J Data Semant
DOI 10.1007/s13740-014-0036-6

ORIGINAL ARTICLE

Semantic Enrichment of GSM-Based Artifact-Centric Models

Riccardo De Masellis · Domenico Lembo · Marco Montali · Dmitry Solomakhin

Received: 14 April 2013 / Revised: 20 February 2014 / Accepted: 25 February 2014

Abstract We provide a comprehensive framework for *semantic GSM artifacts*, discuss in detail its properties, and present main software engineering architectures it is able to capture. The distinguishing aspect of our framework is that it allows for expressing both the data and the lifecycle schema of GSM artifacts in terms of an ontology, i.e., a shared and formalized conceptualization of the domain of interest. To guide the modeling of data and lifecycle we provide an upper ontology, which is specialized in each artifact with specific lifecycle elements, relations, and business objects. The framework thus obtained allows to achieve several advantages. On the one hand, it makes the specification of conditions on data and artifact status attribute fully declarative and enables semantic reasoning over them. On the other, it fosters the monitoring of artifacts and the interoperation and cooperation among different artifact systems. To fully achieve such an interoperation, we enrich our framework by enabling the linkage of the ontology to autonomous database systems through the use of mappings. We then discuss two scenarios of practical interest that show how mappings can be used in the presence of multiple systems. For one of these scenarios we also describe a concrete instantiation of the framework and its application to a real-world use case in the energy domain, investigated in the context of the EU project ACSI.

Keywords Data-aware process modeling · Artifact-centric processes · Guard Stage Milestone lifecycle · Ontologies · Ontology-based data access · Process monitoring

1 Introduction

Recent work in business processes, services, and databases brought the necessity of considering both data and processes simultaneously while designing enterprise systems. This holistic view of considering data and processes together has given rise to a line of research known under the name of *data-aware business processes* [2,4,44,48], aiming to avoid the notorious discrepancy of traditional activity-centric models where these two aspects are considered separately. One of the first proposals in this area is the *artifact-centric* approach (see, e.g., [24,48]), which relies on the notion of *artifact*, a business-relevant conceptual entity in a given domain which combines both static properties, describing the data of interest, and the dynamics, induced by processes that manipulate such data.

Even though the tight integration of data and processes is central for artifact-centric systems, the information managed by artifacts is typically captured by means of rather simple structures, such as lists of relevant attributes. A rich, conceptual, and well-founded modeling of the data component is yet to come. Furthermore, a suitable balance between data and processes is often missing, leading to artifacts which rely on data structures essentially tailored to the process they have to serve, instead of richer structures able to fully reflect the complexity of the domain of interest.

R. De Masellis · D. Lembo (✉)
Sapienza Università di Roma,
Via Ariosto 25, 00185 Rome, Italy
e-mail: lembo@dis.uniroma1.it

R. De Masellis
e-mail: demasellis@dis.uniroma1.it

M. Montali · D. Solomakhin
Free University of Bozen-Bolzano Piazza Domenicani
3, 39100 Bolzano, Italy
e-mail: montali@inf.unibz.it

D. Solomakhin
e-mail: solomakhin@inf.unibz.it

Published online: 11 April 2014

The main negative consequence is that it is difficult to exploit the artifact data that go beyond those of the specific process execution. This, in turn, makes difficult to(i) govern the entire enterprise system, (ii) interconnect the data manipulated by the different artifacts so as to construct a unique, high-level view of them, (iii) evolve the system so as to incorporate new features impacting on the data component, and (iv) support interoperation with new processes and external systems.

By leveraging on the recent, extensive work on ontology-based data access (OBDA) (see, e.g., [14,40,50]), our primary goal is to overcome these limitations by proposing a framework for the semantic enrichment, governance, and management of artifact-centric systems. In particular, we propose to shift artifacts to the business level of abstraction, bringing into artifact models the idea of modeling the domain of interest in terms of an ontology, which thus become the heart of the whole artifact system. At the same time, OBDA techniques enable the (efficient) realization of several, recurring architectural solutions adopted in software engineering to attack the complexity of the system-to-be.

Ontologies provide indeed a formal and explicit conceptualization of the domain of interest [34] and are increasingly adopted in the development of information systems, as they facilitate comprehension, sharing, and communication of domain knowledge, at the same time providing a plethora of reasoning services for intelligent data access and integration. This is possible because (domain) ontologies have a formal underpinning in Description Logics (DLs) [7][1], which are decidable fragments of first-order logic (FOL) that can be used to represent structural knowledge of a domain of interest in an unambiguous, formally grounded way. The plethora of reasoning services associated with DLs[2], including, e.g., query answering, consequently allow for a run-time, live exploitation of ontologies, which goes far beyond their usage for modeling purposes only.

Even though the contribution of this work is orthogonal to the artifact/process modeling language of choice, we show how our framework can be concretely exploited by grounding it in the recently proposed Guard-Stage-Milestone (GSM) artifact modeling language [25,38].

The choice of GSM is motivated by three main reasons: (i) GSM has a precise execution semantics which provides a solid basis supporting both implementation-related aspects and formal investigation; (ii) it particularly benefits from capturing data at the conceptual level, as its declarative nature intensively relies on queries posed over the data to properly drive the execution and evolution of processes and artifacts, and (iii) its main constructs have been recently adopted by the Object Management Group in the standard for Case Management Modeling Notation (cf. [49]).

The contributions of this paper can be summarized as follows:

- We provide a formal definition of the *semantic GSM* model. Differently from the classical GSM, in semantic GSM the information model is given in terms of an ontology, and conditions on data and artifact status attributes, used in the specification of GSM lifecycles, are all expressed over the ontology. Furthermore, in our formalization the GSM lifecycle schema itself is modeled through an ontology. The advantage of this feature is twofold: on the one hand it allows for advanced forms of querying over the status of GSM; on the other hand, the framework provides a common, uniform representation for both the lifecycle and the data schema. To guide the modeling of both such aspects, we provide an *upper* ontology which constitutes the (abstract) core of the overall conceptual schema to be defined in each artifact. Each specific artifact provides then its own specialization of this upper layer, enriching the ontology with its own lifecycle elements, relations, and business objects.
- We enrich the semantic GSM framework by enabling the *linkage* of the ontology towards autonomous database systems, possibly with heterogeneous schemas. To this aim, we borrow the notion of *mapping* from the data integration [28,41] and OBDA [50] literature. The mapping actually establishes a semantic correspondence between data stored in data sources and the instances of the ontology. This correspondence consequently fosters collaboration and communication among different artifact-centric systems, heterogeneous and legacy data management systems, and multiples applications, as they can continue to work with their own data formats and schemas, but the data they maintain can be understood in terms of the ontology. In particular, we discuss two main software engineering scenarios of practical interest in which these needs clearly arise:
  - a system constituted by a back-end information subsystem, controlled by a set of semantic GSM artifacts, and multiple front-end applications running their own processes, which, however, need to access data produced by the back-end;
  - a system encompassing multiple interacting subsystems running their own internal processes, on the top of which an ontology is posed to provide a global view of the manipulated data, which in turn allows to monitor and govern the underlying processes at the business level.

[1] DLs are the logical counterpart of OWL, the W3C standard for ontology specification http://www.w3.org/TR/owl2-overview/.

[2] See, for instance, the services offered by the DL-based reasoners presented by [19,23,35,51,52,55].

– We discuss an instantiation of the framework for semantic monitoring and governance of artifact systems adopted within the EU project ACSI—Artifact Centric Service Interoperation[3], and its application to a real-world use case in the energy domain, investigated in such a project. Examples provided throughout the paper are also taken from the ACSI energy use case. In fact, this use case triggered the research presented in this paper, providing at the same time the fundamental motivations for introducing our framework, and a valid test-bed for experiencing it.

In principle, the framework we present in this paper is parametric with respect to the language used for representing the ontology and for querying it, as well as with respect to the forms of mappings that can be adopted to link the ontology with external data management systems. The only requirement we need to impose is the adoption of an ontology language that is able to encode the upper ontology we put at the core of the framework and to extend it to the domain-specific component of the artifact ontology. We notice that the language expressivity requested to meet this requirement is quite limited and that many common basic ontology languages essentially provide it. As for the query language, we only impose that it has to guarantee decidable query answering. For the sake of concreteness, we propose accordingly the use of a query language which allows for decidable query answering even over expressive ontologies and at the same time ensures enough expressibility for modeling purposes. Even though a complete investigation of the computational problems related to reasoning that arise in the presence of specific choices for the mentioned languages is out of the scope of the present paper, we discuss in depth these issues for the instantiation of the framework that we investigated in the ACSI project.

The rest of the paper is organized as follows: In Sect. 2 and in Sect. 3, we provide some preliminaries on the GSM model, and on DLs and their linkage to data, respectively. In Sect. 4, we propose our framework for semantic GSM artifacts. In Sect. 5, we discuss the architecture in which multiple front-end applications access the data produced by a set of back-end GSM semantic artifacts. In Sect. 6, we present the architecture in which the ontology is used as a conceptual, global entry-point to understand the data produced by multiple running (relational) processes, and discuss how the framework can be exploited towards semantic monitoring and governance of such processes. In Sect. 7, we describe the instantiation of such framework for semantic governance of processes as realized within the ACSI project. Then, in Sect. 8, we discuss some related work, and, in Sect. 9, we close the paper with a final discussion and conclusions.

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

2 The GSM Model

In this section we describe the main characteristics of the GSM model. We first provide an informal description and then give precise formalization of the model.

2.1 Informal Introduction

Artifacts, or *artifact types*, are key business entities of a given domain, which are characterized by

– a *data schema* (also called *information model*) that captures the data maintained by the artifact,
– a *lifecycle schema* that specifies the possible progressions of the artifact, and how the underlying data are manipulated as the result of a progression step.
– a set of *artifact instances*, which are instantiations of the corresponding data and lifecycle models of the artifact type. The description of a particular business process may involve several instances of different artifacts types.

The GSM artifact modeling language, recently introduced by [25] and [38], provides means for specifying business artifact lifecycles in a declarative manner, using intuitively natural constructs that correspond closely to how executive-level stakeholders think about their business. The main GSM notions and components of an artifact are

– the *data schema* (Att)—a set of (possibly nested) attributes, used to capture the domain of interest, which can be either
  – *data attributes*, which represent data relevant to the business,
  – *status attributes*, which hold information about the progress of the artifact instance along its lifecycle.
– *Sentries*: data-aware expressions, involving events and conditions over the artifact data schema. Sentries have the form **on** e **if** $cond$, where e is an event and $cond$ is a condition over data. Both parts are optional, supporting pure event-based or condition-based sentries.
– *Tasks*: units of atomic business-relevant work that are to be performed by an external agent (either human or machine) in order to update the data schema of an artifact instance.
– *Milestones* (Mst): a set of sentries, corresponding to business operational objectives, achieved on the basis of triggering events and/or conditions over the data schema.
– *Stages* (Stg): a set of elements, which correspond to clusters of activities intended to achieve milestones, and may be organized into a hierarchy, as they can be either:
  – *atomic stages*, containing exactly one atomic *task*.

  - *composite stages*, containing other sub-stages.

  At a given moment in time, a stage may be activated (having a status *open*), which corresponds to a state when activities within the stage are being executed. Along the process, any stage may be executed multiple times, but it cannot have two occurrences that are being executed simultaneously.
- *Guards* (Grd): a set of sentries, which control when a stage can be activated for execution.
- *Events*: set of typed events, which describe the interaction between artifact instances and the environment and which can be either:
  - *task invocation*, whose instances are populated by the data from data schema and then sent to the environment to perform a task;
  - *task termination*, whose instances represent the corresponding answer from the environment and are used to incorporate the obtained result back into the artifact data schema;
  - *status event*, which correspond to any change of a status attribute, such as opening a stage or achieving a milestone, and can be further used to govern the artifact lifecycle.
  - *one-way events*, which are sent by the environment and which are used to trigger specific guards or milestones.

The operational semantics for GSM is centered around two notions:

- *snapshot*—at any point in time it is the state of any given artifact instance, which is stored according to its data schema, and is characterized by: (i) values of attributes in the schema, (ii) status of its stages (open or closed), and (iii) status of its milestones (achieved or invalidated).
- *business step*, or *B-step* (formally defined in [25]), which corresponds to a transition from one snapshot of the system (before processing the event) to a new one, resulting from the incorporation of an event sent by the environment. Incorporation of an event corresponds to process all the effects that the event triggers in the system. Such effects are determined based on a set of Event-Condition-Action (ECA) rules and result in issuing a set of status events, each of which can trigger further changes.

In order to guarantee that the set of status events generated during a B-step is actually finite, a GSM schema has to be *well-formed*. Indeed, since the ECA rules can contain negation, they suffer from the same well-known issues in logic programming and datalog, namely they can keep firing indefinitely. Such an undesired behavior is avoided by requiring a sort of stratification (see, e.g., [6,31]), which imposes ECA rules to be acyclic and fire in a specific order.

2.2 Formal Basis

This section formalizes the concepts introduced previously and gives a brief intuition of the incremental semantics for GSM.

Definition 1 *(GSM schema)* A GSM schema is a tuple $(x, Att, Stg, Mst, Lcyc)$, where

1. x is a variable that ranges over (IDs of) instances of the artifact;
2. Att, Mst and Stg are the sets described above and they are called the *data schema* of the artifact;
3. $Lcyc = (Substage, Task, Owns, Guards, Achv)$ is the lifecycle schema, where
   (a) $Substage$ is a hierarchical relation over Stg;
   (b) $Task$ is a function from atomic stages in Stg to the set of possible tasks;
   (c) $Owns$ is a function from Stg to finite, non-empty subsets of Mst;
   (d) $Guards$ is a function from Stg to finite sets of *sentries* (see below);
   (e) $Achv$ is a function from Mst to finite sets of *sentries*;

While sets Stg and Mst are simply the set of stages and milestones, respectively, the set Att is the union of two disjoint sets: att_d, the *data attributes* and att_s, the *status attributes*, plus a special attribute $LastIncEventType$ that stores the type of the event that is currently being consumed. Formally, $Att = att_d \cup att_s \cup LastIncEventType$. The set of status attributes is composed by boolean attributes s for each stage $s \in Stg$, which is $true$ if s is currently open or $false$ otherwise, and boolean attributes m for each milestone $m_j \in Mst$, which specifies whether m is achieved ($true$) or invalidated ($false$).

We now introduce some preliminary definitions required to define the lifecycle schema. We assume to have a set of event names EVENT and a domain Δ that includes individuals used to interpret data attributes and the two boolean values $true$ and $false$.

Definition 2 *(Snapshot)* A snapshot of a GSM data schema is an assignment function from attribute names to the domain and the set of event names $\Sigma : Att \rightarrow \Delta \cup \text{EVENT}$ such that

- $\Sigma(a) \in \{true, false\}$ for each $a \in att_s$ and
- $\Sigma(LastIncEventType) \in \text{EVENT}$.

A snapshot Σ is a snapshot for a GSM schema if it satisfies the following *invariants*:

- GSM-1: A stage and its milestone(s) cannot be both true, i.e., for each stage s and each milestone m owned by s, $\Sigma(s)$ and $\Sigma(m)$ cannot be both true.
- GSM-2: No activity in closed stage. If $\neg\Sigma(s)$ for stage $s \in Stg$ and s' is substage of s, then $\neg\Sigma(s')$.

We now briefly introduce the condition language that will be used for specifying the sentries. The syntax of such a language is formally presented in [43] and is out of the scope of this paper. We just mention that the variables of the language correspond to attributes in Att and event names in EVENT. Hence, to evaluate a formula, we need to associate variables $(x_1 \cdots x_n)$ occurring in it to $\Delta \cup$ EVENT. Given a formula Φ with variables $(x_1 \cdots x_n)$, we write $\Sigma \models \Phi(x_1 \cdots x_n)$ when $\Phi(\Sigma(x_1) \cdots \Sigma(x_n))$ evaluates to true accordingly to the semantics of the condition language. The language can also refer to the so-called *status events*. A *status event* for a GSM data schema is an expression of the form $\neg a \wedge a'$ or $a \wedge \neg a'$ where $a \in att_s$. To ease the notation, from now on we use $+a$ as a shortcut for $\neg a \wedge a'$ and $-a$ for $a \wedge \neg a'$. The intuitive meaning is that $+a$ is true when a shifted from false to true during the course of a B-step, and analogously for $-a$. A status event can hence refer to two snapshots, Σ and Σ', where we establish the convention, as customary in the verification community, that primed snapshots Σ' are constructed after Σ. Consequently, in formulas, we will use primed variable symbols for variables that should be associated with elements in Δ according to Σ' and unprimed variable symbols for variables that should be associated with elements in Δ according to Σ. Formally, given a formula $\Phi(x_1 \cdots x_n, x'_1 \cdots x'_m)$ where $x_1 \cdots x_n, x'_1 \cdots x'_m \in Att$, the pair (Σ, Σ') satisfies Φ, denoted $(\Sigma, \Sigma') \models \Phi$ if $\Phi(x_1/\Sigma(x_1) \cdots x_n/\Sigma(x_n), x'_1/\Sigma'(x_1) \cdots x'_m/\Sigma'(x_m))$ evaluates to true, where $(x_i/\Sigma(x_i)$ substitutes to x_i the value $\Sigma(x_i)$ in Φ.

We are now ready to define the set SENTRY of sentries for a GSM schema. A *sentry* for a GSM data schema is a boolean formula of the form $\tau \wedge \gamma$, where

- τ is either of the following:
  - empty;
  - LastIncEventType $= E$ or
  - $\{+, -\}a$ for some status attribute $a \in att_s$.
- γ is a formula that contains no event type variables nor status events.

Notice that a sentry $\tau \wedge \gamma$ can be expressed in the classical form as **on** τ **if** γ.

We now turn to the notion of event. We already discussed status events above (and the way they can be used in sentries), so we now focus on events that allow artifacts to communicate with the *environment*. The environment represents the external world, or, in other words, everything that is not modeled as an artifact. The environment performs external tasks, such as human tasks, that are invoked by the artifacts through business events. *One-way* events are sent unsolicitedly from the environment to an artifact or from an artifact to another artifact. An incoming *one-way (event) type* is a triple $E = (N, O, \psi)$, where $N \in$ EVENT is the event name, O is the event payload structure which is a list of attributes in att_d, and ψ is a (post-)condition whose variables refers to attributes in O. A *one-way event instance*, or simply a one-way message event is a pair $e = (N, p)$ where $p : O \to \Delta$ is the payload such that $p \models \psi$. The condition ψ in a one-way event type formally represents restrictions on the output attributes.

Before formally introducing task invocation and task termination event types, we define tasks. Let TASK be a set of *task names*, disjoint from the other sets of names already established. A *task* is a tuple (T, I, O, ψ) where $T \in$ TASK is a task name, $I \subseteq att_d$ are the input attributes, $O \subseteq att_d$ are the output attributes, and ψ is a logical formula in the condition language expressing the postconditions of the task. Given that the postcondition should refer to two different snapshots Σ and Σ', where Σ is the snapshot of the system when the task is invoked and Σ' is the next snapshot when it finishes, ψ refers to attributes in I without primes and attributes in O with primes. A *task invocation event type* is then a pair $E = (T, I)$ where T is a task name and I are the input attributes of T. A *task invocation event instance* of type E is a tuple $e = (T, p)$ where $p : I \to \Delta$ is the input payload of the event. A *task termination* event type is a triple $E = (T, I, O)$ where T is a task name, and I and O are the input and output attributes of T. A *task termination event instance* is a triple $e = (T, p, p')$ where (T, p) is a task invocation event instance, $p' : O \to \Delta$ is the output of the task, and $(p, p') \models \psi$ evaluates to true. Here p is called the *input payload* of e and p' is called the *output payload* of e.

Next, we present our running example, which is extracted by a real-world use case scenario developed for the FP7 European Project ACSI.

Example 1 From a high-level perspective, the electric supply system is a net of *control points* (CPs) which generate and distribute the energy for a whole country. Each control point is a point in the net where two electric companies exchange energy between each other. A centralized organization called *system operator* is in charge of planning the production and monitoring the energy trade. Every month, for a certain control point, each company participating in the CP submits a so-called *monthly report application* to the system operator, which contains a set of measurements, each describing energy trade for a specific day with the other company connected to the control point. Such values are determined by

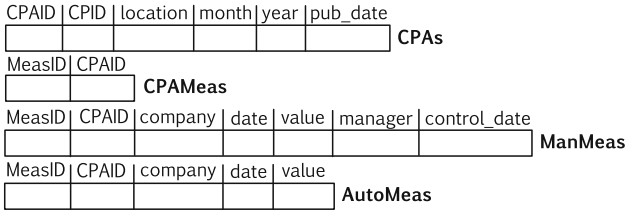


Fig. 1 Graphical representation of the *control point assessment* artifact data schema described in Example 1

companies either automatically by hardware or manually by an energy manager. When the system operator receives two applications for each control point, it cross-checks data and publishes a *control point assessment*, possibly after a manual inspection when values do not match.

We model such a process in GSM by introducing a *control point assessment* (CPA) artifact. We assume a relational representation of data, and Fig. 1 provides a graphical representation of it. The data schema of the artifact provides information about both the assessment to be published for a specific CP and the measures received by the two companies connected to the CP.

Relation $CPAs$ stores the id of the control point assessment ($CPAID$); the id and location of the control point ($CPID$ and $location$); the month and the year the assessment refers to and its publication date. Relation $CPAMeas$ keeps track of the measures ($MeasID$) chosen by the system operator for a given CPA. This measure is among those which have been proposed (in the monthly report application) by the two companies connected to the control point. Later on we explain how such values are chosen. The other two relations store the set of measurements performed by the companies. In particular, manually determined values are kept in the $ManMeas$ table, which contains the id of the measure, the company name, the value of the measure, the date it refers to, the manager in charge of the manual input, and the input date. Automatically determined values, on the other hand, are stored in the $AutoMeas$, in which the id of the measure, the company, the date, and the value of the measure are the only relevant information.

Figure 2 shows a graphical representation of the CPA artifact lifecycle. When an instance of CPA artifact is created (for a specific CP) by a one-way creation event from the environment, Requesting stage opens. The scope of associated activity is to request monthly measures from the two companies connected to the CP. Indeed, ReqMR1 and ReqMR2 open in parallel and their tasks send a task invocation event to the environment. When a response event is consumed, its payload, containing all the information for a monthly request application, is written in the data schema of the artifact (precisely in the $ManMeas$ and $AutoMeas$ relations in Fig. 1) and milestone MR1Received (or MR2Received) is achieved. When both ReqMR1 and ReqMR2 close, milestone MRAppsReceived is achieved and stage Requesting closes. Stage CPADrafting opens when MRAppsReceived and its three substages take care of analyzing the measurement for each day. In particular, EqualMeas opens if there exist a couple of measurements provided by the two companies which agree on their values for a specific date. In this case, indeed, such a measure is written in table $CPAMeas$. In other words, task wrMeas takes care of writing measurements which the two companies agree on. Substage DiffMeasAuto opens when there are two measurements that disagree (for a specific day) but one of them has been performed automatically. In this case, indeed, the system operator will chose the automatic measure to be written in $CPAMeas$. The last substage, DiffMeasMan, covers the case in which the measurements disagree and they are both performed automatically or manually. A manual inspection is then needed in order to choose one of those.

Tables 1 and 2 show sentries for guards and milestone for the energy example, respectively. Such sentries are expressed as first order logic formulas. □

3 Description Logic Ontologies and Their Linkage to Data

In this section we recall some basic notions on Description Logic ontologies, and on mechanisms to map ontologies to databases, which are taken over from the research on data integration [28,41].

3.1 Description Logic Ontologies

Description Logic (DL) ontologies model the domain of interest in terms of *objects* (a.k.a. individuals), *concepts*, which are abstractions for sets of objects, *roles*, which denote binary relations between objects, *value-domains*, which denote sets of values, and *attributes*, which denote binary relations between objects and values. In the rest of the paper we refer to an alphabet Γ, starting from which DL expressions are built. Γ is the disjoint union of Γ_P, containing symbols for atomic concepts, atomic value-domains, atomic attributes, and atomic roles, and Γ_C, which contains symbols for constants (each denoting either an object or a value). Complex expressions are constructed starting from atomic elements, and applying suitable constructs. Different DLs allow for different constructs.

A DL ontology is constituted by two main components: a *TBox* (i.e.,"Terminological Box"), that stores a set of universally quantified FOL assertions stating general properties of concepts and roles, thus representing intensional knowledge of the domain, and an *ABox* (i.e.,"Assertional Box"), that is constituted by assertions on individual objects, thus specifying extensional knowledge. Again, different DLs allow for

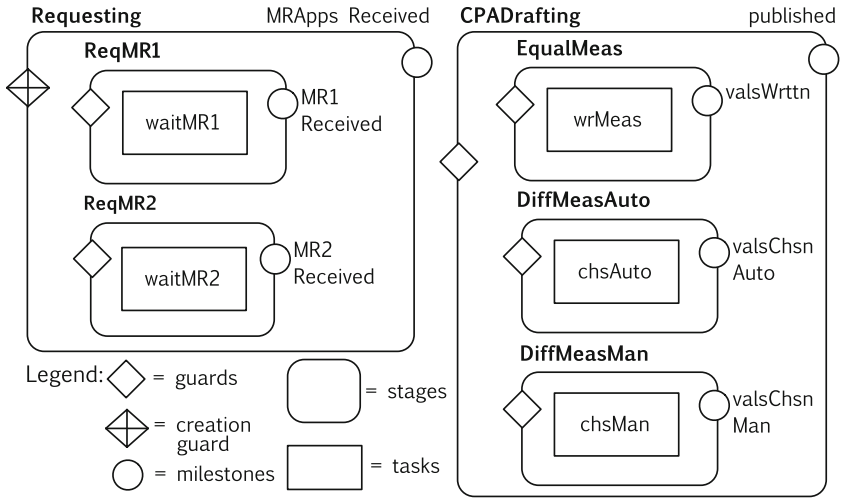


Fig. 2 Graphical representation of the *control point assessment* artifact lifecycle described in Example 1

Table 1 Guards for the lifecycle in Fig. 2

Stage	Guard sentry	
	on	**if**
Requesting	CPACreation Event	–
RequestMR1	+Requesting	–
RequestMR2	+Requesting	–
CPADrafting	+MRAppsReceived	–
EqualMeas	+CPADrafting	$\exists id, m1id, m2id, c1, c2, d, v.CPAs(id,-,-,-,-,-) \wedge c1 \neq c2 \wedge m1id \neq m2id \wedge$ $((ManMeas(m1id, id, c1, d, v, -, -) \wedge ManMeas(m2id, id, c2, d, v, -, -)) \vee$ $(Auto\_meas(m1id, id, c1, d, v) \wedge Auto\_meas(m2id, id, c2, d, v)) \vee$ $(ManMeas(m1id, id, c1, d, v, -, -) \wedge AutoMeas(m2id, id, c2, d, v)))$
DiffMeasAuto	+CPADrafting	$\exists id, m1id, m2id, c1, c2, d, v.CPAs(id,-,-,-,-,-) \wedge$ $c1 \neq c2 \wedge v1 \neq v2 \wedge m1id \neq m2id \wedge$ $ManMeas(m1id, id, c1, d, v1, -, -) \wedge AutoMeas(m2id, id, c2, d, v2)$
DiffMeasMan	+CPADrafting	$\exists id, m1id, m2id, c1, c2, d, v1, v2.CPAs(id,-,-,-,-,-) \wedge$ $c1 \neq c2 \wedge v1 \neq v2 \wedge m1id \neq m2id \wedge$ $((ManMeas(m1id, id, c1, d, v1, -, -) \wedge ManMeas(m2id, id, c2, d, v2, -, -)) \vee$ $(AutoMeas(m1id, id, c1, d, v1) \wedge AutoMeas(m2id, id, c2, d, v2)))$

Table 2 Milestones for the lifecycle in Fig. 2

Milestone	Milestone sentry	
	on	**if**
MR1Received	$WaitMR1TermEvent$	–
MR2Received	$WaitMR2TermEvent$	–
MRAppsReceived	–	MR1Received $\wedge$ MR2Received
valsWrttn	$wrMeasTermEvent$	–
valChsnAuto	$chsAutoTermEvent$	–
valChsnMan	$chsManTermEvent$	–
published	+valsWrttn $\vee$ +valsChnsAuto $\vee$ +valsChsnMan	(valsWrttn $\vee$ ¬EqualMeas) $\wedge$ (valsChsnAuto $\vee$ ¬DiffMeasAuto) $\wedge$ (valsChsnMan $\vee$ ¬DiffMeasMan)

different kinds of TBox and/or ABox assertions. Formally, a DL ontology $\mathcal{O}$ over an alphabet Γ is a pair $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox, whose predicate symbols and constants are from Γ.

The semantics of a DL ontology $\mathcal{O}$ over an alphabet Γ is given in terms of FOL interpretations for Γ (cf. [7]). We denote with $Mod(\mathcal{O})$ the set of models of $\mathcal{O}$, i.e., the set of FOL-interpretations that satisfy all TBox axioms and ABox assertions in $\mathcal{O}$, where the definition of satisfaction depends on the DL language in which $\mathcal{O}$ is specified. An ontology $\mathcal{O}$ is *satisfiable* if $Mod(\mathcal{O}) \neq \emptyset$. A logical sentence, i.e., a closed formula, ϕ, expressed in a certain language $\mathcal{L}$, is *entailed* by an ontology $\mathcal{O}$, denoted $\mathcal{O} \models \phi$, if ϕ is satisfied by every interpretation in $Mod(\mathcal{O})$ (where, again, the definition of satisfaction depends on the language $\mathcal{L}$). All the above notions naturally apply to a TBox $\mathcal{T}$ alone.

Various reasoning services can be performed over DL ontologies and are supported by state-of-the-art automated reasoners (see, e.g., [35,52,55]). Among such services, intensional ones do not consider the ontology ABox, and disclose properties only implicitly specified in the ontology, as well as permit to verify the quality of the modeling. Instead, extensional reasoning also involve the ABox. The most important reasoning service of this kind is *query answering*, which we describe below.

In presenting our framework, we do not refer to a specific ontology language. We only assume that the language we use is expressive enough to capture the upper ontology (in fact a TBox) that we will introduce in Sect. 4 and for specializing it into domain-specific ontologies. As already said in the introduction, also basic ontology languages essentially fulfill these requirements. Among such languages we often refer to *DL-Lite*, which is in fact a family of light-weight DLs [18,21], constituting the formal underpinning of OWL 2 QL, one of the tractable profiles of OWL 2 [46], the W3C standard language for ontology specification [47]. DLs of the *DL-Lite* family allow for specifying basic ontology constructs, such as subsumption between concepts, subsumption between properties (i.e., roles or attributes), typings of properties, mandatory participation of concepts into properties, and functionality of properties. Such DLs guarantee tractability of standard reasoning services over ontologies,and in particular enable for FOL-rewritable query answering, which is a crucial requirement when ontologies are linked to databases, and which we formalize below in the paragraph on querying DL ontologies.

In the rest of the paper, for the sake of simplicity, we will provide only graphical representation of ontologies (or, better, of approximations thereof) given through Entity–Relationship diagrams. Of course, such diagrams are in fact encoded into suitable logical axioms, whose semantics is given in terms of FOL interpretations, as said above.

3.2 Querying DL Ontologies

Given a language $\mathcal{L}$, an $\mathcal{L}$-query over a DL ontology (or TBox) with alphabet Γ is a (possibly open) $\mathcal{L}$-formula over Γ. Let $q(\mathbf{x})$ be an $\mathcal{L}$-query with free (a.k.a. distinguished) variables $\mathbf{x}$ over an ontology $\mathcal{O}$. Then, a tuple $\mathbf{t}$ of constants from Γ_C is a *certain answer* for $q(\mathbf{x})$ if $\mathcal{O} \models q(\mathbf{t})$, where $q(\mathbf{t})$ is the closed formula, i.e., a sentence, obtained by substituting $\mathbf{x}$ with $\mathbf{t}$. The above definition applies to a boolean $\mathcal{L}$-query (i.e., a formula with no free variables) in the standard way: $\langle\rangle$ is the certain answer to q if $\mathcal{O} \models q$, and we say that the query is certainly true; conversely, if $\mathcal{O} \not\models q$, the set of certain answers is empty, and we say that the query is certainly false. Then, the *query answering* reasoning service is defined as follows: given a DL ontology $\mathcal{O}$, and an $\mathcal{L}$-query q over $\mathcal{O}$, compute the set of certain answers to q over $\mathcal{O}$. We denote such set by $cert(q, \mathcal{O})$. It is easy to see that this is a form of reasoning under incomplete information. An important notion related to query answering is that of *FOL-rewritability*, which intuitively means that one can obtain the certain answers to a query q over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ by first rewriting q into a new first-order query q_r over $\mathcal{O}$ and then evaluating q_r over the ABox $\mathcal{A}$ seen as a database. Such a rewriting has to depend only on the TBox. We call q_r the *FOL-rewriting* of q with respect to $\mathcal{T}$ (we refer to [18] for the formal definition). Notably, a FOL query can be translated into SQL and, therefore, can be evaluated over any relational DBMS managing the underlying ABox. This has of course a crucial impact in performances and even in practical realizability of the query answering reasoning service for ontologies.

We notice that our framework is parametric with respect to the language used for querying ontologies. However, for computational reasons, the expressivity of such language has to be somehow controlled in the practice. For example, it is well known that answering FOL queries in the presence of incomplete information is undecidable [3], whereas the most expressive language for which decidability of query answering over various DL ontologies has been shown is that of union of conjunctive queries (UCQs) (e.g., [18,32]).

In the rest of the paper, we refer to a query language proposed in [17], which allows for the use of all FOL constructs in queries in a semantically controlled way. Intuitively, decidability (and even tractability, in some notable cases) of query answering is preserved in such language since reasoning over incomplete information is needed only to answer the UCQ-subcomponents of the queries. This is obtained by virtue of a particular semantic interpretation of the queries allowed in this language, based on the use of an epistemic operator. More precisely, one such a query, called *ECQ*, over a DL ontology $\mathcal{O}$ is a (possibly open) domain independent formula of the following form:

$$Q \longrightarrow [q] \mid \neg Q \mid Q_1 \wedge Q_2 \mid \exists x.Q \mid x \ \mathsf{op} \ y,$$

where q is a UCQ over $\mathcal{O}$, op is one among $=, \neq, >, <, \geq$, and $\leq$, and $[q]$ denotes that q is evaluated under the (minimal) knowledge operator (cf. [17]). To compute the certain answers $cert(Q, \mathcal{O})$ to an ECQ Q over an ontology $\mathcal{O}$, we can compute the certain answers over $\mathcal{O}$ 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.

Interestingly, query answering of UCQ in *DL-Lite* is FOL-rewritable, as shown by [18]. As a consequence of this, also query answering of ECQ queries in *DL-Lite* is FOL-rewritable.

3.3 Linking Data To Ontologies

In the past years, a new paradigm for information integration, called OBDA, has been proposed, which is based on the use of an ontology (in fact a TBox) acting as mediated (a.k.a. global) schema suitably linked to data sources [50]. In OBDA, data sources are seen as a relational database. As in (virtual) data integration, linkage towards data sources is realized through mapping assertions. The most expressive mapping assertions considered in the data integration literature are the so-called GLAV assertions [41], which are expressions of the form $\phi(\mathbf{x}) \rightsquigarrow \psi(\mathbf{x})$, where $\phi(\mathbf{x})$ is a query over the data sources and $\psi(\mathbf{x})$ is a query over the global schema (the ontology in OBDA). Intuitively, such a mapping assertion specifies that the tuples returned by the evaluation of $\phi(\mathbf{x})$ over the source database semantically correspond (in a sense that will be clarified below) to the formula $\psi(\mathbf{x})$ and, therefore, create the bridge between the data in the sources and the objects satisfying the predicates in the ontology. When the formula $\phi(\mathbf{x})$ is a single atom formula of the form $R(\mathbf{x})$, with R a source relational predicate, the mapping assertion is called *Local-As-View (LAV)*. When the formula $\psi(\mathbf{x})$ is a single atom formula of the form $S(\mathbf{x})$, with S an ontology predicate, the mapping assertion is called *Global-As-View (GAV)*.

Formally, an *OBDA specification* is a triple $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$, where $\mathcal{T}$ is a DL TBox, $\mathcal{D}$ is a database, and $\mathcal{M}$ is a set of mappings between $\mathcal{T}$ and $\mathcal{D}$. Its semantics is given in terms of FOL interpretations I over the alphabet of $\mathcal{T}$, such that (i) I satisfies $\mathcal{T}$, and (ii) I satisfies $\mathcal{M}$, i.e., for each assertion $\phi(\mathbf{x}) \rightsquigarrow \psi(\mathbf{x})$ in $\mathcal{M}$ we have that for every tuple $\mathbf{t}$ in the evaluation of $\phi(\mathbf{x})$ over $\mathcal{D}$ it holds that $\psi(\mathbf{t})$ evaluates to true in I, where the notions of evaluation of $\phi(\mathbf{x})$ over $\mathcal{D}$ and $\psi(\mathbf{t})$ over I depend on the particular language in which such queries are specified. Notice that the above (classical) notion of mapping satisfaction actually considers mapping assertions as *sound* implications from the database to the ontology. The different notion of *complete* mappings considers them as opposite implications, i.e., in this case I satisfies an assertion of the form above if for every tuple $\mathbf{t}$ such that $\psi(\mathbf{t})$ evaluates to true in I it holds that $\psi(\mathbf{t})$ is in the evaluation of $\phi(\mathbf{x})$ over $\mathcal{D}$ (cf. [41]). The interpretations satisfying both the ontology and the mapping are the models of the OBDA specification $\mathcal{S}$, and the set of such models is denoted by $Mod(\mathcal{S})$.

A query posed over and OBDA specification $\mathcal{S} = \langle \mathcal{T}, \mathcal{M}, \mathcal{D} \rangle$ is a query posed over its TBox $\mathcal{T}$. Given one such query q, the notion of certain answers to q over $\mathcal{S}$, denoted $cert(q, \mathcal{S})$, is the natural generalization of the analogous notion given for stand-alone ontologies. Analogously, we can naturally extend the notion of FOL-rewritability to OBDA specifications.

We notice that, for computational reasons, it is necessary in practice to control the expressive power of the languages used to specify queries in the mapping. We notice, however, that the query $\phi(\mathbf{x})$ in a mapping assertion is posed over a database, where query answering actually amounts to simple query evaluation. We can, therefore, assume that such a query is a generic FOL query (possibly expressed in SQL), whereas it will consider $\psi(\mathbf{x})$, which is a query over the ontology, expressed in the language ECQ given above.

4 Semantic GSM

In this section we propose *Semantic GSM*, a novel artifact-centric model which merges the expressing power of ontologies for modeling and querying the data of interest with the capability of GSM to express data evolution. Two solutions can be adopted to obtain such a coupling. In the first, the ontology models the domain of interest only, whereas the lifecycle is defined as in classic GSM. The second one amounts to use an integrated ontology that describes both the data and the lifecycle schema. Here, we present the second option, with the aim of providing a unified view of the whole framework that allows for answering complex queries. Therefore, arbitrary queries over the lifecycle are now enabled, as we can access the whole lifecycle structure, i.e., both status and data attributes. As an example, we can directly inquire which atomic stages are waiting for a task termination event from the environment, or which composite stages have an achieved milestone, namely, those which have already been executed. Notice that, in classic GSM, answering such queries requires an extra effort since the lifecycle schema is not explicitly represented.

Let us assume an alphabet Γ for concepts, value-domains, attributes, roles, and constants. Intuitively, the integrated ontology describes, in common language, both the data and the lifecycle schema and provides as its core, a domain independent "upper" ontology which has to be specialized by means of a "lower" ontology, in order to represent the schema of a specific process. Figure 3 shows a graphical representation of the upper ontology, in which EventInstance,

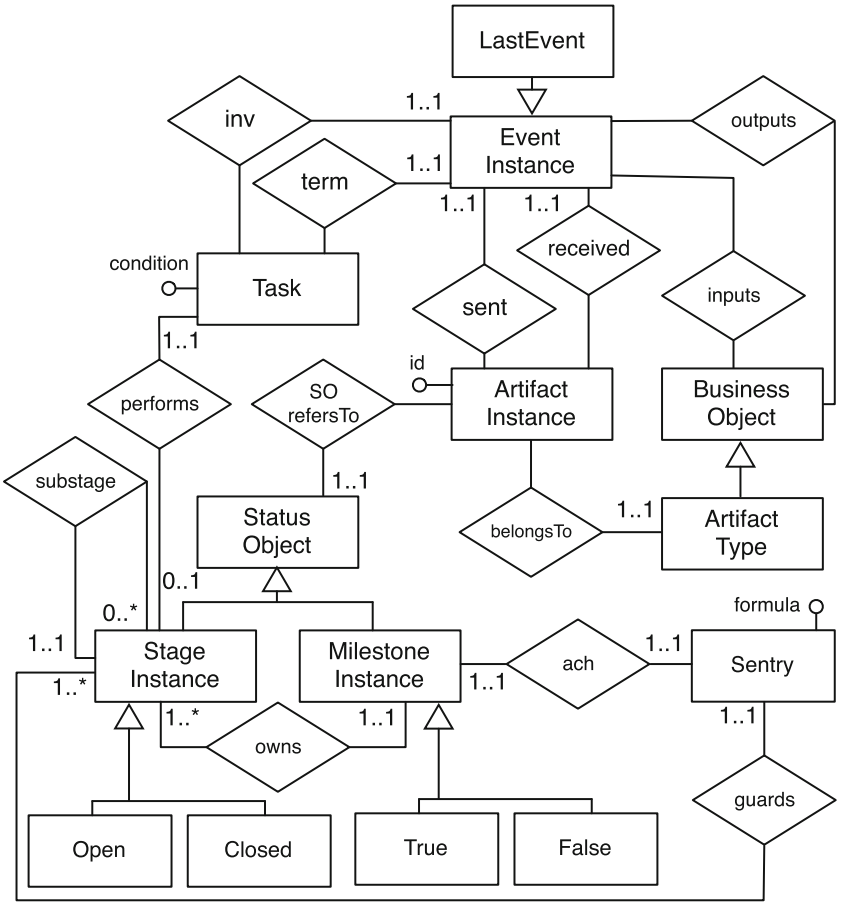


Fig. 3 Upper ontology for semantic GSM artifacts

ArtifactInstance, BusinessObject and StatusObject are the main concepts. Intuitively, business objects represent (the classic GSM) data attributes, while status objects (and relationships between them) the lifecycle. An artifact instance belongsTo a specific ArtifactType. An artifact type is a particular BusinessObject in that it has a lifecycle, i.e., a set of StatusObjects connected to itself (through the SOrefersTo role). A status object is either a milestone or a stage and this specialization is disjoint and complete. Stages, being either open or closed, can be hierarchically organized by the transitive substage role and should have at least one guard (which is a sentry) and a milestone. Milestones have an achieving sentry, and they are either true or false. We distinguish invocation events and task termination events, either of which can be the event currently being consumed by the system. The LastEvent class is actually a singleton class, i.e., it allows for one instance only, given that a GSM system processes one event at a time. Finally, tasks are performed by stages and they have a invocation and a termination event.

The above upper TBox does not describe a specific artifact process as it lacks all the domain-dependent entities. However, it can be specialized to model an actual GSM schema. The main concept of the lower ontology, i.e., the one that is intended to have an associated lifecycle, specializes ArtifactType, while all other concepts specialize BusinessObject. Having two separate concepts for artifact types and instances allows us to decouple data from the execution, as we can have artifacts data even when they are not evolving. Notice that in classic GSM, instead, artifacts always have an associated lifecycle. Moreover, EventInstance generalizes all event type instances required by the process and the same holds for Task. Also roles can be specialized to connect the specialized concepts. Precisely, inv, term, inputs and outputs should relate specific concepts, as well as substage, guards, ach and owns.

Definition 3 A *semantic GSM schema* is a TBox $\mathcal{T}$ over Γ containing the upper ontology in Fig. 3.

According to its definition, a semantic GSM schema has to be expressed in an ontology language that allows for the specification of the ontology in Fig. 3 and for specializing it in the schema of the application at hand. Namely, this means that such language has to capture basic ontology constructs, such as ISA between named concepts or roles (e.g., LastEvent is a subconcept of EventInstance), disjointnesses between named concepts (e.g., Open is disjoint from Closed), role typings (e.g., the role owns is typed on StageInstance and MilestoneInstance), cov-

ering of concepts (e.g., a StageInstance is either Open or Closed), mandatory participation of concepts to roles (e.g., each Task performs at least a StageInstance), and functionality of roles (e.g., each Task performs at most a StageInstance). We point out that such constructs are all expressible in the W3C OWL standard, but also in less expressive DL languages such as those used to capture classical conceptual modeling languages (see, e.g., the UML class diagram encoding described in [13]). Notably, all the aforementioned constructs, with the exception of concept covering, are all enabled also in lightweight ontology languages such as *DL-Lite*, or some languages in the Datalog$^{+/-}$ framework [14]. In such languages, however, the ability of expressing concept covering can be re-gained, without increasing the complexity of reasoning, through the use of some additional *constraints*, i.e., logical axioms whose interpretation is based on the use of an epistemic operator, in the spirit of the approach proposed by [18]. Intuitively, each such constraint can be seen as a boolean ECQ query q (cf. Sect. 3), which is satisfied by an ontology $\mathcal{O}$ if and only if $\mathcal{O}$ entails q. This actually means that the constraint is not an axiom to be exploited for inferring implicit knowledge by the ontology, as enabled in expressive ontology languages, but is rather an assertion that has to be satisfied over the actual data. In other terms, an ABox compliant with our upper ontology expressed in *DL-Lite* enriched with epistemic constraints has to always explicitly assert whether a StageInstance is Closed or Open, and a MilestoneInstance is True or False. We argue that in practical cases this is not a real approximation since we expect to have always complete information on the closed/open stages or true/false milestones.

We are now ready to provide the notion of snapshot for a semantic GSM schema is given below.

Definition 4 A *semantic snapshot* for a semantic data schema $\mathcal{T}$ is an ABox $\mathcal{A}$ such that:

1. $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable;
2. $(\mathcal{T}, \mathcal{A}) \models \forall s, m.\mathsf{owns}(s, m) \rightarrow (True(m) \rightarrow Closed(s))$;
3. $(\mathcal{T}, \mathcal{A}) \models \forall s, s'.\mathsf{substage}(s, s') \rightarrow (\mathsf{Closed}(s') \rightarrow \mathsf{Closed}(s))$.

Intuitively, (2) and (3) corresponds to GSM-1 and GSM-2 invariants, respectively, explained in Sect. 2.

The condition language used for specifying semantic sentries is a variation of the condition language adopted for classic GSM (cf. Sect. 2), in the sense that the queries over the ontologies it uses are interpreted under the *certain answer* semantics.

A *semantic status event* is an expression of the form $+\mathsf{Stg}(s) \equiv \mathsf{Stg}(s) \wedge \mathsf{Close}(s) \wedge \mathsf{Open}'(s)$ or $-\mathsf{Stg}(s) \equiv \mathsf{Stg}(s) \wedge \mathsf{Open}(s) \wedge \mathsf{Close}'(s)$ where Stg is a subclass of StageInstance, or $+\mathsf{Mst}(m) \equiv \mathsf{Mst}(m) \wedge \mathsf{False}(m) \wedge \mathsf{True}'(m)$ or $-\mathsf{Mst}(m) \equiv \mathsf{Mst}(m) \wedge \mathsf{True}(m) \wedge \mathsf{False}'(m)$ where MstName is a subclass of MilestoneInstance. To simplify the notation, in sentries we write Stg instead of $\mathsf{Stg}(s)$ since, given an artifact instance, in the ABox there is only one individual for each concept Stg subclass of StageInstance (analogously for milestones).

Semantic status events are formulas that refers to two semantic snapshots $(\mathcal{A}, \mathcal{A}')$. The intuitive semantics is similar to that presented in Sect. 2, but, once more, formulas are interpreted under the certain answer semantics.

Definition 5 A *semantic sentry* for a semantic GSM data schema is a boolean formula of the form $\tau \wedge \gamma$, where

- τ is either of the following:
  - empty;
  - $Event$, where $Event$ is the most specific class of the (singleton) individual belonging to concept LastEvent;
  - $\{+, -\}$Stg $\{+, -\}$Mst, where Stg and Mst are as before;
- γ is a formula that contains neither $Event$ nor status events.

Notice that in semantic GSM there is no need for formal definitions of events and tasks, as their properties are already captured by the ontology.

The operational semantics of semantic GSM is grounded on that for classic GSM, as its fundamental concepts are, from an high-level perspective, orthogonal to the logical representation of data. Hence we again rely on the notion of snapshot, which is now as in Definition 4, and B-step, during which the ECA rules are processed. From the technical viewpoint, in semantic GSM ECA rules contain queries over the ontology, and then answering them means reasoning to compute their certain answers (cf. Sect. 3). For this reason, the check for well-formedness is in general more involved than in classic GSM. However, when query answering is FOL-rewritable, checking well-formedness can be actually performed as in the classical GSM setting, modulo the computation of FOL-rewritings of the queries occurring in the ECA rules. In other terms, we are able in these cases to reduce a complex check, which requires to reason over incomplete information, to a standard GSM well-formedness check. This is, for example, the case of ECQ queries issued over *DL-Lite* ontologies (cf. Sect. 7). Well-formedness in other, more expressive, settings requires further investigation which we leave for future studies.

Example 2 We model the energy process described in Example 1 with a semantic GSM schema $\mathcal{T}$. Figure 4 shows a graphical representation of the portion of $\mathcal{T}$ which describes

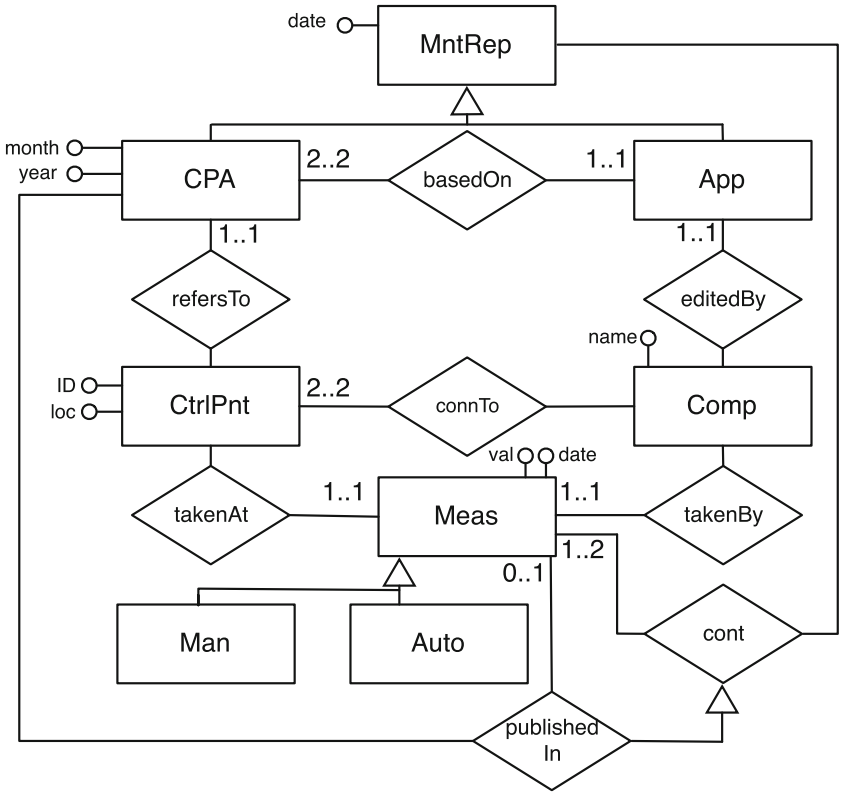


Fig. 4 Fragment of the semantic GSM schema for the energy process in Example 1 describing the domain

the domain of interest. As usual, such a graphical representation is useful for presentation purposes, whereas the ontology TBox is in fact specified through a set of logical axioms (possibly going beyond the ER expressiveness showed in the figure). In this example, the reader may consider the ontology modeled in OWL, or even in *DL-Lite*, provided some suitable approximations, as discussed in before and, more in detail, in Sect. 7. The CPA concept specializes ArtifactType of the upper ontology in Fig. 3, while all other concepts are intended to specialize BusinessObject as a disjoint and complete hierarchy. A monthly report contains a set of (energy) measures and is either a control point assessment or an application. A control point assessment is based on exactly two applications (each one edited by a company) and refers to a control point, which connects exactly two companies. Measures can be either manual or automatic and they are taken at a specific control point by a specific company.

In Fig. 5, a partial fragment of $\mathcal{T}$ which describes the lifecycle of the process is graphically represented. Concepts Requesting, ReqMR1 and ReqMR2 specialize stage instance. Their individuals represent a stage instance of a specific artifact. Roles ReqMR1Substg and ReqMR2Substg specialize the substage role of the upper ontology (once more such a generalization is not pictured) and ReqMst, ReqMR1Mst and ReqMR2Mst, specializing the owns role, make the relation between states and milestones explicit.

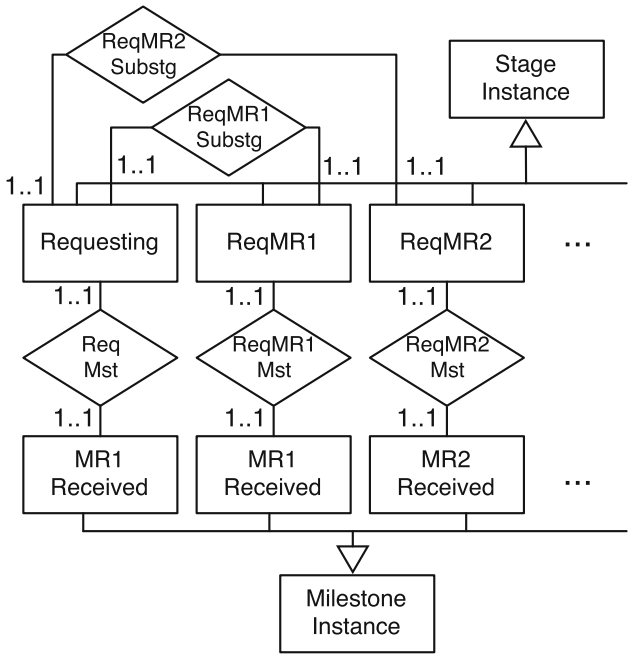


Fig. 5 Fragment of the semantic GSM schema for the energy process in Example 1 partially describing the process' lifecycle

Table 3 shows the guards of the energy example, now expressed in ECQs over the ontology, where CPADrafting, EqualMeas, DiffMeasAuto and DiffMeasMan are subclasses of StageInstance. It is easy to see that, despite their

Table 3 Guards for Example 2

Stage	Guard sentry	
	on	**if**
Requesting	$CPACreationEvent$	–
RequestMR1	+Requesting	–
RequestMR2	+Requesting	–
CPADrafting	+MRAppsReceived	–
EqualMeas	+CPADrafting	$\exists ap1, ap2.App(ap1) \neq App(ap2) \wedge$ $[\exists a, m1, m2, d, v.basedOn(a, ap1) \wedge basedOn(a, ap2) \wedge cont(ap1, m1) \wedge cont(ap2, m2) \wedge$ $date(m1, d) \wedge date(m2, d) \wedge val(m1, v) \wedge val(m2, v)]$
DiffMeasAuto	+CPADrafting	$\exists ap1, ap2, v1, v2.App(ap1) \neq App(ap2) \wedge v1 \neq v2 \wedge$ $[\exists a, m1, m2, d.basedOn(a, ap1) \wedge basedOn(a, ap2) \wedge cont(ap1, m1) \wedge cont(ap2, m2) \wedge$ $date(m1, d) \wedge date(m2, d) \wedge val(m1, v1) \wedge val(m2, v2) \wedge Man(m1) \wedge Auto(m2)]$
DiffMeasMan	+CPADrafting	$\exists ap1, ap2, v1, v2.App(ap1) \neq App(ap2) \wedge v1 \neq v2 \wedge$ $[\exists a, m1, m2, d.basedOn(a, ap1) \wedge basedOn(a, ap2) \wedge cont(ap1, m1) \wedge cont(ap2, m2) \wedge$ $date(m1, d) \wedge date(m2, d) \wedge val(m1, v1) \wedge val(m2, v2) \wedge$ $((Man(m1) \wedge Man(m2)) \vee (Auto(m1) \wedge Auto(m2)))]$

length due to joins, they are easier to manage than the ones in Table 1, referring to the non-semantic GSM modeling of the same process. For example, no unions are requested in the guard for EqualMeas stage.

We notice also that we can now easily pose over the ontology complex queries (possibly not among those designed for the process that the artifact realizes) that could not be immediately expressed over the data schema of a classical GSM artifact as the one given in Fig. 1. For example, we can easily get information about measures sent by a company C to the system operator that are not included in the control point assessment for which they are produced. The query $Q_1(id, y, m, d, v)$ described below returns indeed the CP identifier, the year, month, and date of the control point assessment, and the value of the excluded measure[4].

$\exists cpa, ms.[\exists app, cp.\mathsf{CPA}(cpa) \wedge \mathsf{refersTo}(cpa, cp) \wedge$
$\mathsf{ID}(cp, id) \wedge \mathsf{month}(cpa, m) \wedge \mathsf{year}(cpa, y) \wedge$
$\mathsf{basedOn}(cpa, app) \wedge \mathsf{editedBy}(app, \mathsf{C}) \wedge \mathsf{cont}(app, ms) \wedge$
$\mathsf{date}(ms, d) \wedge \mathsf{val}(ms, v)] \wedge \neg[\mathsf{cont}(cpa, ms)]$

It should be easy to see that to extract the above information from the schema in Fig. 2, a more involved query is needed. Indeed, while in the ontology we can exploit the concept Meas, which generalizes both automatic and manual measures, and the role cont which relates generic measures to generic reports, in the information model schema of classic GSM we have to explicitly access both the $AutoMeas$ and $ManMeas$ relations and separately consider the association of such measures with reports. Such an advantage become more significant as the number of specialized concepts increases.

We also notice that queries over semantic GSM can naturally involve both data and the lifecycle, as done in the query $Q_2(s)$ described below, which returns the closed stages for an artifact instance that is processing the control point assessment of January 2013 for CP with ID 17.

$\exists a, cpa, cp.\mathsf{artifactInstance}(a) \wedge \mathsf{SOrefersTo}(a, s) \wedge$
$\mathsf{closed}(s) \wedge \mathsf{belongsTo}(a, cpa) \wedge \mathsf{month}(cpa, \text{'January'}) \wedge$
$\mathsf{year}(cpa, 2013) \wedge \mathsf{refersTo}(cpa, cp) \wedge \mathsf{ID}(cp, 17)$

In classic GSM such a query is not naturally expressible, as stage instances cannot be returned as the result of a query. It is in principle possible to modify the classic GSM information model to represent the lifecycle schema, but this would require to use extra data structures in the data attributes, thus losing the conceptual distinction between business data and lifecycle data. The upper ontology, instead, provides a natural and clean way to combine such aspects.

Finally, query $Q_3(cpa)$ returns the CPAs for which there exists a stage with a child that has already been executed and one still open. From the business perspective, Q_3 returns the artifacts that are still active and are about to evolve. This information can be useful for the system operator to allocate or deallocate in advance some specific resources.

$\exists id, ps, s_1, s_2, m.\mathsf{belongsTo}(id, cpa) \wedge$
$\mathsf{SOrefersTo}(s_1, id) \wedge \mathsf{SOrefersTo}(s_2, id) \wedge$
$\mathsf{SOrefersTo}(ps, id) \wedge \mathsf{SOrefersTo}(m, id) \wedge$

[4] We assume that the constant C used in the query denotes the object representing the company C.

$\mathsf{substage}(s_1, ps) \wedge \mathsf{substage}(s_2, ps) \wedge$
$\mathsf{Open}(s_1) \wedge \mathsf{Owns}(s_2, m) \wedge \mathsf{True}(m)$

Notably the above query is domain and lifecycle independent. Indeed, not only it does not refer to the lower ontology, but it is also general enough to fit any lifecycle. This is possible by making use of distinctive constructs of ontologies, such as generalization.

Another interesting example of query over the ontology described above is showed in Sect. 7. Such a further query makes it even more evident the importance of reasoning in query answering in semantic GSM. □

The example above makes clear the advantages of using semantic GSM. In the first place, the ontology captures entities of the domain and relationships between them in a clearer and more elegant way than a relational model, which is usually built to serve the implementation level, and not for describing the domain per se. For example, the ontology specifies properties, as for instance the fact that a CPS is based on two applications, which are hidden in the data schema of classical GSM. Furthermore, it allows easier and more expressive queries: on the one hand sentries are more manageable as they can be formulated considering the reasoning services the logics provides (such as certain answer computation), and on the other hand, due to the unified view of the schema, user queries can now directly refer to both data and lifecycle. Finally, it allows for specifying business relevant high-level queries that can be used on all instantiations of semantic GSM, and they are robust to lifecycle refinements.

5 Linking Semantic GSM With Multiple Front-End Applications

In this section, we discuss a combination of GSM, ontologies, and mappings that is suitable in the common situation where the architecture of the system is decomposed into a unique back-end and multiple (possibly legacy) front-ends.

More specifically, we consider the case where

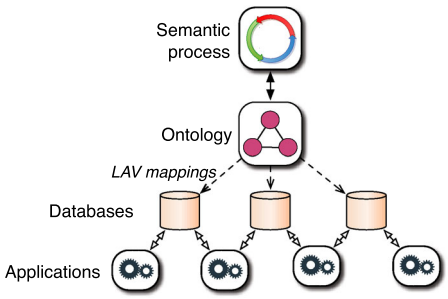
- a unique back-end hosts the business processes that manipulate (i.e., read, write and update) the whole data related to the entire application domain.
- Multiple front-end applications, conforming to different local database schemas, are employed to show (i.e., read and visualize) the data produced by the back-end and to realize services on top of these data; each such an application can also write its own data into the corresponding local database, but without affecting the information maintained by the back-end, that is, local updates in this setting should not be propagated towards the ontology.

Example 3 Consider a company whose main asset is knowledge management in e-agriculture. In particular, the company employs domain experts who manage live, evolving information about plants, insects, parasites, phytosanitary products, weather forecasts, and so on. A plethora of web sites and portals, possibly developed by third parties, rely on this information to realize e-services in the agricultural domain.

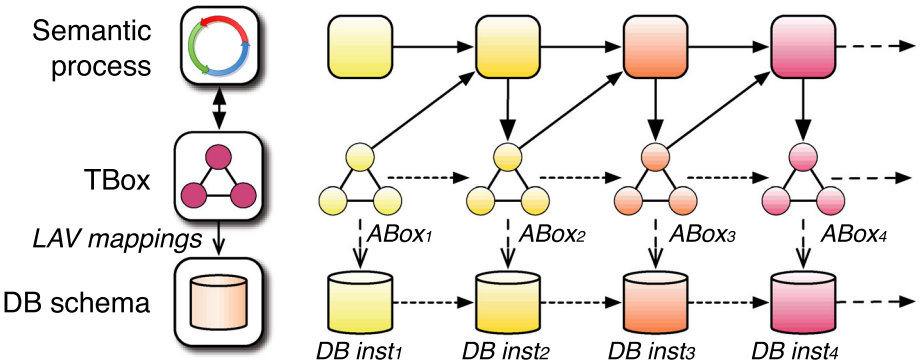
A suitable architecture for the company's information system is one for which a controlled set of back-end business processes with restricted access is used to insert and update the relevant data. On the other hand, the web sites are front-end applications, completely decoupled from the lifecycle of the back-end processes, and relying on their own local database schemas (independent from the "global" schema employed by the back-end). However, they need to access the back-end information system to fetch the relevant data stored there. □

Figure 6 shows how the semantic technologies presented in this work can be combined to support such an architecture, with a twofold advantage: the back-end can manipulate the relevant data at the conceptual level, while seamlessly access and "understand" such data in terms of their local schemas.

More specifically, in Fig. 6 an ontology is used to capture the domain knowledge. Semantic GSM is then employed to construct the back-end processes working on top of the domain ontology, by retaining all the advantages discussed in Sect. 4. At the same time, multiple (external) database schemas are used by the front-end applications. The most critical aspect of the architecture is, therefore, the link between the ontology and such multiple databases. Fortunately, the techniques recalled in Sect. 3 for linking data to ontologies can be exploited to attack this challenging problem. Recall that, in this setting, (part of) the data maintained by the front-end databases must be obtained from the back-end ontology (which is then accessed by front-end databases in read-mode). This suggests that the form of mapping assertions to formally capture the link is the one of LAV mappings. In fact, to specify that a relation R in one of the local, front-end databases, has to be fed with data taken from the ontology, we can devise a mapping assertion of the form $R(\mathbf{x}) \rightsquigarrow \psi(\mathbf{x})$, which actually expresses that $R(\mathbf{x})$ is a (local) view constructed on top of the query $\psi(\mathbf{x})$ posed over the ontology. In other terms, such a LAV assertion describes the content of the relation R in terms of the ontology, which is exactly what we need here. Notice, however, that since in this setting the data flow is from the back-end ontology to the front-end databases, mapping assertions are interpreted as complete rather then sound assertions (cf. Sect. 3). Also, to avoid that local updates on data propagate towards the ontology, we impose that front-end relations mapped to the ontology are only accessible in read mode by local processes (except for the import of data coming from the ontology—cf. below).



(a) Architectural view



(b) Runtime view

Fig. 6 Semantic GSM with LAV mappings exploited by multiple front-end applications

Figure 6 provides an abstract representation of the evolution of data present in the ontology and the local databases at execution time. Every time a (semantic) action is performed, the ABox of the back-end ontology is updated according to the action effects. Through the LAV mapping assertions, this change in the ontology can be also understood by the front-end databases, putting together their own local data with the data present in the ontology. From the operational point of view, this abstract picture can be grounded in the system by exploiting the mapping assertions in two ways:

- effectively *transfer* data extracted from the ontology to the local database.
- answer queries posed over the local database by *transparently accessing* the ontology on-demand.

5.1 Data Transfer

In data transfer approach, some data maintained by the ontology are now replicated in the local database, similarly to data exchange (cf. [39]). The disadvantage of this approach is that it introduces redundancy, and consequently corresponding mechanisms must be implemented to regularly align the data maintained by the local database with the ones present in the ontology. Remember, in fact, that there are back-end processes running on top of the ontology, which could lead to changes that should be propagated to R. On the other hand, the advantage of this approach is that the back-end and the front-end only interact at specific, pre-determined moments in time: a connection between the ontology and the local database is required only when there is an alignment request issued to the local database. Beside these synchronisation points, the two systems operate completely independently from each other.

As an example, let us consider again the e-agricultural company of Example 3. Supposing that the back-end stores fresh forecast data every day before midnight, a front-end application requiring those data can simply trigger an alignment just after midnight, importing the new information into its own local database, then using this local "copy" to provide its specific service, without the need of further interaction with the back-end.

5.2 Transparent Access

With the transparent access approach, the local database does not replicate the data present in the ontology. However, when queries are issued over the local database, mapping assertions are exploited to suitably include in the returned result set also data present in the ontology. From the viewpoint of a front-end application, there is no difference between this approach and the data transfer one, i.e., transparent access constitutes a form of "virtual" data transfer.

While this approach requires a stable, long-running connection between each front-end application and the back-end ontology (making it possible to access the ontology on-demand, every time a query is issued over one of the local databases), it has the advantage that front-end applications always access the fresh, latest data, without incurring in alignment issues.

We point out that the architecture described in this section to link semantic GSM artifacts with a relational storage, independently from the approach adopted (data transfer or transparent access), goes fairly beyond OBDA. Indeed, access to data in our setting is possible both through the (back-end) ontology and through the (front-end) databases. In particular, from the point of view of the front-end databases, the ontology is seen as a data source, but differently from OBDA, where data sources are always plain databases, it is a source with incomplete information. In this respect, both transferring and on-the-fly querying of data turn out to be computationally challenging. A simple, but effective, way to deal with this situation is to assume that in each mapping assertion $R(\mathbf{x}) \rightsquigarrow \psi(\mathbf{x})$, the relation R is a view corresponding to the certain answers of $\psi(\mathbf{x})$ over the ontology. This in fact means to weaken the semantic interpretation of the mapping (w.r.t. the completeness assumption discussed above), and at the same time makes the front-end database rely only on the query answering service exported by the back-end ontology (thus somehow hampering the modularization of the overall system in independent components). We point out that a similar approach has been advocated in the context of peer-to-peer (P2P) information management and integration (cf. [16] and [29]), where analogous computational problems have been faced and various solutions proposed, ranging from the above possible weakening of the mapping, to the devising of topological restrictions in the P2P network and in the languages used in the peer schemas or ontologies (see also [5,15,26,30,36]).

Example 4 Let us now consider again our previous running example and have a closer look at the processes and data managed by the companies which provide monthly report applications to the system operator. Each such company has indeed its own processes, possibly modeled as GSM artifacts, which are executed independently from the processes of the system operator, as well as from the other companies. For these reasons, from the point of view of the control point assessment artifact, such processes are operating in the external environment, and no details on them or on the information schemas they use are needed for the control point assessment to be executed (cf. Figs. 1 and 2). At the same time, databases locally used by various companies can be seen, for the processes they serve, as front-end databases fed from a back-end ontology for what concerns the official measures published by the system operator in a control point assessment. Such a situation resembles exactly those in Fig. 6.

Assume now that a company C wants to store information about measures it sends to the system operator that are not included in the control point assessment. To this aim C maintains locally a relation $R(\mathsf{CP\_ID}, \mathsf{year}, \mathsf{month}, \mathsf{date}, \mathsf{value})$, whose attributes denote, respectively, the identification number of the control point, the year and the month of the control point assessment, the date and the value of the rejected measure. This information can be gathered from the ontology through the mapping assertion $R(id, y, m, d, v) \rightsquigarrow Q_1(id, y, m, d, v)$, where Q_1 is the ontology query in Example 2.

6 Semantic Monitoring and Governance of Relational Artifacts

We discuss now an architectural solution that complements the one discussed in Sect. 5, but is as much common in a typical industrial setting. The operation of a company is typically encapsulated in a plethora of different intra- and inter-organisational processes, each meant to discipline the work of a branch/group/area inside the company, as well the interaction with other related areas and/or external stakeholders. Such processes may have a very different nature (flexible, rigid, unpredictable,…), involve different persons and devices (employees, domain experts, consultants, managers,…), and be partly not under the control of the company, but of third parties (partner companies, customers, sellers, suppliers,…). Furthermore, from the architectural point of view, the data they manipulate are typically scattered around into several (typically relational) data sources with different schemas.

Example 5 Consider a company that maintains a web magazine, accessed by a community of users and containing banners and advertising information for partner companies. Different processes, possibly with different underlying databases, are designed and implemented by the company to accomplish its business objectives. An internal process is executed to feed the web magazine information system with fresh news. A CRM system is used to record information about the partner companies. A related process is followed to negotiate advertising contracts with such companies and to store the banners to be shown on the web. Finally, a set of web processes are executed to let the users register to the magazine and surf the news, at the same time tracking statistical information of banners' views and clicks.

Despite this architectural fragmentation, however, all the processes rely on and concur in the provision of data of interest for the company. In particular, the scattered data sources contribute altogether to provide the extensional information

used by business experts and managers to assess the state of affairs, take strategic decisions, refine the company's goals, and restructure the processes. To understand and communicate such information, a common conceptualization of the domain is needed and is indeed sometimes adopted, typically represented using graphical specification languages such as E-R, UML, or ORM diagrams. Of course, such conceptualization can be naturally captured by a formal ontology.

Since in this case the purpose is to understand data in the data sources through the ontology, i.e., (virtually) transfer data from the source schemas to the conceptual schema, the most natural form of mapping to adopt to interconnect the two layers is the one of GAV. To define a concept N in the ontology in terms of queries posed over the underlying data sources, a set of assertions of the following form may be employed: $\phi_1(x) \rightsquigarrow N(x), \ldots, \phi_n(x) \rightsquigarrow N(x)$. Similarly, to define a role (i.e., a binary relation) P in the ontology, GAV mapping assertions of the following forms may be used: $\phi_1(x, y) \rightsquigarrow P(x, y), \ldots, \phi_n(x, y) \rightsquigarrow P(x, y)$.

Example 6 Consider again our running example on the ACSI energy use case. Assume now that the ontology we have described in Example 2 is used for monitoring at the semantic level the relational artifact described in Example 1. We, therefore, have to specify suitable mappings from the ontology towards the data schema of the relational artifact, which is described in Fig. 1.

We report below some simple GAV mapping assertions that specify how some instances of the ontology can be built starting from the values stored in the underlying data schema[5].

```
SELECT CPAID
FROM CPAs
 ⇝
CPA(CPAID)

SELECT MeasID, CPAID
FROM CPAMeas
 ⇝
publishedIn(MeasID,CPAID)

SELECT MeasID,concat(CPAID,company)
AS AppID
FROM ManMeas
 ⇝
cont(MeasID,AppID)
```

We also notice that in this scenario we can also easily specify mappings towards the data schema of various relational artifacts, possibly run by different electric companies, thus fostering semantic process integration. For example, let us assume that another company uses a different artifact whose data schema contains the following two relational tables to store reports containing claimed measures on a daily basis for a certain control point in a certain month:

R1(MRA_ID, CP_ID, month, year)
R2(MSR_ID, MRA_ID, date, time, value, type).

MRA_ID is the database code (indeed a primary key in R1) assigned to a control point application, CP_ID is the code of the control point, month and year are, respectively, the month and the year the application refers to, MSR_ID is the database code assigned to measures (indeed a primary key in R2), date, time, and value are, respectively, the date, the time, and the value associated with a measure, and type indicates if the measure is taken manually, in this case it has the value 'm', or in an automatic way, in which case it assumes the value 'a'.

We notice that the company, for other own purposes, stores in fact various measures with the same MRA_ID and the same date, all having a different times (i.e., MRA_ID, date, and time form together a key in R2). Only the measure with the greater value for time within a certain date is then communicated to the system operator.

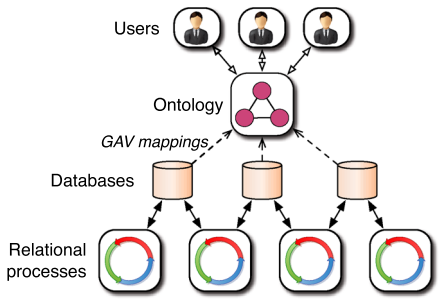
The following SQL query $Q_{SQL}(msr)$ selects only sent measures taken manually:

```
SELECT X1.MSR_ID AS msr FROM R2 AS X1 WHERE
X1.type = 'M' AND not exists (SELECT *
FROM R2 AS X2 WHERE X2.MSR_ID <>
X1.MSR_ID AND X2.MRA_ID = X1.MRA_ID
AND X2.date = X1.date AND X1.time > X2.time)
```

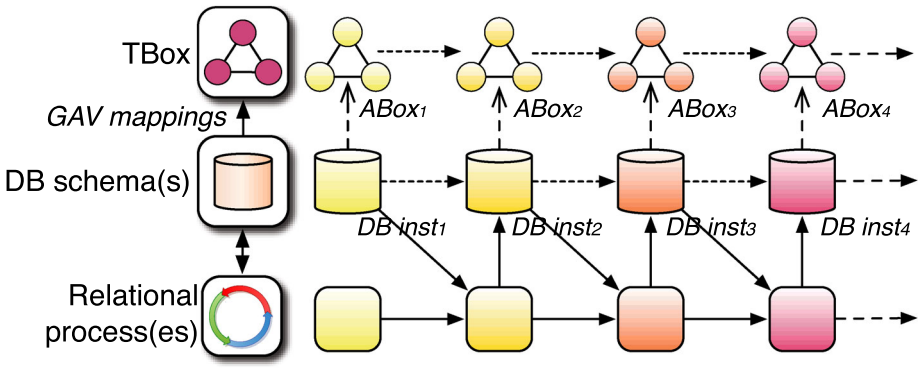
The above query can be then used as database query in a mapping assertion $Q_{SQL}(msr) \rightsquigarrow \mathsf{Man}(msr)$, where Man is the concept denoting manual measures in the ontology given in Fig. 4.

As recalled in Sect. 3, OBDA techniques have been extensively employed to enable the concrete usage of a domain ontology to integrate and access the company's data. We discuss here how these benefits carry over the setting where the underlying data sources are manipulated by artifacts and their corresponding processes. Figure 7 gives an overview of the system architecture that arises in this setting. The difference between a classical OBDA setting is that the underlying data sources are regularly subject to changes due to the running processes. This can be appreciated by considering Fig. 7, which provides an abstract representation of the system evolution. Ideally, every action execution at the relational level triggers a change in at least one of the data sources. Through the mapping assertions, this translates into a corresponding change in the (extensional knowledge of the) ontology (the ABox). The new, resulting snapshot can then be queried at the conceptual level through the ontology itself.

[5] In the third assertion we construct the identifiers of instances of MntRep by concatenating CPAID and company.



(a) Architectural view



(b) Runtime view

Fig. 7 Relational processes with GAV mappings and a unifying ontology

Analogously to what we did in Sect. 5, this abstract picture can be concretely instantiated in two ways: by applying an effective data transfer from the data sources to the ontology, or by exploiting the ontology to access the underlying relational data on-demand.

6.1 Data Transfer

In the data transfer scenario, data are effectively migrated from the underlying data sources to the ontology. Since GAV mapping assertions are sound w.r.t. the data sources, we can in this case effectively materialize the ABox of the ontology by simply evaluating the queries used in the left-hand side of the corresponding assertions and populating the ABox with the union of the obtained result sets.For example, for the aforementioned concept N, its population can be obtained as the answer of the query $\bigvee_{i\in\{1,\ldots,n\}} \varphi(x)$ (similarly for roles).

This approach resembles the one of data warehousing, though in this case the central repository is constituted by a rich, conceptual model. Business managers and analysts can in fact exploit the ontology to query the obtained integrated data at a high level of abstraction and by exploiting a "business-level" vocabulary. This, in turn, provides the basis for reporting and analysis.

The data transfer approach can also be fruitfully exploited to support external audits. In fact, part of an audit is typically dedicated to analyze (a portion of) the real data maintained by the company's information system, to check compliance with regulations and best practices. Obviously, this analysis can be facilitated if, instead of directly accessing the data sources with their heterogeneous schemas, compliance queries are expressed in terms of the ontology.

Another important application of the data transfer approach is in process mining [1].See the discussion about the "semantic event log", provided below.

6.2 Transparent Access

Transparent, on-demand access to the concrete data sources through the ontology can be obtained by relying on the classical framework of OBDA. Here, to achieve transparency, no data are materialized in the ABox, but query answering is realized through query rewriting, which reformulates the user query into a new query expressed in the alphabet of the sources, whose evaluation over the source database returns the certain answer of the user query. Such reformulation takes into account the TBox ontology and the mappings. [50] and [51] provide notable examples in which the above rewritings can be expressed in SQL, thus allowing for delegating its evaluation to the underlying relational data management systems.

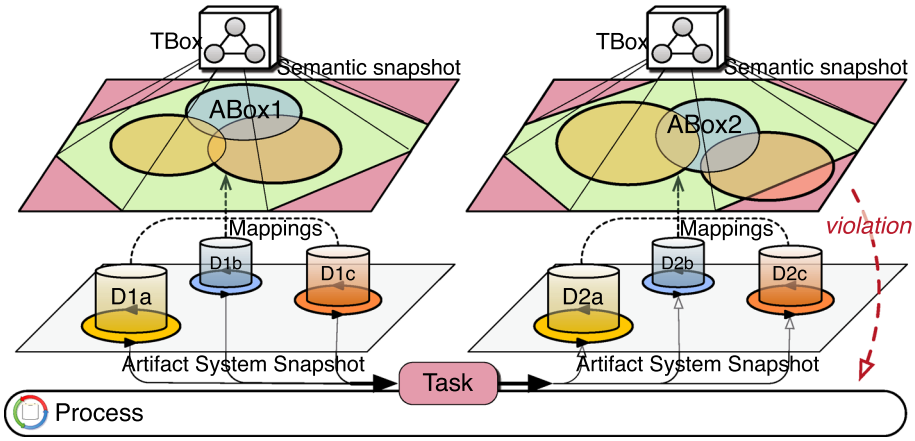


Fig. 8 Ontology-based governance of a relational artifact system

The described framework can be used in our setting to obtain meaningful information about the current state of affairs, reached as a result of the execution of (possibly multiple) business artifacts working over the relational sources and the corresponding processes. Understanding the semantics of data contained in this low-level sources is important to

- Conceptual query answering, with the same advantages discussed in Sect. 4. In particular, if the underlying relational processes are specified in terms of GSM artifact lifecycles, then part of the ontology can be dedicated to capture GSM itself, as shown, e.g., in Figs. 3 and 5. Concepts and relationships used to model the lifecycle can be easily mapped to the status attributes maintained by the underlying GSM information schemas through suitable GAV mapping assertions, supporting the possibility of flexibly pose conceptual queries asking about the current process status, possibly relating it also to the current data, as shown in Example 2.
- Govern the underlying processes, blocking the finalization of those process actions that manipulate the data leading to a globally inconsistent situation, where some semantic constraint in the ontology is violated (see Fig. 8). Obviously, this requires a mechanism to evaluate the action effects before effectively enforcing them, triggering an exceptional behavior if a violation is detected. In particular, this must be propagated down to the process responsible of the action, which in turn can activate a compensation phase, finding an alternative execution path. To show how this approach can be applied also with classical process specifications, Fig. 9 sketches the meta-model of a BPMN task execution that exploits a transaction to coordinate with the ontology governance service, triggering a roll-back (and a corresponding compensation sub-process) in the case of non-conformance.
- Relate different artifacts that share information, though possibly with very different representation, in their artifact instances. This is even more critical in the case of inter-organizational processes combining artifacts of multiple companies.
- Discipline the introduction of new artifacts and processes in the system, checking whether they seamlessly integrate with the already existing artifacts and processes, and supporting various forms of conformance tests.
- Facilitate the realization and enforcement of authorization views, proposed by [42] in the context of so-called artifact interoperation hubs (see [37]), to formally regulate to which pieces of information the various stakeholders participating to the hub share an access.

An example of a real-world application in which we adopted the framework for semantic monitoring of relational artifact discussed in this section is given in Sect. 7.

6.3 Semantic Event Log

We discuss now a particular scenario in which the approach presented in this section is exploited to provide the basis for process analysis, improvement, and re-engineering. In particular, we sketch how the combination of ontology and mapping assertions from the process data sources to the ontology can be used as a basis for process mining [1].

Process mining combines business process analysis with data mining, to the aim of discovering, monitoring, diagnosing, and ultimately improving business processes. Traditionally, process mining is applied to post-mortem data, i.e., data related to already completed process instances. Recently, its applicability has been broadened including also a plethora of operational decision support tasks that are exploited at run-time, i.e., by considering live data of running process instances.

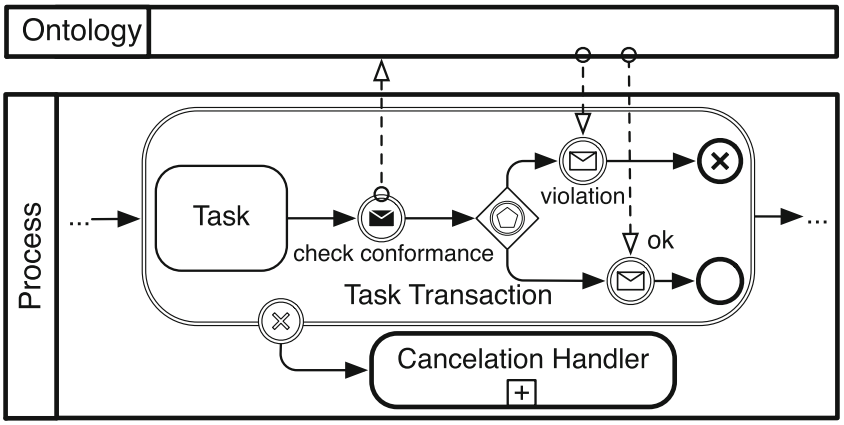


Fig. 9 Meta-model of an ontology-governed BPMN task

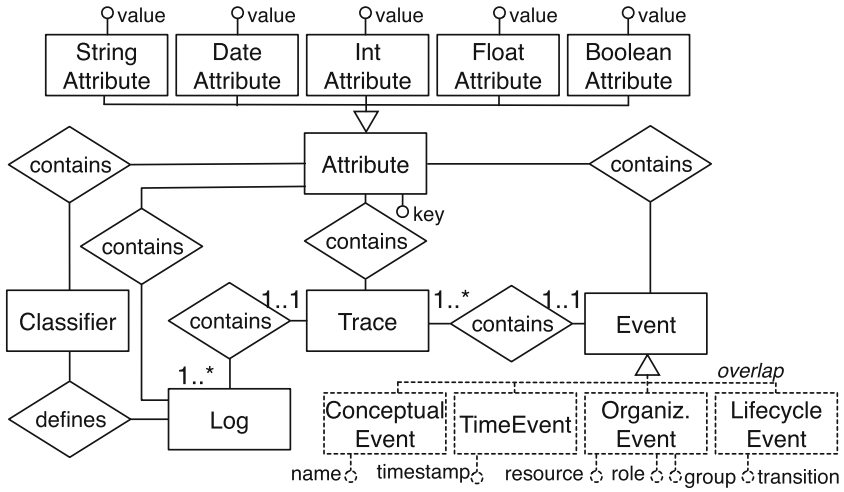


Fig. 10 E–R diagram capturing a portion of the XES meta-model; the dashed part is reported for clarity, but is concretely realized in XES through the notion of extension and corresponding required attributes for the events

Independently from the phase in which process mining techniques are exploited, central for their applicability is the availability of process *event logs*, which explicitly trace all the relevant events (and the corresponding data) occurred so far due to the process execution. The availability of event logs of good quality poses a twofold challenge. On the one hand, the meaningful information associated with the events is typically scattered around different tables in the underlying database and possibly even in several data sources. On the other hand, standard formats for event logs, such as XES (http://www.xes-standard.org/), have been proposed to make it possible to apply process mining algorithms and tools without the need of customizing how they are fed with input data on a per-company basis.

For these reasons, the extraction of a unique, standard event log from a company's information system is far from trivial. In this respect, OBDA can be effectively applied to

- include in the ontology a set of concepts and relationships dedicated to capture event logs according to the chosen format representation standard (see, e.g., Fig. 10).
- Establish mapping assertions from the data sources to this portion of the ontology, in such a way to be able to understand event-related data in terms of the standard representation.

In this setting, the data transfer scenario can be exploited to extract an event log containing post-mortem data and to apply process mining techniques off-line. Conversely, the on-demand access approach can be used for monitoring purpose, writing compliance queries on top of the event log, and consequently checking whether a process execution trace is currently respecting or violating certain business rules, in the style of [22,45].

7 Instantiation of the Framework for Semantic Monitoring in the ACSI Project

In this section we briefly describe how the framework proposed in this paper has been instantiated within the European Project ACSI (Artifact-Centric Service Integration), for semantic governance and verification of artifact systems.

We point out that such instantiation has been concretely applied within a real-world use case, called the energy use case, concerning the context of Spanish electric supply system and focused on the energy exchange between different electric companies, controlled by a central system operator. An excerpt of such a scenario has been already presented throughout the paper in the form of a running example. To describe semantic governance of artifact systems in ACSI, we continue to refer to such a scenario.

Within ACSI, the processes of interest in the energy use case were extensively modeled in terms of (classical) GSM artifacts. Example 1 describes one of them, namely CPA artifact. Other artifacts that have been realized refer to submission of possible objections issued by a company concerning the measures published by the system operator and to the final liquidation process related to the energy exchange. The realized artifacts were effectively deployed in the so-called *ACSI Interoperation Hub*, an environment for the execution of independent GSM artifacts operating for a common goal.

In a parallel activity, we designed an ontology for the energy setting, by leveraging on the upper level ontology given in Fig. 3, and by specializing it into a specific ontology representing the domain of interest for the energy use case, similarly to what is described in Example 2. We specified the resulting overall ontology in *DL-Lite*. A graphical representation of a portion of this ontology is given in Figs. 4 and 5. We point out that some properties represented in such figures, non-expressible in native way in *DL-Lite*, such as the covering of the concept Meas, or the maximum cardinality 2 on roles basedOn, connTo and cont, have been suitably approximated by means of epistemic constraints (cf. Sect. 4). On the other hand, we fully exploited the expressive power of *DL-Lite*, which also allows for specifying additional constraints that are not rendered graphically in the form of ER-constructs, such as the following identification assertions:

- no two applications based on the same CPA are edited by the same company (i.e., App is identified by its participation in the roles editedBy and basedOn);
- it is not possible for two CPAs to have the same year–month and refer to the same control point.

In a second phase, we connected the artifact and semantic layers by linking the data schema of the underlying GSM artifacts to the concepts and relations of the ontology, for governance and monitoring purposes. In accordance with the framework described in Sect. 6, the linkage of the ontology towards the underlying data schema was realized through GAV mapping assertions. Each such assertion associates an element of the ontology with an SQL query over the data schema of the artifact layer (see Example 6). This allowed a transparent access to the data stored in the artifact layer, by fetching information only on-demand, without performing any (costly) materialization of the data maintained by the artifacts in terms of an ABox instantiating the TBox of the ontology.

Some prototype tools have been developed within ACSI for the management and exploitation of the semantic layer. One such a tool has the purpose of maintaining and documenting the ontology, of storing and inspecting the mappings and, most notably, of providing support for querying the ontology. A snapshot of the querying environment provided by this tool is given in Fig. 11. We stress that the system we realized is coupled with an OBDA reasoner featuring a sound and complete technique for query answering over ontologies. Specifically, within ACSI we employed Mastro [23], an OBDA reasoner able to process unions of conjunctive queries expressed in SPARQL syntax, and posed over OBDA specifications where the TBox is expressed in *DL-Lite*, and the mappings are GAV. Mastro is also able to process ECQ queries specified in a proprietary syntax. To process queries, Mastro proceeds in two steps. In the first step, called *ontology rewriting*, it computes the FOL-rewriting of the input query with respect to the TBox of the OBDA specification. In the second step, called *mapping rewriting*, it further rewrites the FOL-rewriting obtained in the previous step by considering the contribution of the mappings, following a classical unfolding procedure [41]. Intuitively, this procedure substitutes each ontology predicate in the query with the SQL view that the mapping associates to it. Notably, the final rewriting is expressed into a standard SQL query that can be directly processed at the artifact layer, i.e., OBDA specifications managed by Mastro support FOL-rewritable query answering (cf. Sect. 3).

We effectively exploited this tool for monitoring artifact systems by issuing queries over the ontology. Through such queries, we were able to understand at the conceptual level the impact of the evolution of the artifact system in terms of the managed information and govern it. For example, we had the possibility of relying on specific queries so as to identify execution steps which, once executed at the artifact layer, would lead to a new (virtual) semantic snapshot that is inconsistent with the ontology TBox.

An example of monitoring query is given in the following:

$$\exists\, rep, ms_1, ms_2, d.[\mathsf{MntRep}(rep) \wedge \mathsf{cont}(rep, ms_1) \wedge \mathsf{cont}(rep, ms_2) \wedge \mathsf{date}(ms_1, d) \wedge \mathsf{date}(ms_2, d)] \wedge ms_1 \neq ms_2$$

This query is an ECQ asking for the existence of a monthly report that contains two distinct measures referring to the same date. This is an unwanted situation, since every report has to exhibit only a single measure per day[6].

[6] Notice that the above query in fact constitutes an epistemic constraints specified over the TBox.

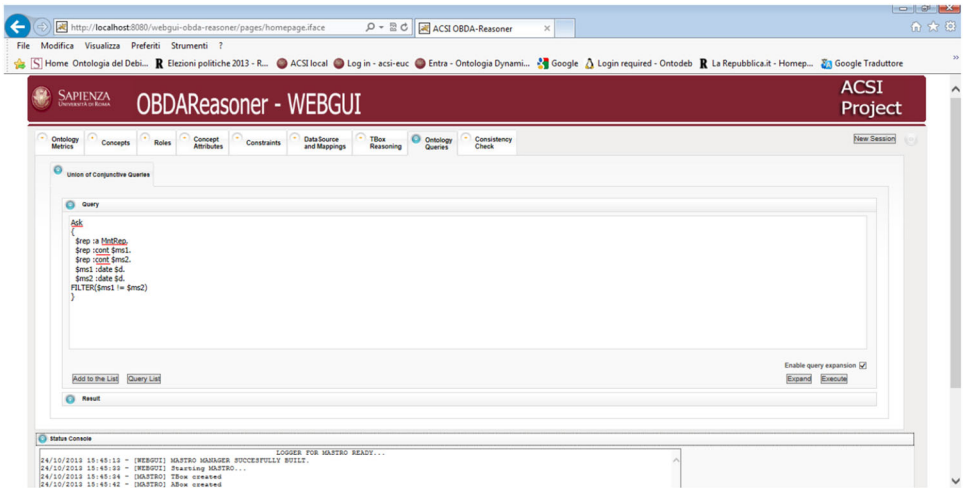


Fig. 11 Semantic artifact monitoring ACSI tool

To show the benefit of writing queries at the semantic level, we briefly discuss how Mastro processes the aforementioned query, and how the final SQL rewriting looks like. According to the semantics of ECQs, answering the above query amounts to first compute the certain answers of the CQ $Q = \mathsf{MntRep}(rep) \wedge \mathsf{cont}(rep, ms_1) \wedge \mathsf{cont}(rep, ms_2) \wedge \mathsf{date}(ms_1, d) \wedge \mathsf{date}(ms_2, d)$ and then verify the inequality $ms_1 \neq ms_2$ over such certain answers. To this aim, Mastro first computes the ontology rewriting Q_o, which is the FOL query (in fact a UCQ with inequalities) partially reported below:

$$\exists\, rep, ms_1, ms_2, d.((\mathsf{App}(rep) \wedge \mathsf{cont}(rep, ms_1) \wedge \mathsf{cont}(rep, ms_2) \wedge \mathsf{date}(ms_1, d) \wedge \mathsf{date}(ms_2, d)) \vee (\mathsf{CPA}(rep) \wedge \mathsf{publishedIn}(rep, ms_1) \wedge \mathsf{publishedIn}(rep, ms_2) \wedge \mathsf{date}(ms_1, d) \wedge \mathsf{date}(ms_2, d)) \vee \ldots) \wedge ms_1 \neq ms_2$$

We notice that Q_o is obtained by rewriting the conjunctive query Q and then imposing the inequality $ms_1 \neq ms_2$ over the rewriting. Intuitively, to rewrite Q Mastro exploits inclusions between concepts and roles, typings of roles, and mandatory participations of concepts into roles asserted in the ontology (for further details we refer the reader to [18]). In the above example, since App is a sub-concept of MntRep, and publishedIn is a sub-role of cont, the rewriting algorithm substitutes the predicates MntRep and cont in Q with App and publishedIn, respectively, generating all disjuncts the stem from all possible substitutions. In total, Mastro generates an overall number of approximately 50 disjuncts. To finalize the FOL-rewriting, Mastro then computes the mapping rewriting of Q_o by unfolding it according to the mapping assertions (which we partially described in Example 4). This produces a final number of disjuncts which is in general exponential w.r.t. the number of atoms in Q_o. We notice that the ontology rewriting phase and, therefore, the reasoning over the ontology, turned out to be crucial in this query: unfolding directly the original query would have returned an empty query, since MntRep and cont are in fact not directly associated with any query in the mappings.

We close this section by discussing the impact of choosing *DL-Lite* as the ontology language for expressing the dynamics of an artifact-centric system. The choice of *DL-Lite* is supported by the fact that the same benefits of the FOL-rewritability property for query answering and ODBA also apply to the system dynamics. Such benefits hold not only when the dynamics of the artifact system is directly specified at the semantic level, as in the case of semantic GSM (cf. Sect. 4), but also when the dynamics is specified at a lower level, which is relational in the majority of cases (as in standard GSM), and declarative dynamic constraints are expressed over the semantic layer posed on top of the relational one. In the first setting, the semantic GSM specification of an artifact can be automatically manipulated by technologies like Mastro to be translated into a corresponding standard GSM specification (by simply rewriting each sentry). The resulting specification can then be directly executed using the ACSI Interoperation Hub (or any execution engine for relational GSM artifacts). In the second setting, temporal/dynamic properties are added to the semantic layer, so as to express conceptual constraints on the expected or forbidden evolutions of concepts and relations. Within ACSI, we used in particular a first-order variant of μ-calculus, virtually the most powerful logic for verification, to express such dynamic constraints. In this variant, local properties are in fact ECQs over the alphabet of the ontology TBox. Notably, [20] show that, by virtue of the FOL-rewritability, the rewriting of dynamic constraints (taking into account both TBox assertions and GAV mappings) can be directly posed over the data schema of relational artifacts. This result

has been effectively implemented in the OBGSM tool [10] and it is applicable in the setting described in Sect. 6, that is, when relational GSM artifacts are linked to a (*DL-Lite*) ontology through GAV mappings. This tool implements the reformulation technique described by [20] and returns a corresponding temporal formula which, together with the specification of GSM artifacts, can be directly fed into the GSMC model checker proposed by [33] for verification of GSM artifacts.

In summary, the key property of FOL-rewritability enjoyed by *DL-Lite* makes it possible to start from a rich semantic artifact framework and reformulate it so as to use traditional ontology-agnostic tools for query answering, execution, and verification of artifact systems.

8 Related Work

Since the introduction of artifact-centric processes as a means to integrate processes and data [48], extensive research has been done along two main lines: foundational approaches for formally representing and verifying artifact-centric systems (such as, e.g., [11,27]), or proposals of modeling languages and corresponding execution environments (such as, e.g., [38,54]).

The majority of approaches related to artifact-centric systems assumes that artifacts maintain relational data. Furthermore, as far as we know only a few works exploit the artifact information model beyond just its use to support the artifact execution. An example is [54], where a unifying view encompassing two different artifact modeling languages (GSM and EZ-flow) is used to wrap the execution of hybrid artifact-centric systems, implemented partly in GSM and partly in EZ-flow. The reconciliation between the underlying information models and the common, abstract one is done by exploiting very simple mapping assertions and could be, therefore, accommodated by the framework here presented.

As for the verification of artifact-centric (or, more in general, data-centric) business processes, the same trend discussed above can be seen: the majority of proposed frameworks relies on the relational model for the data component [9,11,27]. A notable exception is constituted by the so-called DL Knowledge and Action Bases (KABs, see [8]). KABs are constituted by a knowledge component modeled by means of a data-oriented DL ontology called *DL-Lite*and by an action component containing a set of actions (and a process built on top of them) used to progress the knowledge component. In this respect, the work here presented can be considered as a concretization of the KAB framework, where the action component is grounded in the GSM language. Verification of (classical) GSM artifacts has been tackled only very recently, with some preliminary but promising theoretical results [12,53] and practical developments [33]. One could wonder whether verification of GSM could be extended to the semantic GSM presented here. Thanks to [53], we have, in principle, an affirmative answer. In fact, to show verifiability of GSM artifacts, [53] presents a translation mechanism into a formal framework whose action component closely corresponds to the one of KABs. Since the translation is orthogonal w.r.t. the data component, it can be seamlessly applied to obtain a KAB starting from a semantic GSM specification, consequently enabling the possibility of applying the verification results obtained for KABs.

In [20], a framework for the verification of data-centric processes is proposed, which mixes a formal representation of relational artifacts with the notion of ontology as considered here. Differently from KABs, processes progress their execution at the relational layer, and the ontology is used to understand and govern the execution at a higher level of abstraction. The link between the underlying relational information models and the unified conceptual representation provided by the ontology is established through mapping assertions, i.e., relying to OBDA. This work exactly provides the formal underpinning for the governance and monitoring architectural framework presented in Sect. 6, showing that this two-level architecture can be subject to verification. In particular, it recasts the query unfolding approach typically used in *DL-Lite*OBDA to the more general case of temporal properties composed by temporal operators combined with queries over the ontology. Such a result constituted the basis for the verification framework described in [10] (cf. Sect. 7).

9 Discussion and Conclusion

In this paper we have provided a comprehensive framework for semantic GSM artifacts. The key characteristic of the framework is that it allows for expressing both the data and the lifecycle schema of GSM artifacts in terms of an ontology. We have provided an upper ontology for the semantic GSM model, which is specialized in each artifact with specific lifecycle elements, relations, and business objects. We have then enriched our framework by enabling the linkage of the ontology to autonomous database systems through the use of mapping assertions, as done in data integration and OBDA. We have discussed two main scenarios of practical interest where the use of mappings turn out to be crucial for enabling collaboration and communication among different and heterogeneous systems. We have discussed a concrete instantiation of our framework based on the use of *DL-Lite* ontologies and have shown a real-world application of it in the energy domain, investigated in the context of the EU project ACSI. Notably, FOL-rewritability of query answering enjoyed by *DL-Lite* ontologies allowed us to rely in the practice on the same artifact management tools

based on relational technology used for classical GSM artifacts. We leave to future studies the investigation of those cases where ontology languages are more expressive than *DL-Lite*. However, from the computational point of view we can already notice that in these more expressive settings a purely intensional approach based on query rewriting is difficult to achieve. Indeed, as shown in [21], *DL-Lite* is one of the maximal ontology languages enjoying FOL-rewritability of query answering, and even limited extensions of it lead to inherently more complex query answering, which can, therefore, not be encoded into evaluation of a FOL query (which can be directly translated into SQL and, therefore, managed under relational technology). We argue that in these cases, approaches (even partially) based on some data manipulation should be pursued, to achieve reasonable performances.

A natural follow-up of this paper is to study, from both the practical and the theoretical perspective, the scenario that results from the merging of the systems described in Sect. 5 and in Sect. 6. In this scenario, global semantic GSM artifacts, operating over the ontology, and relational GSM artifacts, operating over autonomous databases, coexist, and act both as consumers and producers of information. Therefore, all forms of mapping assertions and semantic assumptions on them considered in this paper need to be adopted, to guarantee both semantic governance of relational artifact and access to ontology data by front-end systems. On the one hand, this scenario sets the stage for advanced and completely distributed semantic reasoning over artifacts, and on the other, it presents challenging computational issues. For example, data manipulation at the ontology level in this setting is closely related to the longstanding problem of view updates [56].

Acknowledgments This work has been supported by the EU FP7 project ACSI (Grant No. 257593) and by the FP7 large-scale integrating project Optique (Grant No. 318338).

References

1. van der Aalst WMP, et al (2011) Process mining manifesto. In: Proceedings of BPM Workshops, LNBIP. Springer, New York, pp 169–194
2. van der Aalst WMP, Barthelmess P, Ellis CA, Wainer J (2001) Proclets: a framework for lightweight interacting workflow processes. Int J Coop Inf Syst 10(4):443–481
3. Abiteboul S, Hull R, Vianu V (1995) Foundations of databases. Addison-Wesley, UK
4. Abiteboul S, Bourhis P, Galland A, Marinoiu B (2009) The AXML artifact model. In: Proceedings of TIME 2009, pp 11–17
5. Adjiman P, Chatalic P, Goasdoué F, Rousset MC, Simon L (2006) Distributed reasoning in a peer-to-peer setting: application to the Semantic Web. J Artif Intell Res 25:269–314
6. Apt KR, Blair HA, Walker A (1988) Towards a theory of declarative knowledge. In: Foundations of deductive databases and logic programming. Morgan Kaufmann, Massachusetts, pp 89–148
7. Baader F, Calvanese D, McGuinness D, Nardi D, Patel-Schneider PF (eds) (2007) The description logic handbook: theory, implementation and applications, 2nd edn. Cambridge University Press, Cambridge
8. Bagheri Hariri B, Calvanese D, De Giacomo G, De Masellis R, Felli P, Montali M (2013a) Description logic knowledge and action bases. J Artif Intell Res 46:651–689
9. Bagheri Hariri B, Calvanese D, De Giacomo G, Deutsch A, Montali M (2013b) Verification of relational data-centric dynamic systems with external services. In: Proceedings of PODS, pp 163–174
10. Bagheri Hariri B, Calvanese D, Montali M, Santoso A, Solomakhin D (2013c) Verification of semantically-enhanced artifact systems. In: Proceedings of ICSOC, LNCS. Springer, New York
11. Belardinelli F, Lomuscio A, Patrizi F (2012a) An abstraction technique for the verification of artifact-centric systems. In: Proceedings of KR, pp 319–328
12. Belardinelli F, Lomuscio A, Patrizi F (2012b) Verification of GSM-based artifact-centric systems through finite abstraction. In: Proceedings of ICSOC, LNCS. Springer, New York, pp 17–31
13. Berardi D, Calvanese D, De Giacomo G (2005) Reasoning on UML class diagrams. Artif Intell 168(1–2):70–118
14. Calì A, Gottlob G, Lukasiewicz T (2009) A general datalog-based framework for tractable query answering over ontologies. In: Proceedings of PODS, pp 77–86
15. Calvanese D, De Giacomo G, Lembo D, Lenzerini M, Rosati R (2004a) What to ask to a peer: ontology-based query reformulation. In: Proceedings of KR, pp 469–478
16. Calvanese D, De Giacomo G, Lenzerini M, Rosati R (2004b) Logical foundations of peer-to-peer data integration. In: Proceedings of PODS, pp 241–251
17. Calvanese D, De Giacomo G, Lembo D, Lenzerini M, Rosati R (2007a) EQL-Lite: Effective first-order query processing in description logics. In: Proceedings of IJCAI, pp 274–279
18. Calvanese D, De Giacomo G, Lembo D, Lenzerini M, Rosati R (2007b) Tractable reasoning and efficient query answering in description logics: the DL-Lite family. J Autom Reason 39(3):385–429
19. Calvanese D, De Giacomo G, Lembo D, Lenzerini M, Poggi A, Rodriguez-Muro M, Rosati R, Ruzzi M, Savo DF (2011) The Mastro system for ontology-based data access. Semant Web J 2(1):43–53
20. Calvanese D, De Giacomo G, Lembo D, Montali M, Santoso A (2012) Ontology-based governance of data-aware processes. In: Proceedings of RR, LNCS, vol 7497. Springer, New York, pp 25–41
21. Calvanese D, De Giacomo G, Lembo D, Lenzerini M, Rosati R (2013) Data complexity of query answering in description logics. Artif Intell 195:335–360
22. Chesani F, Mello P, Montali M, Riguzzi F, Sebastianis M, Storari S (2009) Checking compliance of execution traces to business rules. In: Proceedings of BPM Workshops, LNBIP, vol 17. Springer, New York, pp 134–145
23. Civili C, Console M, De Giacomo G, Lembo D, Lenzerini M, Lepore L, Mancini R, Poggi A, Rosati R, Ruzzi M, Santarelli V, Savo DF (2013) MASTRO STUDIO: managing ontology-based data access applications. PVLDB 12:1314–1317
24. Cohn D, Hull R (2009) Business artifacts: a data-centric approach to modeling business operations and processes. IEEE Bull Data Eng 32(3):3–9
25. Damaggio E, Hull R, Vaculín R (2011) On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. In: Proceedings of BPM, LNCS. Springer, New York, pp 396–412

26. De Giacomo G, Lembo D, Lenzerini M, Rosati R (2007) On reconciling data exchange, data integration, and peer data management. In: Proceedings of PODS, pp 133–142
27. Deutsch A, Hull R, Patrizi F, Vianu V (2009) Automatic verification of data-centric business processes. In: Proceedings of ICDT, ACM, pp 252–267
28. Doan A, Halevy AY, Ives ZG (2012) Principles of data integration. Morgan Kaufmann, Massachusetts
29. Franconi E, Kuper G, Lopatenko A, Serafini L (2003) A robust logical and computational characterisation of peer-to-peer database systems. In: Proceedings of DBISP2P, pp 64–76
30. Fuxman A, Kolaitis PG, Miller RJ, Tan WC (2005) Peer data exchange. ACM Trans Database Syst 31(4):1454–1498
31. Gelder AV (1989) Negation as failure using tight derivations for general logic programs. J Log Program 6(1&2):109–133
32. Glimm B, Horrocks I, Lutz C, Sattler U (2008) Conjunctive query answering for the description logic SHIQ. J Artif Intell Res 31:151–198
33. Gonzalez P, Griesmayer A, Lomuscio A (2012) Verifying GSM-based business artifacts. In: Proceedings of ICWS, IEEE, pp 25–32
34. Gruber TR (1993) Formal ontology in conceptual analysis and knowledge representation. In: Poli R, Guarino N (eds) Towards principles for the design of ontologies used for knowledge sharing. Kluwer Academic Publishers, New York
35. Haarslev V, Möller R (2001) RACER system description. In: Proceedings of IJCAR, LNAI, vol 2083. Springer, pp 701–705
36. Halevy A, Ives Z, Suciu D, Tatarinov I (2003) Schema mediation in peer data management systems. In: Proceedings of ICDE, pp 505–516
37. Hull R, Narendra NC, Nigam A (2009) Facilitating workflow interoperation using artifact-centric hubs. In: Proceedings of ICSOC, LNCS, vol 5900. Springer, New york, pp 1–18
38. Hull R, Damaggio E, De Masellis R, Fournier F, Gupta M, Heath FT III, Hobson S, Linehan M, Maradugu S, Nigam A, Sukaviriya PN, Vaculin R (2011) Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In: Proceedings of DEBS, ACM, pp 51–62
39. Kolaitis PG (2005) Schema mappings, data exchange, and metadata management. In: Proceedings of PODS, pp 61–75
40. Kontchakov R, Lutz C, Toman D, Wolter F, Zakharyaschev M (2010) The combined approach to query answering in DL-Lite. In: Proceedings of KR, pp 247–257
41. Lenzerini M (2002) Data integration: a theoretical perspective. In: Proceedings of PODS, pp 233–246
42. Limonad L, Boaz D, Hull R, Vaculín R, Heath F (2012) A generic business artifacts based authorization framework for cross-enterprise collaboration. In: Proceedings of SRII, pp 70–79
43. Linehan M (2011) GSM expression language. Technical report, IBM Research, available on request
44. Meyer A, Smirnov S, Weske M (2011) Data in business processes. EMISA. Forum 31(3):5–31
45. Montali M, Maggi FM, Chesani F, Mello P, van der Aalst WMP (2013) Monitoring business constraints with the event calculus. ACM Trans Intell Syst Technol 5(1):17
46. Motik B, Fokoue A, Horrocks I, Wu Z, Lutz C, Cuenca Grau B (2009) OWL 2 web ontology language profiles. W3C Recommendation, World Wide Web Consortium. Available at http://www.w3.org/TR/owl-profiles/
47. Motik B, Cuenca Grau B, Horrocks I, Wu Z, Fokoue A, Lutz C (2012) OWL 2 Web Ontology Language profiles, 2nd edn. W3C recommendation, World Wide Web Consortium. Available at http://www.w3.org/TR/owl2-profiles/
48. Nigam A, Caswell NS (2003) Business artifacts: an approach to operational specification. IBM Syst J 42(3):428–445
49. Object Management Group (OMG) (2013) Case management model and notation vers. 1.0 - Beta 1. http://www.omg.org/spec/CMMN/1.0/Beta1/PDF/
50. Poggi A, Lembo D, Calvanese D, De Giacomo G, Lenzerini M, Rosati R (2008) Linking data to ontologies. J Data Semant X:133–173
51. Rodriguez-Muro M, Calvanese D (2012) High performance query answering over DL-Lite ontologies. In: Proceedings of KR, pp 308–318
52. Sirin E, Parsia B, Kalyanpur A, Katz Y (2007) Pellet: a practical OWL-DL reasoner. J Web Semant 5(2):51–53
53. Solomakhin D, Montali M, De Masellis R, Tessaris S (2013) Verification of artifact-centric systems: decidability and modeling issues. In: ICSOC, pp 252–266
54. Sun Y, Xu W, Su J, Yang J (2012) Sega: A mediator for artifact-centric business processes. In: Proceedings of of CoopIS, LNCS, vol 7567. Springer, New York, pp 658–661
55. Tsarkov D, Horrocks I (2006) FaCT++ description logic reasoner: system description. In: Proceedings of IJCAR, pp 292–297
56. Winslett M (1988) A model-based approach to updating databases with incomplete information. ACM Trans Database Syst 13(2):167–196