such that $\Upsilon^{\text{R}}_{\mathcal{S}} \equiv \Upsilon^{\text{DCDS}}_{\mathcal{S}_{\text{REL}}}$, where $\Upsilon^{\text{DCDS}}_{\mathcal{S}_{\text{REL}}}$ is the RTS constructed for the DCDS $\mathcal{S}_{\text{REL}}$ following the definition in [6]. This also means that $\mathcal{S}$ is state-bounded if and only if $\mathcal{S}_{\text{REL}}$ is state-bounded.

Let us now consider a $\mu\mathcal{L}^{\text{EQL}}_{\text{C}}$ property Φ. From Theorem 1, we know that $\Upsilon^{\text{S}}_{\mathcal{S}} \models \Phi$ if and only if $\Upsilon^{\text{R}}_{\mathcal{S}} \models \Phi'$, where $\Phi' = \textsc{unfold}(\textsc{rew}(\Phi, \mathcal{T}), \mathcal{M})$. By recalling that $\Upsilon^{\text{R}}_{\mathcal{S}} \equiv \Upsilon^{\text{DCDS}}_{\mathcal{S}_{\text{REL}}}$, we get that $\Upsilon^{\text{S}}_{\mathcal{S}} \models \Phi$ if and only if $\Upsilon^{\text{DCDS}}_{\mathcal{S}_{\text{REL}}} \models \Phi'$. The proof is then obtained from the decidability of verification of $\mu\mathcal{L}_P$ properties for state-bounded DCDSs with non-deterministic services [6], by recalling that Φ' is a $\mu\mathcal{L}_C$ property, and by observing that $\mu\mathcal{L}_C$ is trivially contained in $\mu\mathcal{L}_P$ [6]. □

Example 6. Consider the SGDAP $\mathcal{S} = \langle \mathcal{O}, \mathcal{P}, D_0 \rangle$, where $\mathcal{O}$ is the OBDA system defined in Example 1, and $\mathcal{P}$ the process component defined in Example 3. It is easy to see that the resulting RTS $\Upsilon^{\text{R}}_{\mathcal{S}}$ 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 $\Upsilon^{\text{R}}_{\mathcal{S}}$ is at most twice the size of D_0, thus verification of $\mu\mathcal{L}^{\text{EQL}}_{\text{C}}$ properties over the STS $\Upsilon^{\text{S}}_{\mathcal{S}}$ is decidable. □

We close this section by mentioning that the sufficient syntactic conditions for state-boundedness of DCDSs given in [6] can be easily applied to SGDAPs as well, given that the structure of the process component remains unchanged.

8 Conclusion

In this paper, we have introduced semantically-governed data-aware processes, where an ontology in *DL-Lite*$_{\mathcal{A},id}$ is used to capture the information manipulated by the process at the conceptual level, and to understand and govern the process itself. Our key result is the ability of extending FOL rewritability, typical of *DL-Lite*, to arbitrary temporal formulae expressed in μ-calculus. Indeed, in this paper we have used this result to show decidability of temporal verification in our setting for an interesting class of data-aware processes. The exact computational complexity of verification remains to be investigated.

However, the result appears to be much more general, and can be exploited to lift results obtained lately for (relational) data-aware processes [8,6,17], to the case in which an ontology-based governance component is introduced. We plan to investigate this further in the future, and study the requirements on the languages used to express the ontology and the mappings that make this lifting feasible. The framework described in this paper is being applied in the context of the EU project ACSI[5] to two significant real-world case studies.

References

1. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A framework for lightweight interacting workflow processes. Int. J. of Cooperative Information Systems 10(4), 443–481 (2001)

[5] http://www.acsi-project.eu

2. Abiteboul, S., Bourhis, P., Galland, A., Marinoiu, B.: The AXML artifact model. In: Proc. of TIME. pp. 11–17 (2009)
3. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press (2003)
4. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., De Masellis, R.: Verification of conjunctive-query based semantic artifacts. In: Proc. of DL. CEUR, `ceur-ws.org`, vol. 745 (2011)
5. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., De Masellis, R., Felli, P.: Foundations of relational artifacts verification. In: Proc. of BPM. LNCS, vol. 6896, pp. 379–395. Springer (2011)
6. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., Deutsch, A., Montali, M.: 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`
7. Belardinelli, F., Lomuscio, A., Patrizi, F.: Verification of deployed artifact systems via data abstraction. In: Proc. of ICSOC. LNCS, vol. 7084, pp. 142–156. Springer (2011)
8. Belardinelli, F., Lomuscio, A., Patrizi, F.: An abstraction technique for the verification of artifact-centric systems. In: Proc. of KR. pp. 319–328 (2012)
9. Bhattacharya, K., Gerede, C., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: Proc. of BPM. LNCS, vol. 4714, pp. 288–234. Springer (2007)
10. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodríguez-Muro, M., Rosati, R.: Ontologies and databases: The *DL-Lite* approach. In: Tessaris, S., Franconi, E. (eds.) Semantic Technologies for Informations Systems – 5th Int. Reasoning Web Summer School (RW 2009), LNCS, vol. 5689, pp. 255–356. Springer (2009)
11. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: EQL-Lite: Effective first-order query processing in description logics. In: Proc. of IJCAI. pp. 274–279 (2007)
12. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. J. of Automated Reasoning 39(3), 385–429 (2007)
13. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Path-based identification constraints in description logics. In: Proc. of KR. pp. 231–241 (2008)
14. Cangialosi, P., De Giacomo, G., De Masellis, R., Rosati, R.: Conjunctive artifact-centric services. In: Proc. of ICSOC. LNCS, vol. 6470, pp. 318–333. Springer (2010)
15. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. The MIT Press, Cambridge, MA, USA (1999)
16. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Bull. on Data Engineering 32(3), 3–9 (2009)
17. De Masellis, R., De Giacomo, G., Rosati, R.: Verification of conjunctive artifact-centric services. Int. J. of Cooperative Information Systems (2012), to appear
18. Emerson, E.A.: Automated temporal reasoning about reactive systems. In: Moller, F., Birtwistle, G. (eds.) Logics for Concurrency: Structure versus Automata, LNCS, vol. 1043, pp. 41–101. Springer (1996)
19. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
20. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. J. on Data Semantics X, 133–173 (2008)
21. Rodriguez-Muro, M., Calvanese, D.: High performance query answering over *DL-Lite* ontologies. In: Proc. of KR. pp. 308–318 (2012)