ACSI – Artifact-Centric Service Interoperation

Deliverable D2.4.2

Techniques and Tools for KAB, to Manage Action Linkage with Artifact Layer - Iteration 2

Project Acronym	ACSI	
Project Title	Artifact-Centric Service Interoperation	
Project Number	257593	
Workpackage	WP2 – Formal Based Techniques and Tools	
Lead Beneficiary	FUB	
Editor(s)	Diego Calvanese	FUB
	Babak Bagheri Hariri	FUB
	Riccardo De Masellis	UNIROMA1
	Domenico Lembo	UNIROMA1
	Marco Montali	FUB
	Ario Santoso	FUB
	Dimitry Solomakhin	FUB
	Sergio Tessaris	FUB
Contributor(s)	Giuseppe De Giacomo	UNIROMA1
	Alessio Lomuscio	Imperial
	Fabio Patrizi	UNIROMA1
Reviewer(s)	Lior Limonad	IBM Haifa
Dissemination Level	PU	
Contractual Delivery Date	28/02/2013	
Actual Delivery Date	31/05/2013	
Version	1.3	

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement no. 257593.

Abstract

This document reports on the activities related to Task 2.4: *Techniques and Tools for KAB, to Manage Action Linkage with Artifact Layer*, which is part of Workpackage 2 within the ACSI project. In particular, it describes the results of the second iteration of Task T2.4.

The main goal of this activity is to tackle the key issues for actually realizing the Knowledge and Action Base (KAB), by marrying the descriptions of the static and the dynamic aspects of the domain of interest. We start from the logic-based frameworks and the decidability results for KAB verification achieved during the first iteration, and link them to concrete languages, techniques and settings. We apply the discussed technique to the ACSI Energy use case, showing the benefits of adding a Semantic Layer on top of the GSM-based artifacts used to handle energy control points and the corresponding monthly reports.

Document History

Version	Date	Comments
V1.1	15/02/2013	First draft
V1.2	12/04/2011	Second draft
V1.3	31/05/2013	Final version

Table of Contents

List of Figures

1 Introduction

This document reports on the activities related to Deliverable 2.4.2: *Techniques and Tools for KAB, to Manage Action Linkage with Artifact Layer*, which is part of Work Package 2 within the ACSI project. In particular, it describes the results of the second iteration of Task T2.4, leveraging on the ones presented at the end of the first iteration.

The main goal of this activity is to tackle the key issues for actually realizing the Knowledge and Action Base (KAB), by marrying the descriptions of the static and the dynamic aspects of the domain of interest. In particular, we are interested in capturing artifact-centric systems by placing different hypothesis on how data are captured, and how the system dynamics interacts with data. In particular, we consider three main scenarios:

1. artifact systems where data are modeled with complete information, by resorting to relational databases;

2. "semantic" artifact systems where data are modeled under incomplete information, by resorting to full-fledged (description logic) ontologies;

3. a combination of the two approaches, where artifacts are equipped with a relational information model, but a Semantic Layer is added on top of them in order to provide a high-level, conceptual understanding of the domain of interest.

In the first iteration, we have dealt with logic-based frameworks to attack these different scenarios, with the main goal of making artifact-systems amenable to formal verification (model checking against variants of first-order temporal properties). This is far from trivial: differently from traditional model checking, the transition system representing the execution semantics of an artifact-centric system has infinite states, and verification turns out to be highly undecidable in general, even when considering simple propositional temporal properties.

In this second iteration, starting from these logic-based frameworks together and their decidability results (described in Part I), we link them to concrete languages, techniques and settings. In particular, we show how they can be employed to:

- provide decidability results related to the verification of artifact-systems based on the Guard-Stage-Milestone approach [35, 25], which has been adopted by ACSI as the main language for declaratively modeling and enacting artifact-centric systems (Part II);

- support different concrete architectural frameworks by tuning how relational artifact systems can be combined with a Semantic Layer (Part III);

- realize an effective tool for the verification of GSM-based artifact systems equipped with a Semantic Layer, by combining techniques and technologies developed in the context of Ontology-Based Data Access [42], with the GSMC model checker for GSM, implemented as part of the ACSI project [32] (Part IV).

In addition, we apply the discussed technique to the ACSI Energy use case, showing the benefits of adding a Semantic Layer on top of the GSM-based artifacts used to handle energy control points and the corresponding monthly reports.

More in detail, the deliverable is organized in four parts, described in the following.

Part I. In this part of the deliverable, we provide an overview of the logic-based frameworks and the corresponding results reported in the first iteration of the deliverable to specify and verify Knowledge and Action Bases (KABs). In particular, we describe:

- *Data-Centric Dynamic Systems* (DCDS) as a mean to capture artifacts equipped with a relational information model;

- *Knowledge-Based Dynamic Systems* as a mean to capture semantic artifact, equipped with a full-fledged ontology for their information models;
- *Semantically-governed Artifact Systems*, where a Semantic Layer is added on top of the Artifact Layer.

For each one of such frameworks, we discuss the main restrictions that have been investigated to obtain interesting decidability results for their verification. The results of this activity are reported in the Proceedings of the *32nd ACM Symposium on Principles of Database Systems* (PODS 2013) [8], the Proceedings of the *20th European Conference on Artificial Intelligence* (ECAI 2012) [5], and the *Journal of Artificial Intelligence* Research [6], and rely on further technical results reported in ACSI Deliverable D2.5.2 [22].

Part II. In this part of the deliverable, we provide a translation procedure that, starting from a GSM model, produces a corresponding DCDS that faithfully reproduces the possible executions of the original GSM model. This allows us to provide decidability results for the verification of GSM-based artifact systems by relying on decidability of verification for DCDSs. The results of this activity are going to be submitted to the *International Conference on Service Oriented Computing* (ICSOC).

Part III. In this part of the deliverable, we discuss several concrete architectural solutions that can be realized by combining GSM-based artifact systems with a Semantic Layer. To show the potential of such solutions, we ground them in the context of the ACSI Energy use case. The results of this activity have been submitted to the *Journal on Data Semantics*.

Part IV. In this last part of the deliverable, we present a tool for the verification of GSM-based artifact systems equipped with a Semantic Layer. The tool includes a language for specifying mapping assertions that link the GSM artifact information models with domain-relevant concepts and relations present in the Semantic Layer. We show how Ontology-Based Data Access techniques can be applied to manipulate conceptual temporal properties specified over the evolutions of the system understood through the lens of the Semantic Layer, in such a way that they can be verified by the GSMC model checker. To show the potential of the tool, we show its application in the context of the ACSI Energy use case. The results of this activity are going to be submitted to the *International Conference on Service Oriented Computing* (ICSOC).

Part I
Knowledge and Action Bases

For the sake of self containment, in this part we provide a summary of the three main foundational frameworks studied within ACSI to formally account for knowledge and action bases. All such frameworks share the presence of two interacting layers: a *data component*, used to maintain the data of interest, and a *process component*, used to manipulate and evolve such data. In the context of artifact-centric systems, the data component can be seen as a pristine way for representing the artifact information models, whereas the process component reflects the artifact lifecycle constraints. Starting from this common basis, we review in particular:

- *Data-Centric Dynamic Systems* (DCDSs), where the data component is constituted by a relational database with constraints, and the process component manipulate instances of the database, possibly injecting new data from the external environment, through service calls.
- *Knowledge-Based Dynamic Systems* (KBDSs)[1], where the data component is constituted by a full-fledge (description logic) knowledge base, whose intensional knowledge (TBox) is maintained fixed, and whose extensional knowledge (ABox) is evolved by the process component, possibly injecting new data from the external environment, through service calls.
- *Semantically-Governed Artifact Systems* (SASs), which provide a formal account for the notion of ACSI *semantic layer*, which equips the artifact system with a full-fledged ontology that accounts for a conceptual description of the domain, and that is used to *understand* and *govern* the evolution of the artifact layer.

Part of the material presented in the following is extracted from previous ACSI deliverables [12, 18].

2 Data-Centric Dynamic Systems

In this section, we introduce the notion of *(relational) data-centric dynamic system*, or simply DCDS. A DCDS is a pair $\mathcal{S} = \langle \mathcal{D}, \mathcal{P} \rangle$ formed by two interacting layers: a *data layer* $\mathcal{D}$ and a *process layer* $\mathcal{P}$ over it. Intuitively, the data layer keeps all the data of interest, while the process layer modifies and evolves such data. We keep the structure of both layers to the minimum, in particular we do not distinguish between various possible components providing the data, nor those providing the subprocesses running concurrently. Indeed the framework can be further detailed in several directions, while keeping the results obtained here (see, e.g., Part II).

2.1 Data Layer

The information of interest in the modeled domain is represented in the data layer. It is constituted by a relational schema R equipped with equality constraints[2] $\mathcal{E}$, e.g., to state keys of relations, and an initial database instance $\mathcal{I}_0$, which conforms to the relational schema and the equality constraints. The values stored in this database belong to a predefined, possibly infinite, set $\mathcal{C}$ of constants. These constants are interpreted as themselves, blurring the distinction between constants and values. We will use the two terms interchangeably.

Given a database instance $\mathcal{I}$, its active domain $\textsc{adom}(\mathcal{I})$ is the subset of $\mathcal{C}$ such that $c \in \textsc{adom}(\mathcal{I})$ if and only if c occurs in $\mathcal{I}$.

A *data layer* is a tuple $\mathcal{D} = \langle \mathcal{C}, R, \mathcal{E}, \mathcal{I}_0 \rangle$ where:

[1] In the literature, these systems are referred to as KABs [6, 21], and as semantic artifacts in [12]. We use here KBDSs as an umbrella term, avoiding confusion with the general notion of KAB as defined in the ACSI A^3M.

[2] Other kinds of constraints can also be included without affecting the results reported here [8].

- $\mathcal{C}$ is a countably infinite set of constants/values.
- $R = \{R_1, \ldots, R_n\}$ is a database schema, constituted by a finite set of relation schemas.
- $\mathcal{E}$ is a finite set $\{\mathcal{E}_1, \ldots, \mathcal{E}_m\}$ of equality constraints. Each $\mathcal{E}_i$ has the form

$$Q_i \rightarrow \bigwedge_{j=1,\ldots,k} z_{ij} = y_{ij},$$

  where Q_i is a domain independent FO query over R using constants from the active domain $\textsc{adom}(\mathcal{I}_0)$ of $\mathcal{I}_0$ and whose free variables are $\vec{x}$, and z_{ij} and y_{ij} are either variables in $\vec{x}$ or constants in $\textsc{adom}(\mathcal{I}_0)$.[3]
- $\mathcal{I}_0$ is a database instance that represents the initial state of the data layer, which conforms to the schema R and *satisfies* the constraints $\mathcal{E}$: namely, for each constraint $Q_i \rightarrow \bigwedge_{j=1,\ldots,k} z_{ij} = y_{ij}$ and for each tuple (i.e., substitution for the free variables) $\theta \in ans(Q_i, \mathcal{I})$, it holds that $z_{ij}\theta = y_{ij}\theta$.[4]

2.2 Process Layer

The process layer constitutes the progression mechanism for the DCDS. We assume that at every time the current instance of the data layer can be arbitrarily queried, and can be updated through action executions, possibly involving external service calls to get new values from the environment. Hence the process layer is composed of three main notions: actions, which are the atomic progression steps for the data layer; external services, which can be called during the execution of actions; and processes, which are essentially nondeterministic programs that use actions as atomic instructions. While we require the execution of actions to be sequential, we do not impose any such constraints on processes, which in principle can be formed by several concurrent branches, including fork, join, and so on. Concurrency is to be interpreted by interleaving and hence reduced to nondeterminism, as often done in formal verification [9, 28]. There can be many ways to provide the control flow specification for processes. Here we adopt a simple rule-based mechanism, but our results can be immediately generalized to any process formalism whose processes control flow is finite-state. Notice that this does not imply that the transition system associated to a process over the data layer is finite-state as well, since the data manipulated in the data layer may grow over time in an unbounded way.

Formally, a process layer $\mathcal{P}$ over a data layer $\mathcal{D} = \langle \mathcal{C}, R, \mathcal{E}, \mathcal{I}_0 \rangle$, is a tuple $\mathcal{P} = \langle \mathcal{F}, A, \varrho \rangle$ where:

- $\mathcal{F}$ is a finite set of *functions*, each representing the interface to an *external service*. Such services can be called, and as a result the function is activated and the answer is produced. How the result is actually computed is *unknown* to the DCDS since the services are indeed external.
- A is a finite set of *actions*, whose execution progresses the data layer, and may involve external service calls.
- ϱ is a finite set of *condition-action rules* that form the specification of the overall *process*, which tells at any moment which actions can be executed.

An *action* $\alpha \in A$ has the form

$$\alpha(p_1, \ldots, p_n) : \{e_1, \ldots, e_m\},$$

where: *(i)* $\alpha(p_1, \ldots, p_n)$ is the *signature* of the action, constituted by a name α and a sequence $p_1, \ldots, p_n$ of *input parameters* that need to be substituted with values for the execution of the action, and *(ii)* $\{e_1, \ldots, e_m\}$, also denoted as $\textsc{effect}(\alpha)$, is a set of *effect specifications*, whose specified effects are assumed to take place simultaneously. Each e_i has the form $q_i^+ \wedge Q_i^- \rightsquigarrow E_i$, where:

[3] For convenience, and without loss of generality, we assume that all constants used inside formulae appear in $\mathcal{I}_0$.

[4] We use the notation $t\theta$ (resp., $\varphi\theta$) to denote the term (resp., the formula) obtained by applying the substitution θ to t (resp., φ). Furthermore, given a FO query Q and a database instance $\mathcal{I}$, the *answer* $ans(Q, \mathcal{I})$ to Q over $\mathcal{I}$ is the set of assignments θ from the free variables of Q to the domain of $\mathcal{I}$, such that $\mathcal{I} \models Q\theta$. We treat $Q\theta$ as a boolean query, and with some abuse of notation, we say $ans(Q\theta, \mathcal{I}) \equiv \mathsf{true}$ if and only if $\mathcal{I} \models Q\theta$.

- $q_i^+ \wedge Q_i^-$ is a query over R whose terms are variables $\vec{x}$, action parameters, and constants from $\text{ADOM}(\mathcal{I}_0)$. The query q_i^+ is a UCQ, and the query Q_i^- is an arbitrary FO formula whose free variables are included in those of q_i^+. Intuitively, q_i^+ selects the tuples to instantiate the effect, and Q_i^- filters away some of them.
- E_i is the effect, i.e., a set of facts for R, which includes as terms: terms in $\text{ADOM}(\mathcal{I}_0)$, input parameters, free variables of q_i^+, and in addition Skolem terms formed by applying a function $f \in \mathcal{F}$ to one of the previous kinds of terms. Such Skolem terms involving functions represent external service calls and are interpreted so as to return a value chosen by an external user/environment when executing the action.

The *process* ϱ is a finite set of *condition-action rules*, of the form $Q \mapsto \alpha$, where α is an action in A and Q is a FO query over R whose free variables are exactly the parameters of α, and whose other terms can be either quantified variables or constants in $\text{ADOM}(\mathcal{I}_0)$.

2.3 Semantics via Transition System

The semantics of a DCDS is defined in terms of a possibly infinite (relational) transition system whose states are labeled by databases. Such a transition system represents all possible computations that the process layer can do on the data layer. A *relational transition system*(RTS) Υ is a tuple of the form $\langle \Delta, R, \Sigma, s_0, db, \Rightarrow \rangle$, where:

- Δ is a countably infinite set of values;
- R is a database schema;
- Σ is a set of states;
- $s_0 \in \Sigma$ is the initial state;
- db is a function that, given a state $s \in \Sigma$, returns the database associated to s, which is made up of values in Δ and conforms to R;
- $\Rightarrow \subseteq \Sigma \times \Sigma$ is a transition relation between pairs of states.

Given a DCDS $\mathcal{S}$, we sketch in the following how to construct the RTS that captures its semantics (the complete description can be found in [8, 12]). The initial state corresponds to the initial database instance in the data layer of $\mathcal{S}$. The actions (with parameter assignments) that are executable in the initial database instance according to the condition-action rules of the process are determined, and then non-deterministically applied to obtain all possible candidate successor states. Among these candidates, only those that satisfy the constraints in the data layer are preserved as successors. This approach is then iteratively applied over such successors, and so on.

A key issue in the construction of the transition system is how to determine the successor state starting from the current state and a given action with parameter assignment. This is done by applying the effect specifications contained in the action, querying the current state with their left-hand side, and creating the facts contained in the right hand-side grounded with all the obtained answers. However, such facts may contain service calls, that requires to be evaluated so as to obtain the effective successor state(s). How the service call results are computed is determined by the service call semantics. In particular, two semantics have been considered:

- *Deterministic service calls* return the same value whenever they are invoked with the same input parameters along a run of the system.
- *Nondeterministic service calls* may return different values even when they are called with the same input parameters along a run of the system.

To reflect the fact that the behavior of external services is not under the control of the system, when a service is invoked with given input parameters, each possible value must be nondeterministically considered as being a possible obtained result (with the exception of an invocation which was already issued in the past, under the deterministic service call semantics).

Consequently, in general the obtained transition system may contain three sources of infinity/unboundedness:

- infinite branching, due to the interaction with external services, and the need for considering all possible results;
- infinite runs, constituted by infinitely many different database instances;
- unbounded database instances, obtained by iteratively applying actions that increase the number of tuples present in a state of the data layer.

2.4 Verification

To specify dynamic properties over a DCDS, we use μ-calculus [28, 45, 10], one of the most powerful temporal logics for which model checking has been investigated in the finite-state setting. Indeed, such a logic is able to express both linear time logics such as LTL and PSL, and branching time logics such as CTL and CTL* [24]. The main characteristic of μ-calculus is the ability of expressing directly least and greatest fixpoints of (predicate-transformer) operators formed using formulae relating the current state to the next one. By using such fixpoint constructs one can easily express sophisticated properties defined by induction or co-induction. This is the reason why virtually all logics used in verification can be considered as fragments of μ-calculus. From a technical viewpoint, μ-calculus separates local properties, i.e., properties asserted on the current state or on states that are immediate successors of the current one, and properties that talk about states that are arbitrarily far away from the current one [10]. The latter are expressed through the use of fixpoints.

In this work, we use a first-order variant of the μ-calculus [41], called $\mu\mathcal{L}$ and defined as follows:

$$\Phi ::= Q \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists x.\Phi \mid \langle - \rangle \Phi \mid Z \mid \mu Z.\Phi$$

where Q is a possibly open FO query, and Z is a second order predicate variable (of arity 0). We make use of the following abbreviations: $\forall x.\Phi = \neg(\exists x.\neg\Phi)$, $\Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2)$, $[-]\Phi = \neg\langle - \rangle \neg\Phi$, and $\nu Z.\Phi = \neg\mu Z.\neg\Phi[Z/\neg Z]$.

As usual in μ-calculus, formulae of the form $\mu Z.\Phi$ (and $\nu Z.\Phi$) must obey to the *syntactic monotonicity* of Φ wrt Z, which states that every occurrence of the variable Z in Φ must be within the scope of an even number of negation symbols. This ensures that the least fixpoint $\mu Z.\Phi$ (as well as the greatest fixpoint $\nu Z.\Phi$) always exists.

Since $\mu\mathcal{L}$ also contains formulae with both individual and predicate free variables, given an RTS Υ, we introduce an individual variable valuation v, i.e., a mapping from individual variables x to Δ, and a predicate variable valuation V, i.e., a mapping from predicate variables Z to subsets of Σ. With these three notions in place, we assign meaning to formulae by associating to Υ, v, and V an *extension function* $(\cdot)^{\Upsilon}_{v,V}$, which maps formulae to subsets of Σ. Formally, the extension function $(\cdot)^{\Upsilon}_{v,V}$ is defined inductively as shown in Figure 1.

Intuitively, the extension function $(\cdot)^{\Upsilon}_{v,V}$ assigns to such constructs the following meaning:

- The boolean connectives and quantification of individuals have the expected meaning.
- The extension of $\langle - \rangle \Phi$ consists of the states s such that for *some* state s' with $s \Rightarrow s'$, we have that Φ holds in s', while the extension of $[-]\Phi$ consists of the states s such that for *all* states s' with $s \Rightarrow s'$, we have that Φ holds in s'.
- The extension of $\mu X.\Phi$ is the *smallest subset* $\mathcal{S}_\mu$ of Σ such that, assigning to Z the extension $\mathcal{S}_\mu$, the resulting extension of Φ is contained in $\mathcal{S}_\mu$. That is, the extension of

$$
\begin{array}{rcl}
(Q)^{\Upsilon}_{v,V} & = & \{s \in \Sigma \mid ans(Qv, db(s))\} \\
(\neg\Phi)^{\Upsilon}_{v,V} & = & \Sigma \setminus (\Phi)^{\Upsilon}_{v,V} \\
(\Phi_1 \wedge \Phi_2)^{\Upsilon}_{v,V} & = & (\Phi_1)^{\Upsilon}_{v,V} \cap (\Phi_2)^{\Upsilon}_{v,V} \\
(\exists x.\Phi)^{\Upsilon}_{v,V} & = & \{s \in \Sigma \mid \exists t.t \in \Delta \text{ and } s \in (\Phi)^{\Upsilon}_{v[x/t],V}\} \\
(\langle - \rangle \Phi)^{\Upsilon}_{v,V} & = & \{s \in \Sigma \mid \exists s'.s \Rightarrow s' \text{ and } s' \in (\Phi)^{\Upsilon}_{v,V}\} \\
(Z)^{\Upsilon}_{v,V} & = & V(Z) \\
(\mu Z.\Phi)^{\Upsilon}_{v,V} & = & \bigcap\{\mathcal{S} \subseteq \Sigma \mid (\Phi)^{\Upsilon}_{v,V[Z/\mathcal{S}]} \subseteq \mathcal{S}\}
\end{array}
$$

Figure 1 – Semantics of $\mu\mathcal{L}$

$\mu X.\Phi$ is the *least fixpoint* of the operator $(\Phi)^{\Upsilon}_{v,V[Z/\mathcal{S}]}$ (here $V[Z/\mathcal{S}]$ denotes the predicate valuation obtained from V by forcing the valuation of Z to be $\mathcal{S}$).

- Similarly, the extension of $\nu X.\Phi$ is the *greatest subset* $\mathcal{S}_\nu$ of Σ such that, assigning to X the extension $\mathcal{S}_\nu$, the resulting extension of Φ contains $\mathcal{S}_\nu$. That is, the extension of $\nu X.\Phi$ is the *greatest fixpoint* of the operator $(\Phi)^{\Upsilon}_{v,V[X/\mathcal{S}]}$. Formally, $(\nu Z.\Phi)^{\Upsilon}_{v,V} = \bigcup\{\mathcal{S} \subseteq \Sigma \mid \mathcal{S} \subseteq (\Phi)^{\Upsilon}_{v,V[Z/\mathcal{S}]}\}$.

Example 2.1. An example of $\mu\mathcal{L}$ formula is:

$$\exists x_1, \ldots, x_n. \bigwedge_{i \neq j} x_i \neq x_j \wedge \bigwedge_{i \in \{1,\ldots,n\}} \mu Z.[Stud(x_i) \vee \langle - \rangle Z] \tag{1}$$

The formula asserts that there are at least n distinct objects/values, each of which eventually denotes a student along some execution path. Notice that the formula does not imply that all of these students will be in the same state, nor that they will all occur in a single run. It only says that in the entire RTS there are (at least) n distinct students. ■

When Φ is a closed formula, $(\Phi)^{\Upsilon}_{v,V}$ depends neither on v nor on V, and we denote the extension of Φ simply by $(\Phi)^{\Upsilon}$. We say that a closed formula Φ holds in a state $s \in \Sigma$ if $s \in (\Phi)^{\Upsilon}$. In this case, we write $\Upsilon, s \models \Phi$. We say that a closed formula Φ holds in Υ, denoted by $\Upsilon \models \Phi$, if $\Upsilon, s_0 \models \Phi$, where s_0 is the initial state of Υ. We call *model checking* verifying whether $\Upsilon \models \Phi$ holds.

In particular we are interested in formally verifying properties of a DCDS. Given the RTS $\Upsilon_{\mathcal{S}}$ of a DCDS $\mathcal{S}$ and a dynamic property Φ expressed in $\mu\mathcal{L}$,[5] we say that $\mathcal{S}$ *verifies* Φ if

$$\Upsilon_{\mathcal{S}} \models \Phi.$$

The challenging point is that $\Upsilon_{\mathcal{S}}$ is in general-infinite state, so we would like to devise a finite-state RTS that is a faithful abstraction of $\Upsilon_{\mathcal{S}}$, in the sense that it preserves the truth value of all $\mu\mathcal{L}$ formulae. Unfortunately, this program is doomed right from the start if we insist on using full $\mu\mathcal{L}$ as the verification formalism. Indeed formulae of the form (1) defeat any kind of finite-state transition system. So next we introduce two interesting sublogics of $\mu\mathcal{L}$ that serve better our objective.

2.5 History-Preserving Mu-Calculus

The first fragment of $\mu\mathcal{L}$ that we consider is $\mu\mathcal{L}_A$, which is characterized by the assumption that quantification over individuals is restricted to individuals that are present in the current database. To enforce such a restriction, we introduce a special predicate LIVE(x), which states that x belongs to the current active domain. The logic $\mu\mathcal{L}_A$ is defined as follows:

$$\Phi ::= Q \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists x.\text{LIVE}(x) \wedge \Phi \mid \langle - \rangle \Phi \mid Z \mid \mu Z.\Phi$$

[5] We remind the reader that, without loss of generality, we assume that all constants used inside formulae Φ appear in the initial database instance of the DCDS.

We make use of the usual abbreviation, including $\forall x.\textsc{live}(x) \rightarrow \Phi = \neg(\exists x.\textsc{live}(x) \wedge \neg\Phi)$. Formally, the extension function $(\cdot)^{\Upsilon}_{v,V}$ is defined inductively as in Figure 1, with the new special predicate $\textsc{live}(x)$ interpreted as follows:

$$(\textsc{live}(x))^{\Upsilon}_{v,V} = \{s \in \Sigma \mid x/d \in v \text{ implies } d \in \textsc{adom}(db(s))\}$$

Example 2.2. As an example, consider the following $\mu\mathcal{L}_A$ formula:

$$\nu X.(\forall x.\textsc{live}(x) \wedge \mathit{Stud}(x) \rightarrow \mu Y.(\exists y.\textsc{live}(y) \wedge \mathit{Grad}(x,y) \vee \langle - \rangle Y) \wedge [-]X),$$

which states that, along every path, it is always true, for each student x, that there exists an evolution that eventually leads to a graduation of the student (with some final mark y). ■

We are going to show that under suitable conditions we can get a faithful finite abstraction for a DCDS that preserves all formulae of $\mu\mathcal{L}_A$, and hence enables us in principle to use standard model checking techniques. Towards this goal, we introduce a notion of bisimulation that is suitable for the kind of transition systems we consider here. In particular, we have to take into account that the two RTSs are over different data domains, and hence we have to consider the correspondence between the data in the two transition systems and how such data evolve over time. To do so, we introduce the following notions.

Given two domains Δ_1 and Δ_2, a *partial bijection* h between Δ_1 and Δ_2 is a bijection between a subset of Δ_1 and Δ_2. Given a partial function $f : S \rightarrow S'$, we denote with $\textsc{dom}(f)$ the domain of f, i.e., the set of elements in S on which f is defined, and with $\textsc{im}(f)$ the image of f, i.e., the set of elements s' in S' such that $s' = f(s)$ for some $s \in S$. A partial bijection h' *extends* h if $\textsc{dom}(h) \subseteq \textsc{dom}(h')$ (or equivalently $\textsc{im}(h) \subseteq \textsc{im}(h')$) and $h'(x) = h(x)$ for all $x \in \textsc{dom}(h)$ (or equivalently $h'^{-1}(y) = h^{-1}(y)$ for all $y \in \textsc{im}(h)$). Let db_1 and db_2 be two databases over two domains Δ_1 and Δ_2 respectively, both conforming to the same schema R. We say that a partial bijection h *induces an isomorphism* between db_1 and db_2 if $\textsc{adom}(db_1) \subseteq \textsc{dom}(h)$, $\textsc{adom}(db_2) \subseteq \textsc{im}(h)$, and h projected on $\textsc{adom}(db_1)$ is an isomorphism between db_1 and db_2.

Given two RTSs $\Upsilon_1 = \langle \Delta_1, R, \Sigma_1, s_{01}, db_1, \Rightarrow_1 \rangle$ and $\Upsilon_2 = \langle \Delta_2, R, \Sigma_2, s_{02}, db_2, \Rightarrow_2 \rangle$, and the set H of partial bijections between Δ_1 and Δ_2, a *history preserving bisimulation* between Υ_1 and Υ_2 is a relation $\mathcal{B} \subseteq \Sigma_1 \times H \times \Sigma_2$ such that $\langle s_1, h, s_2 \rangle \in \mathcal{B}$ implies that:

1. h is a partial bijection between Δ_1 and Δ_2 that induces an isomorphism between $db_1(s_1)$ and $db_2(s_2)$;
2. for each s'_1, if $s_1 \Rightarrow_1 s'_1$ then there is an s'_2 with $s_2 \Rightarrow_2 s'_2$ and a bijection h' that extends h, such that $\langle s'_1, h', s'_2 \rangle \in \mathcal{B}$.
3. for each s'_2, if $s_2 \Rightarrow_2 s'_2$ then there is an s'_1 with $s_1 \Rightarrow_1 s'_1$ and a bijection h' that extends h, such that $\langle s'_1, h', s'_2 \rangle \in \mathcal{B}$.

A state $s_1 \in \Sigma_1$ is *history preserving bisimilar* to $s_2 \in \Sigma_2$ *wrt a partial bijection* h, written $s_1 \approx_h s_2$, if there exists a history preserving bisimulation $\mathcal{B}$ between Υ_1 and Υ_2 such that $\langle s_1, h, s_2 \rangle \in \mathcal{B}$. A state $s_1 \in \Sigma_1$ is *history preserving bisimilar* to $s_2 \in \Sigma_2$, written $s_1 \approx s_2$, if there exists a partial bijection h and a history preserving bisimulation $\mathcal{B}$ between Υ_1 and Υ_2 such that $\langle s_1, h, s_2 \rangle \in \mathcal{B}$. An RTS Υ_1 is *history preserving bisimilar* to Υ_2, written $\Upsilon_1 \approx \Upsilon_2$, if there exists a partial bijection h_0 and a history preserving bisimulation $\mathcal{B}$ between Υ_1 and Υ_2 such that $\langle s_{01}, h_0, s_{02} \rangle \in \mathcal{B}$. The next theorem gives us the classical invariance result of μ-calculus wrt bisimulation, in our setting.

Theorem 2.1. *Consider two RTSs Υ_1 and Υ_2 such that $\Upsilon_1 \approx \Upsilon_2$. Then for every $\mu\mathcal{L}_A$ closed formula Φ, we have:*

$$\Upsilon_1 \models \Phi \text{ if and only if } \Upsilon_2 \models \Phi.$$

2.6 Persistence Preserving Mu-Calculus

The second fragment of $\mu\mathcal{L}$ that we consider is $\mu\mathcal{L}_P$, which further restricts $\mu\mathcal{L}_A$ by requiring that individuals over which we quantify must continuously persist along the system evolution for the quantification to take effect.

With a slight abuse of notation, in the following we write $\textsc{live}(x_1, \ldots, x_n) = \bigwedge_{i \in \{1,\ldots,n\}} \textsc{live}(x_i)$.

The logic $\mu\mathcal{L}_P$ is defined as follows:

$$\Phi ::= Q \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists x.\textsc{live}(x) \wedge \Phi \mid \langle - \rangle(\textsc{live}(\vec{x}) \wedge \Phi) \mid [-](\textsc{live}(\vec{x}) \wedge \Phi) \mid Z \mid \mu Z.\Phi$$

where Q is a possibly open FO query, Z is a second order predicate variable, and the following assumption holds: in $\langle - \rangle(\textsc{live}(\vec{x}) \wedge \Phi)$ and $[-](\textsc{live}(\vec{x}) \wedge \Phi)$, the variables $\vec{x}$ are exactly the free variables of Φ, with the proviso that we substitute to each bounded predicate variable Z in Φ its bounding formula $\mu Z.\Phi'$. We use the usual abbreviations, including: $\langle - \rangle(\textsc{live}(\vec{x}) \rightarrow \Phi) = \neg[-](\textsc{live}(\vec{x}) \wedge \neg\Phi)$ and $[-](\textsc{live}(\vec{x}) \rightarrow \Phi) = \neg\langle - \rangle(\textsc{live}(\vec{x}) \wedge \neg\Phi)$. Intuitively, the use of $\textsc{live}(\cdot)$ in $\mu\mathcal{L}_P$ ensures that individuals are only considered if they persist along the system evolution, while the evaluation of a formula with individuals that are not present in the current database trivially leads to false or true (depending on the use of negation).

Example 2.3. Getting back to the example above, its variant in $\mu\mathcal{L}_P$ is

$$\nu X.(\forall x.\textsc{live}(x) \wedge \mathit{Stud}(x) \rightarrow \mu Y.(\exists y.\textsc{live}(y) \wedge \mathit{Grad}(x, y) \vee \langle - \rangle(\textsc{live}(x) \wedge Y)) \wedge [-]X)$$

which states that, along every path, it is always true, for each student x, that there exists an evolution in which x persists in the database until she eventually graduates (with some final mark y). Formula

$$\nu X.(\forall x.\textsc{live}(x) \wedge \mathit{Stud}(x) \rightarrow \mu Y.(\exists y.\textsc{live}(y) \wedge \mathit{Grad}(x, y) \vee \langle - \rangle(\textsc{live}(x) \rightarrow Y)) \wedge [-]X)$$

instead states that, along every path, it is always true, for each student x, that there exists an evolution in which either x is not persisted, or eventually graduates (with final mark y). ∎

The bisimulation relation that captures $\mu\mathcal{L}_P$ is as follows. Given two RTSs $\Upsilon_1 = \langle \Delta_1, R, \Sigma_1, s_{01}, db_1, \Rightarrow_1 \rangle$ and $\Upsilon_2 = \langle \Delta_2, R, \Sigma_2, s_{02}, db_2, \Rightarrow_2 \rangle$, and the set H of partial bijections between Δ_1 and Δ_2, a *persistence preserving bisimulation* between Υ_1 and Υ_2 is a relation $\mathcal{B} \subseteq \Sigma_1 \times H \times \Sigma_2$ such that $\langle s_1, h, s_2 \rangle \in \mathcal{B}$ implies that:

1. h is an isomorphism between $db_1(s_1)$ and $db_2(s_2)$;[6]
2. for each s_1', if $s_1 \Rightarrow_1 s_1'$ then there exists an s_2' with $s_2 \Rightarrow_2 s_2'$ and a bijection h' that extends $h|_{\textsc{adom}(db_1(s_1)) \cap \textsc{adom}(db_1(s_1'))}$, such that $\langle s_1', h', s_2' \rangle \in \mathcal{B}$;[7]
3. for each s_2', if $s_2 \Rightarrow_2 s_2'$ then there exists an s_1' with $s_1 \Rightarrow_1 s_1'$ and a bijection h' that extends $h|_{\textsc{adom}(db_1(s_1)) \cap \textsc{adom}(db_1(s_1'))}$, such that $\langle s_1', h', s_2' \rangle \in \mathcal{B}$.

We say that a state $s_1 \in \Sigma_1$ is *persistence preserving bisimilar* to $s_2 \in \Sigma_2$ *wrt a partial bijection* h, written $s_1 \sim_h s_2$, if there exists a persistence preserving bisimulation $\mathcal{B}$ between Υ_1 and Υ_2 such that $\langle s_1, h, s_2 \rangle \in \mathcal{B}$. A state $s_1 \in \Sigma_1$ is *persistence preserving bisimilar* to $s_2 \in \Sigma_2$, written $s_1 \sim s_2$, if there exists a partial bijection h and a persistence preserving bisimulation $\mathcal{B}$ between Υ_1 and Υ_2 such that $\langle s_1, h, s_2 \rangle \in \mathcal{B}$. An RTS Υ_1 is *persistence preserving bisimilar* to Υ_2, written $\Upsilon_1 \sim \Upsilon_2$, if there exists a partial bijection h_0 and a persistence preserving bisimulation $\mathcal{B}$ between Υ_1 and Υ_2 such that $\langle s_{01}, h_0, s_{02} \rangle \in \mathcal{B}$. The next theorem shows the invariance of $\mu\mathcal{L}_P$ under this notion of bisimulation.

[6] Notice that this implies $\textsc{dom}(h) = \textsc{adom}(db_1(s_1))$ and $\textsc{im}(h) = \textsc{adom}(db_2(s_2))$.

[7] Given a set D, we denote by $f|_D$ the *restriction* of f to D, i.e., $\textsc{dom}(f|_D) = \textsc{dom}(f) \cap D$, and $f|_D(x) = f(x)$ for every $x \in \textsc{dom}(f) \cap D$.

	DETERMINISTIC					NONDETERMINISTIC				
	$\mu\mathcal{L}$		$\mu\mathcal{L}_A$		$\mu\mathcal{L}_P$	$\mu\mathcal{L}$		$\mu\mathcal{L}_A$		$\mu\mathcal{L}_P$
unrestricted	**U**	$\leftarrow$	**U**	$\leftarrow$	**U**[1]	**U**	$\leftarrow$	**U**	$\leftarrow$	**U**[1]
	$\uparrow$		$\uparrow$			$\uparrow$		$\uparrow$		
state-bounded	**U**	$\leftarrow$	**U**		**D**	**U**	$\leftarrow$	**U**		**D**[3]
					$\downarrow$					$\downarrow$
run-bounded	**?**[2]		**D**[3]	$\rightarrow$	**D**	**?**[2]		**D**[3]	$\rightarrow$	**D**

[1]Holds even for propositional LTL/CTL.
[2]Decidability cannot rely on faithful finite-state abstraction.
[3]Via reduction to finite-state model checking.

Table 1 – Summary of (un)decidability results for verification of DCDSs

Theorem 2.2. *Consider two RTSs* Υ_1 *and* Υ_2 *such that* $\Upsilon_1 \sim \Upsilon_2$*. Then for every* $\mu\mathcal{L}_P$ *closed formula* Φ*, we have:*

$$\Upsilon_1 \models \Phi \text{ if and only if } \Upsilon_2 \models \Phi.$$

2.7 Summary of (Un)Decidability Results

Table 1 summarizes the main results related to decidability and undecidability of verification for DCDSs for the logics introduced before, and by placing different assumptions on the DCDS under study. Notice that studying the verification problem with different assumptions on the DCDS is required. As shown in the first row of Table 1, verification for unrestricted DCDSs is already in place for propositional LTL/CTL properties; the expressiveness of such logics is clearly far below what is needed to query the (database instances contained in the) states of the system, and to formalize properties that predicate about the evolution of the individuals extracted with such queries.

We consider in particular two classes of DCDSs:

- *run-bounded* DCDSs, for which the fresh data introduced along each run are bounded, though they might not be bounded in the overall system;
- *state-bounded* DCDSs, for which the values that accumulate in each state are bounded, tough they might not be bounded along the system runs.

Notably, we proved decidability of verification of $\mu\mathcal{L}_A$ properties for run-bounded DCDSs, and decidability of verification fo $\mu\mathcal{L}_P$ properties for state-bounded DCDSs. The complete proofs for all results presented in Table 1 can be found in [7, 12].

As a last remark, we observe that both run- and state-boundedness are semantic properties that are undecidable to check. To contrast this problem, in [7, 12] syntactic, sufficient conditions are introduced, which can be actually checked against the specification of a DCDS to guarantee its run- or state-boundedness.

3 Knowledge-Based Dynamic Systems

Knowledge-Based Dynamic Systems (KBDSs) are data-centric dynamic systems that, differently from DCDSs, adopt a full-fledged (description logic) knowledge base to conceptually describe the data layer, under incomplete data. Different variants of this setting have been studied in the context of ACSI, mainly by changing the description logic used to formally capture the knowledge base, and the way service calls are treated.

We report here the variant that adopts the lightweight description logic *DL-Lite*$_\mathcal{A}$, which belongs to the *DL-Lite* family of ontology languages [17, 3, 13], which suitably balance between expressiveness (they capture the key constructs used in conceptual modeling languages such as UML and E-R) and good computational properties (especially in connection with huge amounts of data). See also [12, 6] to complete the picture presented here.

3.1 *DL-Lite*$_\mathcal{A}$ Knowledge Bases

For expressing knowledge bases, we use *DL-Lite*$_\mathcal{A}$ [42, 13]. The syntax of *concept* and *role expressions* in *DL-Lite*$_\mathcal{A}$ is as follows

$$B \longrightarrow N \mid \exists R \qquad\qquad R \longrightarrow P \mid P^-$$

where N denotes a *concept name*, P a *role name*, and P^- an *inverse role*. A *DL-Lite*$_\mathcal{A}$ *knowledge base* (KB) is a pair (T, A), where: *(i)* A is an Abox, i.e., a finite set of *ABox membership assertions* of the form $N(t_1) \mid P(t_1, t_2)$, where t_1, t_2 denote individuals *(ii)* T is a TBox, i.e., $T = T_p \uplus T_n \uplus T_f$, with T_p a finite set of *positive inclusion assertions* of the form $B_1 \sqsubseteq B_2$, T_n a finite set of *negative inclusion assertions* of the form $B_1 \sqsubseteq \neg B_2$, and T_f a finite set of *functionality assertions* of the form (funct R).

We adopt the standard FOL semantics of DLs based on FOL interpretations $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ such that $c^\mathcal{I} \in \Delta^\mathcal{I}$, $N^\mathcal{I} \subseteq \Delta^\mathcal{I}$, and $P^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. The semantics of the construct, of TBox and ABox assertions, and the notions of *satisfaction* and of *model* are as usual. We also say that A is T*-consistent* if (T, A) is satisfiable, i.e., admits at least one model, otherwise we say A is T*-inconsistent*.

Queries. As usual (cf. OWL 2 QL), answers to queries are formed by terms denoting individuals explicitly mentioned in the ABox. The *domain of an ABox* A, denoted by $\textsc{adom}(A)$, is the (finite) set of terms appearing in A. A *union of conjunctive queries* (UCQ) q over a KB (T, A) is a FOL formula of the form $\bigvee_{1 \le i \le n} \exists \vec{y_i}.conj_i(\vec{x}, \vec{y_i})$ with free variables $\vec{x}$ and existentially quantified variables $\vec{y_1}, \ldots, \vec{y_n}$. Each $conj_i(\vec{x}, \vec{y_i})$ in q is a conjunction of atoms of the form $N(z)$, $P(z, z')$, where N and P respectively denote a concept and a role name occurring in T, and z, z' are constants in $\textsc{adom}(A)$ or variables in $\vec{x}$ or $\vec{y_i}$, for some $i \in \{1, \ldots, n\}$. The *(certain) answers* to q over (T, A) is the set $ans(q, T, A)$ of substitutions σ of the free variables of q with constants in $\textsc{adom}(A)$ such that $q\sigma$ evaluates to true in every model of (T, A). If q has no free variables, then it is called *boolean* and its certain answers are either true or false.

We compose UCQs using ECQs, i.e., queries of the query language *EQL-Lite*(UCQ) [16], which is the FOL query language whose atoms are UCQs evaluated according to the certain answer semantics above. An *ECQ* over T and A is a possibly open formula of the form

$$Q \;:=\; [q] \;\mid\; \neg Q \;\mid\; Q_1 \wedge Q_2 \;\mid\; \exists x.Q$$

where q is a UCQ. The *answer to* Q *over* (T, A), is the set $\textsc{ans}(Q, T, A)$ of tuples of constants in $\textsc{adom}(A)$ defined by composing the certain answers $ans(q, T, A)$ of UCQs q through first-order constructs, and interpreting existential variables as ranging over $\textsc{adom}(A)$.

Finally, we recall that *DL-Lite*$_\mathcal{A}$ enjoys the *FO rewritability* property, which states that for every UCQ q, $ans(q, T, A) = ans(rew(q), \emptyset, A)$, where $rew(q)$ is a UCQ computed by the reformulation algorithm in [13]. Notice that this algorithm can be extended to ECQs [16], and that its effect is to "compile away" the TBox. Similarly, also ontology satisfiability is FO rewritable for *DL-Lite*$_\mathcal{A}$ TBoxes [13].

3.2 Knowledge-Based Dynamic Systems: Definition

Similarly to DCDSs, we make use of a countably infinite set $\mathcal{C}$ of constant to denote all possible value in the system. Moreover, we also make use of a finite set $\mathcal{F}$ of functions that represent service calls, and can be used to inject fresh values into the system.

A knowledge-based dynamic system (KBDS) is a tuple $\mathcal{K} = (T, A_0, \Gamma, \Pi)$ where T and A_0 form the knowledge base (KB), and Γ and Π form the action base. Intuitively, the KB maintains the information of interest. It is formed by a fixed *DL-Lite*$_\mathcal{A}$ TBox T and an initial T*-consistent DL-Lite*$_\mathcal{A}$ ABox A_0. A_0 represents the initial state of the system and, differently from T, it evolves and incorporates new information from the external world by executing actions Γ, according to the sequencing established by process Π. The definitions for Γ and Π resemble the ones given for DCDSs, though by using UCQs and ECQs to specify conditions/queries. Γ is a

finite set actions. An *action* $\gamma \in \Gamma$ generates a new ABox A' from the current ABox A, by adding or deleting assertions. γ is constituted by a signature and an effect specification. The *action signature* is constituted by a name and a list of individual *input parameters*. Such parameters need to be instantiated with individuals for the execution of the action. Given a substitution θ for the input parameters, we denote by $\gamma\theta$ the instantiated action with the *actual* parameters coming from θ. The *effect specification* consists of a set $\{e_1, \ldots, e_n\}$ of effects, which take place simultaneously. An *effect* e_i has the form $[q_i^+] \wedge Q_i^- \rightsquigarrow A_i'$, where:

- q_i^+ is an UCQ, and Q_i^- is an arbitrary ECQ whose free variables occur all among the free variables of q_i^+;
- A_i' is a set of facts (over the alphabet of T) which include as terms: individuals in A_0, free variables of q_i^+, and Skolem terms $f(\vec{x})$ having as arguments free variables $\vec{x}$ of q_i^+.

The process Π is a finite set of condition/action rules. A *condition/action rule* $\pi \in \Pi$ is an expression of the form $Q \mapsto \gamma$, where γ is an action in Γ and Q is an ECQ over T, whose free variables are exactly the parameters of γ. The rule expresses that, for each tuple σ for which condition Q holds, the action γ with actual parameters σ *can* be executed.

3.3 Semantics via Transition System

The semantics of a KBDS is defined in terms of a possibly infinite (semantic) transition system whose states are labeled by ABoxes. Such a transition system represents all possible computations that the process layer can do on the extensional part of the knowledge base. A *semantic transition system* (STS) Υ is a tuple of the form $\langle \Delta, T, \Sigma, s_0, db, \Rightarrow \rangle$, where:

- Δ is a countably infinite set of values;
- R is a database schema;
- Σ is a set of states;
- $s_0 \in \Sigma$ is the initial state;
- *abox* is a function that, given a state $s \in \Sigma$, returns the ABox associated to s, which is made up of values in Δ and is consistent with T;
- $\Rightarrow \subseteq \Sigma \times \Sigma$ is a transition relation between pairs of states.

The construction of the STS that capture the execution semantics of a KBDS follows the same steps sketched in Section 2.3 for DCDSs, with two main differences: each ABox is required to be consistent with the TBox of the KBDS, and the progression mechanism used to construct the successor ABoxes (by nondeterministically applying the executable actions) is not anymore driven by query evaluation, but by logical implications, computing certain answers and consequently taking into account the constraints of the TBox, as well as the fact that ABoxes contain incomplete data.

3.4 Summary of (Un)Decidability Results

We now summarise the main results related to decidability of verification for unrestricted, run-bounded, and state-bounded KBDSs, where the considered logics are variants of $\mu\mathcal{L}_A$ (cf. Section 2.5) and $\mu\mathcal{L}_P$ (cf. Section 2.6). In particular, we consider the two logics $\mu\mathcal{L}_A^{\text{EQL}}$ and $\mu\mathcal{L}_P^{\text{EQL}}$, which respectively correspond to $\mu\mathcal{L}_A$ and $\mu\mathcal{L}_P$ by replacing the local first-order queries with ECQs (cf. Section 3.1). These logics support ECQs over the ABoxes of the KBDS transition system, and provide restricted forms of quantification across states so as to predicate about the evolution of the individuals returned by computing the certain answers of such ECQs.

Table 2 shows that the (un)decidability results established for DCDSs can be mirrored also in the KBDS setting. The entires in the table are obtained by combining the results presented

	DETERMINISTIC					NONDETERMINISTIC				
	$\mu\mathcal{L}$		$\mu\mathcal{L}_A^{\text{EQL}}$		$\mu\mathcal{L}_P^{\text{EQL}}$	$\mu\mathcal{L}$		$\mu\mathcal{L}_A^{\text{EQL}}$		$\mu\mathcal{L}_P^{\text{EQL}}$
unrestricted	**U**	$\leftarrow$	**U**	$\leftarrow$	**U**[1]	**U**	$\leftarrow$	**U**	$\leftarrow$	**U**[1]
	$\uparrow$		$\uparrow$			$\uparrow$		$\uparrow$		
state-bounded	**U**	$\leftarrow$	**U**		**D**	**U**	$\leftarrow$	**U**		**D**[3]
					$\downarrow$					$\downarrow$
run-bounded	**?**[2]		**D**[3]	$\rightarrow$	**D**	**?**[2]		**D**[3]	$\rightarrow$	**D**

[1]Holds even for LTL/CTL formulae containing just an instance query.
[2]Decidability cannot rely on faithful finite-state abstraction.
[3]Via reduction to finite-state model checking.

Table 2 – Summary of (un)decidability results for verification of KBDSs

in [21, 20, 6], and the ones provided for DCDSs, recalling that, thanks to the FO rewritability of *DL-Lite*$_\mathcal{A}$, it is possible to "compile away" the TBox of a KBDS and obtain a corresponding DCDS.

As a last observation, we point out that the syntactic, sufficient conditions introduced to check run- and state-boundedness over DCDSs can be recast for KBDSs as well. The main difference is that the contribution of the TBox must be taken into account. In particular, as shown in [6], such syntactic conditions have to be tested over the action base of the KBDS under study, after having rewritten all the contained queries w.r.t. the TBox.

4 Semantically-governed Artifact Systems

In this section, we discuss the framework of *semantically-governed artifact systems* (SASs), which fully account with Artifact Systems as defined in the ACSI setting, covering the Artifact and Semantic Layers of the abstract ACSI architecture (cf. [14]). According to such architecture, SASs model systems in which processes are executed at the Artifact Layer, typically relying on relational databases to capture the artifact information models, but the execution can be understood at the semantic layer, by accessing these information models through a full-fledged ontology/conceptual schema.

In particular, SASs are constituted by:

- An Artifact Layer, which abstractly account for the (relational) information models of the artifacts, and which uses a transition relation to globally capture the evolution of such information models.
- A description logic knowledge base, which typically relies on the *DL-Lite* family [17] (such as *DL-Lite*$_\mathcal{A}$ – cf. Section 3.1), to conceptually model the domain under study, and equip the artifact system with a Semantic Layer.
- A set of *mappings* that describe how to virtually project data concretely maintained at the Artifact Layer into the Semantic Layer, thus providing a link between data and the ontology. We assume that the Artifact Layer exploits relational databases to maintain the data of interest, and can be hence queried by means of SQL statements; in this context context, each mapping assertion relates an arbitrary (SQL) query over the databases of the artifacts to a set of atoms whose predicates are the concepts and roles of the ontology, and whose arguments are terms built using specific function symbols applied to the answer variables of the SQL query. Such mappings specify how to populate the elements of the ontology from the data in the relational database, and the function symbols are used to construct (abstract) objects (which we call *object terms*) from the concrete values retrieved from the database.
- The temporal logics $\mu\mathcal{L}_A^{\text{EQL}}$ and $\mu\mathcal{L}_P^{\text{EQL}}$ (cf. Section 3.4) to declaratively capture at the semantic layer the dynamic laws that must be guaranteed by the artifact system. Notice

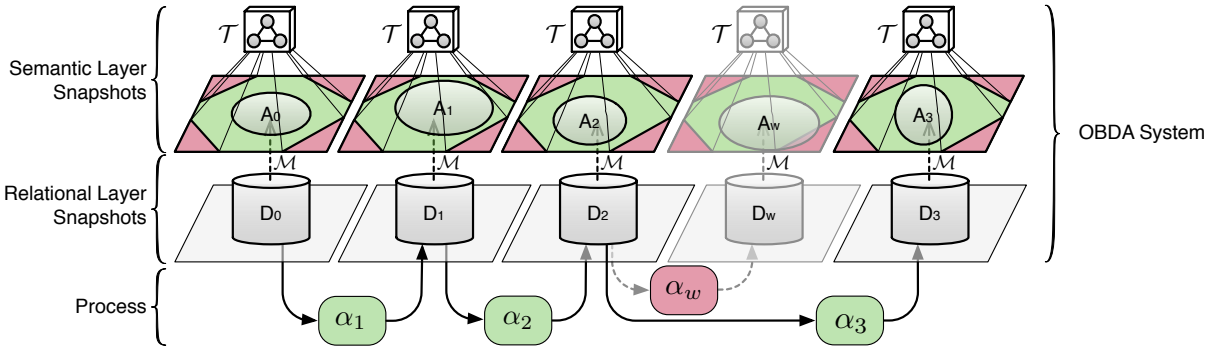


Figure 2 – Evolution of an SAS, condensing artifact's data in a unique database

that these formulae predicate about the evolution of data at the conceptual level, taking into account the knowledge and constraints present in the ontology, which allows for inferring information that is only implicitly available.

Since the main focus of this investigation is on how data are manipulated at the Artifact Layer and understood/governed at the Semantic Layer, we assume, as done for DCDSs and KBDSs, that the artifact databases are combined into a unique database maintaining the whole information present at the Artifact Layer, which is then referred to as the *Relational Layer*.

4.1 The Role of the Semantic Layer

As specified in the A^3M, we assume that artifacts evolve as a result of processes running over the Artifact Layer, where the execution of an action leads to manipulate the data maintained by one or more artifacts. This is represented, in A^3M, as a transition relation that connects the current artifact system snapshot with the successor one(s) [14]. In this setting, the TBox and mappings of the KAB can be used not only to *understand* the evolution of the artifact system at the conceptual level (see Fig. 3), but also to *govern* the evolution of the artifact system at the conceptual level, rejecting those snapshot transitions that, currently executed at the Artifact Layer, would lead to a new semantic snapshot that is inconsistent with the KAB's TBox (see Fig. 4).

Governance through the Semantic Layer gives raise to a so-called *Semantically-governed Artifact System* (SAS). We remark that, without loss of generality, we can condens all databases maintained by the artifacts into a unique database characterizing the whole Artifact Layer, which becomes, in this case, a Relational Layer. Under this assumption, an evolution of the system can be understood as depicted in Figure 2.

In this section, we study SASs using the lightweight description logics (cf. 3) to represent their KABs. In particular, building on FO rewritability of queries over the system state that is typical of *DL-Lite*$_\mathcal{A}$ and of ontology languages for ontology-based data access (OBDA [42]) in general, we show that the static as well as dynamic component of the KAB can be rewritten in terms of the underlying relational database schema. This is in line with the A^3M, where the Semantic Layer is used to provide a conceptual, high-level and user-oriented view of an artifact system, and where the artifact system evolves independently from the Semantic Layer, as a result of processes defined over the Artifact (Relational) Layer. More specifically, after having defined the execution semantics of an SAS, we discuss how to:

- Reformulate the semantic constraints of the TBox as denial constraints posed directly over the database schemas of the artifact information models. This means that, as far as the evolution of the artifact system is concerned, the Semantic Layer can be compiled away, provided that the transition relation of the Artifact Layer is modified so as to take such denial constraints into account.

- Reformulate the dynamic laws of the KAB into temporal properties expressed over the

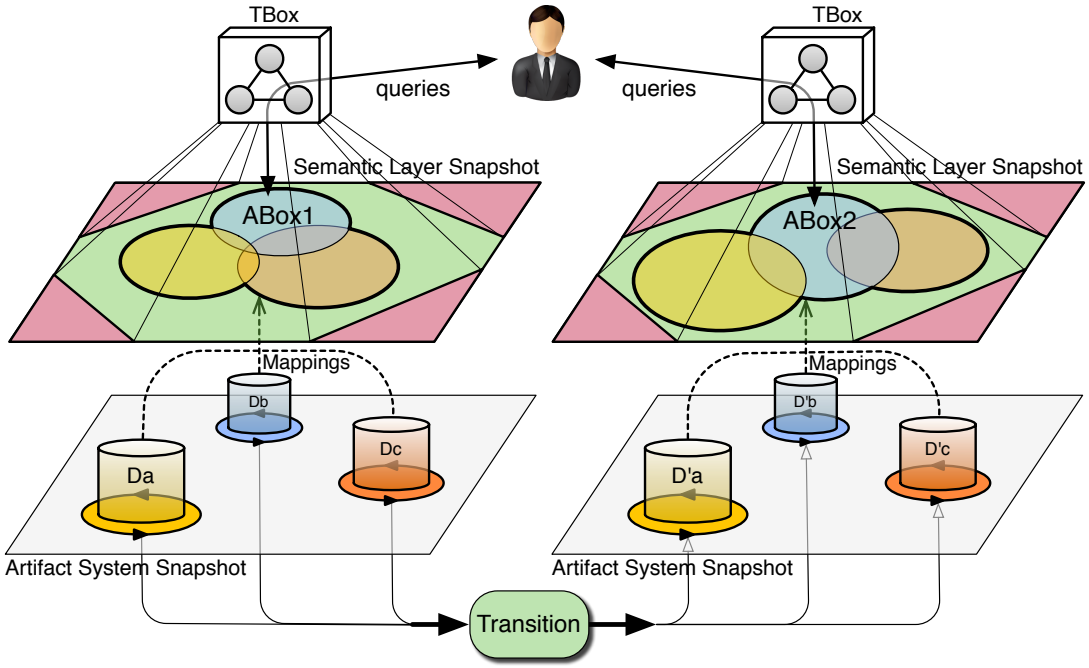


Figure 3 – Semantic Layer as a mean to understand the snapshots of an artifact system evolution

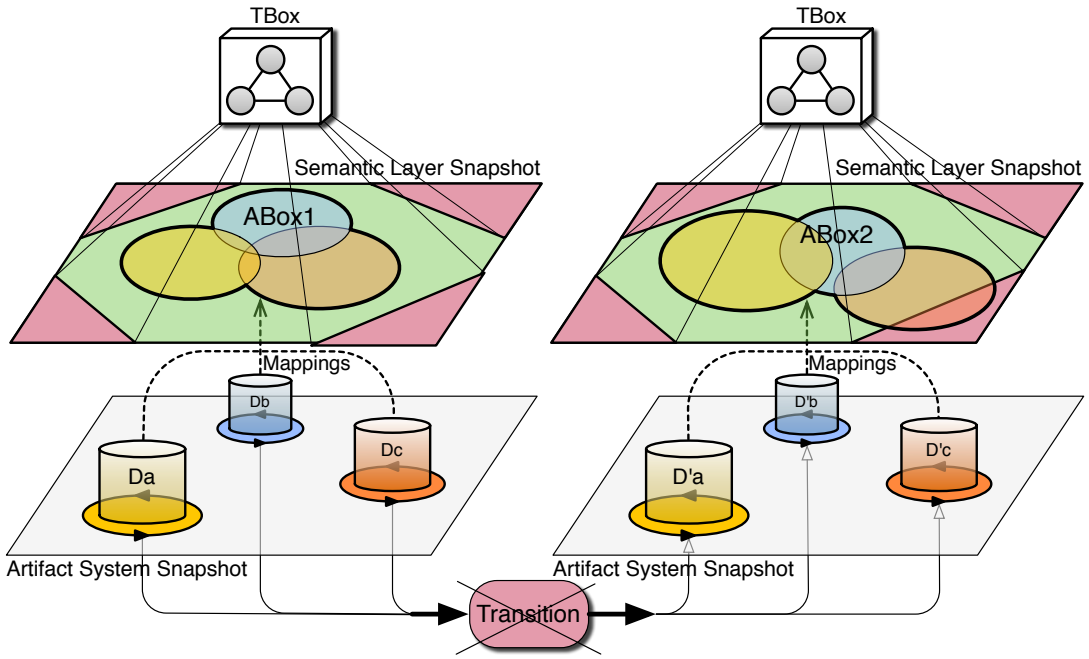


Figure 4 – Semantic Layer as a mean to govern an artifact system evolution

underlying artifact database schemas. This enables to verify whether processes specified over the Artifact Layer comply with such semantic dynamic laws.

4.2 Linking Data to Ontologies

In the last years, a new paradigm for information integration, called ontology-based data access (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 [42]. 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

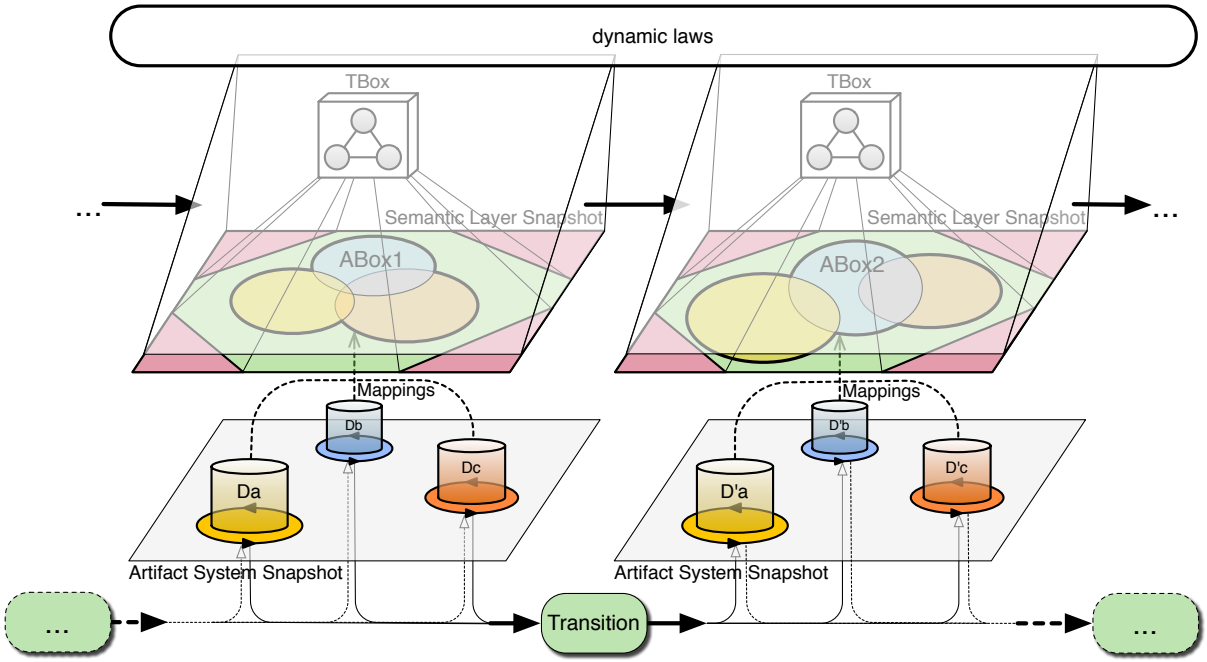


Figure 5 – Semantic Layer as a mean to govern an artifact system evolution

the data integration literature are the so-called GLAV assertions [38], which are expressions of the form $\phi(\vec{x}) \rightsquigarrow \psi(\vec{x})$, where $\phi(\vec{x})$ is a query over the data sources and $\psi(\vec{x})$ 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 $\phi(\vec{x})$ over the source database semantically correspond (in a sense that will be clarified below) to the formula $\psi(\vec{x})$, and therefore create the bridge between the data in the sources and the objects satisfying the predicates in the ontology. When the formula $\phi(\vec{x})$ is a single atom formula of the form $R(\vec{x})$, with R a source relational predicate, the mapping assertion is called *LAV (Local-As-View)*. When the formula $\psi(\vec{x})$ is a single atom formula of the form $S(\vec{x})$, with S an ontology predicate of arity n and $\vec{x}$ a sequence of n distinct variables, the mapping assertion is called *GAV (Global-As-View)*.

Formally, an *OBDA specification* is a triple $\mathcal{S} = \langle T, \mathcal{M}, \mathcal{D} \rangle$, where T is a DL TBox, $\mathcal{D}$ is a database, and $\mathcal{M}$ is a set of mappings between T and $\mathcal{D}$. Its semantics is given in terms of FOL interpretations I over the alphabet of T, such that (i) I satisfies T, and (ii) I satisfies $\mathcal{M}$, i.e., for each assertion $\phi(\vec{x}) \rightsquigarrow \psi(\vec{x})$ in $\mathcal{M}$ we have that for every tuple $\vec{t}$ in the evaluation of $\phi(\vec{x})$ over $\mathcal{D}$ it holds that $\psi(\vec{t})$ evaluates to true in I, where the notions of evaluation of $\phi(\vec{x})$ over $\mathcal{D}$ and $\psi(\vec{t})$ 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 $\vec{t}$ such that $\psi(\vec{t})$ evaluates to true in I it holds that $\psi(\vec{t})$ is in the evaluation of $\phi(\vec{x})$ over $\mathcal{D}$ (cf. [38]). 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 $\mathcal{S} = \langle T, \mathcal{M}, \mathcal{D} \rangle$ is a query posed over its TBox 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 ontologies.

Analogously to the language used to query the ontology, 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 $\phi(\vec{x})$ in a mapping assertion is posed over a database, where query answering actually amounts to simple query evaluation. We can therefore assume that such query is a generic FOL query (possibly expressed in SQL), whereas will consider $\psi(\vec{x})$, which is a query over the ontology, expressed in the language ECQ given above.

4.3 Semantically-Governed Artifact Systems: Definition

An SAS $\mathcal{S}$ is a tuple $\langle R, \mathcal{I}_0, \mathcal{F}, \mathcal{KA} \rangle$, where:

- R is the database schema of the artifact system's Relational Layer;
- $\mathcal{I}_0$ is the initial database instance of the Relational Layer, made up of values in $\mathcal{V}$ and conforming to R, which describes the starting state of the system;
- $\mathcal{F}$ is the transition relation that describes the overall progression mechanism of the Relational Layer;
- $\mathcal{KA}$ is the KAB that describes the Semantic Layer of the system.

The KAB $\mathcal{KA}$, in turn, is defined by the pair $\langle \mathcal{K}, \Gamma \rangle$, where:

- $\mathcal{K} = \langle T, \mathcal{M} \rangle$ is the knowledge component of $\mathcal{KA}$, constituted by a *DL-Lite*$_\mathcal{A}$ TBox T and by a mapping specification $\mathcal{M}$ that contains GAV mapping assertions connecting R to T;
- Γ is a set of $\mu\mathcal{L}_A^{\text{EQL}}/\mu\mathcal{L}_P^{\text{EQL}}$ formulae declaratively describing the dynamic laws to which the executions of the system must obey.

Notice that the tuple $\mathcal{O} = \langle R, T, \mathcal{M} \rangle$ constitutes in fact an OBDA system. For an overall picture providing the intuition for SASs, see Figure 5.

4.4 Execution Semantics

The execution semantics of $\mathcal{S}$ can be captured in terms of two transition systems, one describing the allowed evolutions at the Relational Layer, and one abstracting them at the Semantic Layer. The first transition system has the form of an RTS (cf. Section 2.3), whereas the second has the form of an STS (cf. Section 3.3).

RTS. The RTS $\Upsilon_\mathcal{S}^\text{R}$ of the SAS $\mathcal{S}$ is formally defined as $\langle R, \Sigma, s_0, db, \Rightarrow \rangle$, where Σ, $\Rightarrow$ and db are defined by simultaneous induction as the smallest sets such that $s_0 \in \Sigma$, with $db(s_0) = \mathcal{I}_0$, and satisfying the following property. Given $s \in \Sigma$, consider every database instance $\mathcal{I}'$ such that $\langle db(s), \mathcal{I}' \rangle \in \mathcal{F}$. Then:

1. if there exists $s' \in \Sigma$ such that $db(s') = \mathcal{I}'$, then $s \Rightarrow s'$;
2. otherwise, if $\mathcal{O} = \langle R, T, \mathcal{M} \rangle$ is *satisfiable* wrt $\mathcal{I}'$, then $s' \in \Sigma$, $s \Rightarrow s'$ and $db(s') = \mathcal{I}'$, where s' is a fresh state.

We observe that the satisfiability check done in the last step of the RTS construction accounts for *semantic governance*.

STS. The STS $\Upsilon_\mathcal{S}^\text{S}$ of $\mathcal{S}$ is an STS $\langle T, \Sigma, s_0, abox, \Rightarrow \rangle$, where T is the TBox of $\mathcal{S}$. In particular, $\Upsilon_\mathcal{S}^\text{S}$ is defined as a "virtualization" of the RTS $\Upsilon_\mathcal{S}^\text{R} = \langle R, \Sigma, s_0, db, \Rightarrow \rangle$ at the Semantic Layer: it maintains the structure of $\Upsilon_\mathcal{S}^\text{R}$ unaltered, reflecting that the progression of the system is determined at the Relational Layer, but it associates each state to a virtual ABox obtained from the application of the mapping specification $\mathcal{M}$ to the database instance associated by $\Upsilon_\mathcal{S}^\text{R}$ to the same state. Formally, the *abox* function of $\Upsilon_\mathcal{S}^\text{S}$ is defined as follows: for each $s \in \Sigma$ with $db(s) = \mathcal{I}$, $abox(s) = \mathcal{M}(\mathcal{I})$. Figure 6 provides a pictorial intuition of the two transition systems and their interconnection.

The STS $\Upsilon_\mathcal{S}^\text{S}$ allows us to define the *verification* problem for a SAS, i.e., verifying whether a $\mu\mathcal{L}_P^{\text{EQL}}/\mu\mathcal{L}_A^{\text{EQL}}$ *closed formula* Φ *holds for the SAS* $\mathcal{S}$: $\Upsilon_\mathcal{S}^\text{S} \models \Phi$. This, in turn, allows us to tackle the *conformance* problem of $\Upsilon_\mathcal{S}^\text{S}$ wrt the dynamic laws Γ of a KAB $\mathcal{KA} = \langle \mathcal{K}, \Gamma \rangle$: $\Upsilon_\mathcal{S}^\text{S}$ conforms to Γ if

$$\Upsilon_\mathcal{S}^\text{S} \models \bigwedge_{\Phi_i \in \Gamma} \Phi_i$$

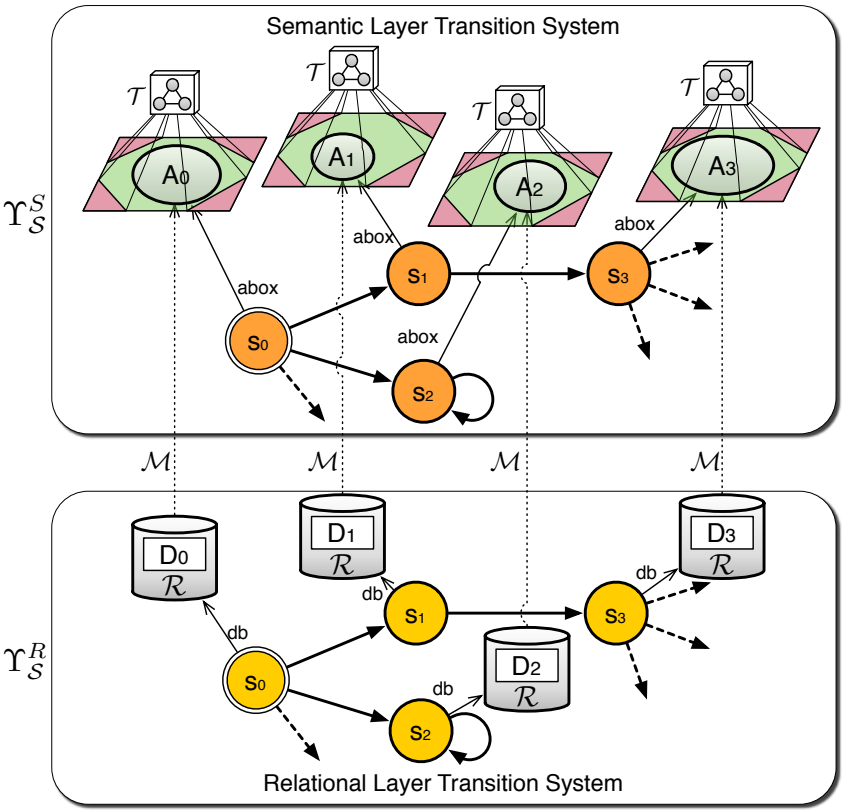


Figure 6 – Relationship between the RTS and STS of an SAS $\mathcal{S}$

4.5 Compilation of Semantic Constraints

The construction of the RTS $\Upsilon^{\mathrm{R}}_{\mathcal{S}}$ involves the use of the KAB's TBox T for semantic governance, i.e., to check that the states of the transition system, projected through the mapping specification, satisfy the semantic constraints of T. In this Section we argue that, thanks to the FOL rewritability of *DL-Lite*$_{\mathcal{A}}$, it is possible to compile away the KAB and rewrite the consistency check into a denial constraint directly posed over R in the Relational Layer. This can be done in three steps:

1. the formula $\mathsf{q}_{\mathsf{unsat}}(T)$ is constructed, following the reduction described in [13, 18];
2. such a formula is unfolded by using the mapping assertion $\mathcal{M}$, and the result is used to build a denial constraint of the form $\textsc{unfold}(\mathsf{q}_{\mathsf{unsat}}(T), \mathcal{M}) \rightarrow \mathsf{false}$;
3. the denial constraint is incorporated into the transition relation $\mathcal{F}$, which now also enforces that two snapshots $\langle \mathcal{I}, \mathcal{I}' \rangle \in \mathcal{F}$ only if both $\mathcal{I}$ and $\mathcal{I}'$ satisfy $\textsc{unfold}(\mathsf{q}_{\mathsf{unsat}}(T), \mathcal{M}) \rightarrow \mathsf{false}$.

If we assume that the transition relation of the Relational Layer is induced by a DCDS, this translation shows that the T can effectively be compiled away, and embedded into the constraints of the DCDS data layer.

4.6 Rewriting and Unfolding of Dynamic Laws

We now describe how $\mu\mathcal{L}^{\mathrm{EQL}}_P$ properties can be processed, i.e., rewritten and then unfolded, in order to obtain corresponding $\mu\mathcal{L}_P$ properties that can be directly verified at the Relational Layer. The same line of reasoning can be applied to rewrite $\mu\mathcal{L}^{\mathrm{EQL}}_A$ properties into $\mu\mathcal{L}_A$ properties. We do so by extending the definition of rewriting and unfolding in OBDA [42, 18].

This allow us to redefine the conformance problem of an SAS as a verification problem over its Relational Transition System.

In the remainder of this section, we consider a an SAS $\mathcal{S} = \langle R, \mathcal{I}_0, \mathcal{F}, \mathcal{KA} \rangle$, where $\mathcal{KA} = \langle \mathcal{K}, \Gamma \rangle$, and $\mathcal{K} = \langle T, \mathcal{M} \rangle$.

Rewriting of Dynamic Laws The KAB's TBox T does not focus on the dynamics of the system, but only on the information maintained by $\mathcal{S}$. Therefore, for the rewriting of a $\mu\mathcal{L}_P^{\text{EQL}}$ formula Φ wrt the TBox T we separate the treatment of the dynamic part and of the embedded ECQs. The rewriting is then done by maintaining the dynamic part unaltered, and by replacing each embedded ECQ with its corresponding rewriting wrt T.

Unfolding of Dynamic Laws The unfolding of $\mu\mathcal{L}_P^{\text{EQL}}$ dynamic laws is applied after the rewriting, i.e., after having compiled away the KAB's TBox. As for the rewriting, we separate the treatment of the dynamic part and of the embedded ECQs.

As described in Section 4.4, the execution semantics of $\mathcal{S}$ is defined in such a way that both the STS $\Upsilon_{\mathcal{S}}^{\text{S}}$ and the RTS $\Upsilon_{\mathcal{S}}^{\text{R}}$ have the same structure. This reflects the intuition that the dynamics of the artifact system is determined at the Relational Layer. Therefore, the unfolding of a $\mu\mathcal{L}_P^{\text{EQL}}$ formula maintains its dynamic part unaltered, and unfolds all the embedded ECQs wrt the mapping assertion $\mathcal{M}$. However, two cases deserve a discussion:

- the unfolding of $\textsc{live}(x)$;
- the unfolding of quantification across states, i.e., $\exists x.\textsc{live}(x) \wedge \Phi$.

As far as $\textsc{live}(x)$ is concerned, we observe that $\textsc{live}(x)$ corresponds to the following equivalent UCQ, which can then be unfolded in the usual way:

$$\textsc{live}(x) \equiv \bigvee_{N \text{ in } T} N(x) \vee \bigvee_{P \text{ in } T} (P(x, \_) \vee P(\_, x))$$

Quantification across states is dealt with in the same way as quantification in ECQs:

$$\textsc{unfold}(\exists x.\textsc{live}(x) \wedge \Phi, \mathcal{M}) = \bigvee_{(f/n) \in \text{FS}(\mathcal{M})} \exists x_1, \ldots, x_n.\textsc{unfold}(\textsc{live}(f(x_1, \ldots, x_n)), \mathcal{M}) \\ \wedge \textsc{unfold}(\Phi[x/f(x_1, \ldots, x_n)], \mathcal{M})$$

Notice that such unfolding technique guarantees that the unfolded formula is indeed a $\mu\mathcal{L}_P$ formula. In the case of unfolding, a $\mu\mathcal{L}_P^{\text{EQL}}$ formula over the TBox T (Semantic Layer) of the SAS under study is translated into a $\mu\mathcal{L}_P$ formula over the SAS relational schema R (Relational Layer). To ensure that the unfolded formula is in $\mu\mathcal{L}_P$, we observe that $\textsc{unfold}(\textsc{live}(f(x_1, \ldots, x_n)), \mathcal{M})$ implies that $\textsc{live}(x_1) \wedge \cdots \wedge \textsc{live}(x_n)$. In fact, an object term that is live in a given state of the STS can be only produced starting from live values in the corresponding state of the RTS. Consequently, even if $\textsc{live}(x_1), \ldots \textsc{live}(x_n)$ are not syntactically produced by the unfolding of $\textsc{live}(f(x_1, \ldots, x_n))$, they are implied by the obtained formula.

This approach shows that the verification problem for SASs can be reduced to a verification problem directly posed over the Relational Layer only. In particular, when the dynamics of the Relational Layer is specified by means of a DCDS, we can take advantage from this translation, and from the TBox compilation approach discussed in the previous section, to reduce the verification problem for an SAS into a verification problem for the underlying DCDS (extended with the denial constraint obtained from the TBox compilation). This, in turn, allows to exploit the decidability results shown in Section 2.7 also in this extended setting.

Part II
KAB Instantiation: the Case of GSM

We now draw a correspondence between the formal framework of DCDSs, and the realization of the ACSI Artifact Layer in terms of the Guard-Stage-Milestone (GSM) artifact model [35, 25]. The choice of DCDSs is motivated by the fact that GSM specifications rely on a relational information model (with nested tuples).

On the one hand, this correspondence shows that DCDSs are expressive enough to capture virtually any artifact-centric concrete model. On the other hand, it can be exploited to attack the verification problem for GSM-based artifacts, taking advantage from the decidability results established for GSM.

This part of the document of is organized as follows. To make the document self-contained, in Section 5 provide an introduction of the GSM model, also discussing a running example that consists of an excerpt extracted from the ACSI Energy use case. In Section 6, we discuss that verification of GSM-based artifacts is a very challenging task, which can be immediately shown to be undecidable in the general case. We then move in Section 7 to the correspondence between GSM and DCDSs, introducing a translation mechanism that, given a GSM model, produces a corresponding faithful DCDS. In Section 8, we finally exploit this translation to provide decidability results on the verification of state-bounded GSM models, providing at the same time design guidelines on how to turn arbitrary GSM models into state-bounded models.

5 The Guard-Stage-Milestone 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.

5.1 Informal Introduction

The GSM artifact modeling language recently introduced by [35, 25] 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 components are:

- *data schema* (Att) for modeling relevant data domain (data attributes) and keeping track of progresses of artifact (status attributes);
- *milestones* (Mst), which correspond to business operational objectives and are achieved based on triggering events and/or conditions over the data schema;
- *stages* (Stg), which correspond to clusters of activities intended to achieve milestones, and which can have a hierarchical structure.
- *guards* (Grd), which control when a stage can be activated for execution.
- *events*, which describe interaction with the environment.

The GSM data schema uses (possibly nested) attribute/value pairs to capture the domain of interest. It distinguishes *data attributes*, which represent data relevant to the business, and *status attributes*, which hold information about the progress of the artifact instance along its lifecycle.

The description of a particular business process may involve several *instances* of artifacts described by a GSM schema. At any point in time, the state of any given artifact instance (*snapshot*) 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).

The key elements of a lifecycle schema are *stages*, *milestones* and *guards*. Stages may be organized into a hierarchy, as they can be either *atomic* or *composite*. An atomic stage contains exactly one *task*, which corresponds to a unit of business-relevant work that is to be performed

by an external agent (either human or machine). A composite stage contains 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 control the activation of stages and, like milestones, are described in terms of data-aware expressions, called *sentries*, 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 represent the atomic units of work, which are used to update the data schema of an artifact instance. Artifact instances may interact with the external world by exchanging typed *events*. In fact, *tasks* are considered to be performed by an external agent, and their corresponding execution is captured with two event types: an *invocation*, whose instances are populated by the data from data schema and then sent to the environment; and a *termination*, whose instances represent the corresponding answer from the environment and are used to incorporate the obtained result back into the artifact data schema. The environment can also issue unsolicited (one-way) events, to trigger specific guards or milestones.

We also consider a specific type of events: *status events*, 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.

The operational semantics for GSM is specified in terms of how a single event is incorporated into the current snapshot. Incorporation of an event corresponds to processing of all the effects 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 status changes. A business step, or *B-step*, corresponds to a transition from a snapshot of the system (before processing the event) to a new one, resulting from the incorporation of the event. B-steps correspond to the smallest units of business-relevant change that can occur to a GSM system and consist of a terminating sequence of internal events, triggered by the event.

5.2 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, the sett 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 which includes the elements of the domain and the two boolean constants $true$ 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*
- $\Sigma(LastIncEventType) \in$ 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 language $\mathcal{L}$ that will be used for specifying the sentries. The syntax of $\mathcal{L}$ is formally presented in [39] 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, in order to evaluate a formula, we need to associate variables $(x_1 \dots x_n)$ to $\Delta \cup$ EVENT. Given a formula $\Phi \in \mathcal{L}$, we write $\Sigma \models \Phi(x_1 \dots x_n)$ when $\Phi(\Sigma(x_1) \dots \Sigma(x_n))$ evaluates to true accordingly to the semantics of $\mathcal{L}$.

The language $\mathcal{L}$ 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 $-a$. It is then clear that if a formula contains status events it is *temporal*, meaning that it can refer to different snapshots of the system. A temporal formula 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 to elements in Δ according to Σ', and unprimed variables symbols for variables that should be associated to elements in Δ according to Σ. Formally, given a formula $\Phi(x_1 \dots x_n, x'_1 \dots x'_m)$ where $x_1 \dots x_n, x'_1 \dots x'_m \in Att$, the pair (Σ, Σ') satisfies Φ, denoted $(\Sigma, \Sigma') \models \Phi$ if $\Phi(x_1/\Sigma(x_1) \dots x_n/\Sigma(x_n), x'_1/\Sigma'(x_1) \dots x'_m/\Sigma'(x_m))$ evaluates to true, where $(x_i/\Sigma(x_i)$ substitues to x_i the value $\Sigma(x_i)$ in Φ. We call *local* a formula Φ with no primed attributes.

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;
  - $\mathsf{LastIncEventType} = E$ or
  - $\{+, -\}a$ for some status attribute $a \in att_s$.
- γ is a $\mathcal{L}$-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. As a first classification, we can distinguish between business events and status events. We already discussed status events above (and the way they can be used in sentries), so we now focus on businesses events. *Business events* are used in GSM to 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. Business events can be again separated into *one-way* and *two-way* 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

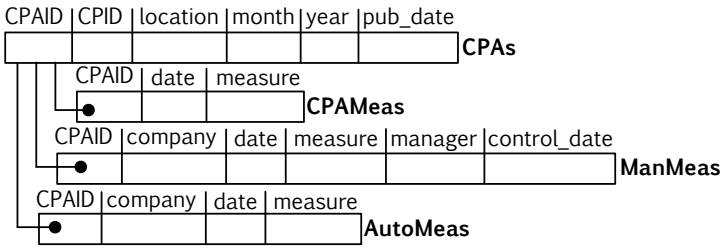


Figure 7 – Graphical representation of the *control point assessment* artifact data schema described in Example 5.1.

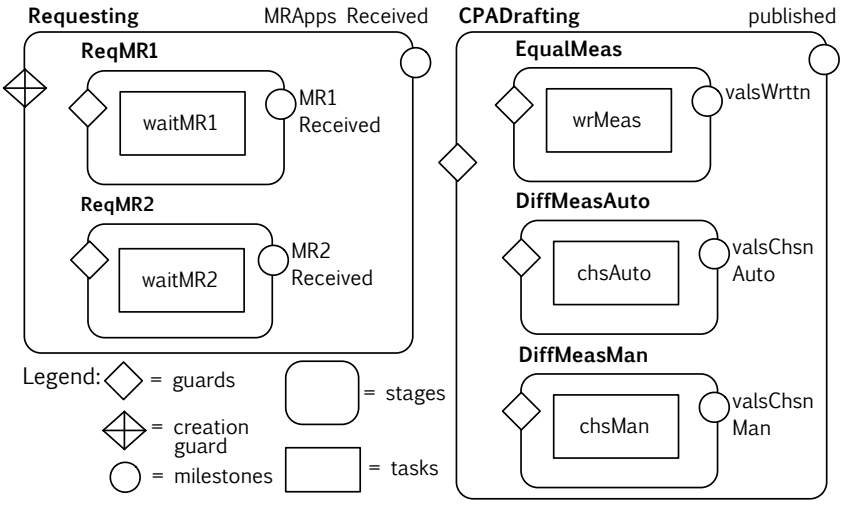


Figure 8 – Graphical representation of the *control point assessment* artifact lifecycle described in Example 5.1.

att_d, and ψ is a condition (a local formula in $\mathcal{L}$) 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.

Two-way events, instead, model a task invocation to the environment and its related response. 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 $\mathcal{L}$ 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 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.

5.3 An Example from the ACSI Energy Use Case

Next, we present our running example, which stems from the ACSI Energy use case.

Example 5.1. 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

	on	if
Requesting	$CPACreationEvent$	–
RequestMR1	+Requesting	–
RequestMR2	+Requesting	–
CPADrafting	+MRAppsReceived	–
EqualMeas	+CPADrafting	$\exists id, c1, c2, d, m.CPAs(id, -, -, -, -, -) \wedge c1 \neq c2 \wedge$ $((ManMeas(id, c1, d, m, -, -) \wedge ManMeas(id, c2, d, m, -, -)) \vee$ $(Auto\_meas(id, c1, d, m) \wedge Auto\_meas(id, c2, d, m)) \vee$ $(ManMeas(id, c1, d, m, -, -) \wedge AutoMeas(id, c2, d, m)))$
DiffMeasAuto	+CPADrafting	$\exists id, c1, c2, d, m.CPAs(id, -, -, -, -, -) \wedge c1 \neq c2 \wedge m1 \neq m2 \wedge$ $ManMeas(id, c1, d, m1, -, -) \wedge AutoMeas(id, c2, d, m2)$
DiffMeasMan	+CPADrafting	$\exists id, c1, c2, d, m.CPAs(id, -, -, -, -, -) \wedge c1 \neq c2 \wedge m1 \neq m2 \wedge$ $((ManMeas(id, c1, d, m1, -, -) \wedge ManMeas(id, c2, d, m2, -, -)) \vee$ $(AutoMeas(id, c1, d, m1) \wedge AutoMeas(id, c2, d, m2)))$

Table 3 – Guards for the lifecycle in Figure 8.

	on	if
MR1Received	$WaitMR1TermEvent$	–
MR2Received	$WaitMR2TermEvent$	–
MRAppsReceived	–	MR1Received ∧ MR2Received
valsWrttn	$wrMeasTermEvent$	–
valChsnAuto	$chsAutoTermEvent$	–
valChsnMan	$chsManTermEvent$	–
published	+valsWrttn ∨ +valsChnsAuto∨ +valsChsnMan	(valsWrttn ∨ ¬EqualMeas) ∧ (valsChsnAuto ∨ ¬DiffMeasAuto) ∧(valsChsnMan ∨ ¬DiffMeasMan)

Table 4 – Milestones for the lifecycle in Figure 8.

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 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 Figure 7 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 chosen by the system operator for each day of the month. This value is always among the two values which have been proposed (in the monthly report application) by the two companies connected to the control point. Later on we will 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 company name, measure, date it refers to, manager in charge of the manual input and the date of input. Automatically determined values, on the other side, are stored in the $AutoMeas$, in which company, date and measure are the only relevant information.

Figure 8 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 two-way 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 Figure 7) and milestone MR1Received (or MR2Received) is achieved. When both ReqMR1 and ReqMR2 closes, 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 value is written in table $CPAMeas$. In other words, task wrMeas takes care of writing values of 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 value to be written in $CPAMeas$. The last substage, DiffMeasMan covers the case in which the measurements disagree and they are both been performed automatically or manually. A manual inspection is then needed in order to choose one of those values.

Table 3 and 4 show sentries for guards and milestone for the energy example, respectively. Such sentries are expressed as first order logic formulas. ■

6 Undecidability of GSM Verification

In this section, we show that verifying the infinite-state transition system representing the execution semantics of a given GSM model is an extremely challenging problem, undecidable even for a very simple propositional reachability property.

Theorem 6.1. *There exists a GSM model for which verification of a propositional reachability property is undecidable.*

Proof. To show undecidability of verification, we illustrate that a Turing machine can be easily captured in GSM, and that the halting problem can be stated in terms of a verification problem. In particular, we consider a deterministic, single tape Turing machine $\mathcal{M} = \langle Q, \Sigma, q_0, \delta, q_f, \textvisiblespace \rangle$, where Q is a finite set of (internal) states, $\Sigma = \{0, 1, \textvisiblespace\}$ is the tape alphabet (with $\textvisiblespace$ the blank symbol), $q_0 \in Q$ and $q_f \in Q$ are the initial and final state, and $\delta \subseteq Q \setminus \{q_f\} \times \Sigma \times Q \times \Sigma \times \{L, R\}$ is a transition relation. We assume, wlog, that δ consists of k right-shift transitions $R_1, \ldots, R_k$ (those having R as last component), and n left-shift transitions $L_1, \ldots, L_n$ (those having L as last component). The idea of translation into a GSM model is the following. Beside status attributes, the GSM information model is constituted by: *(i)* a *curState* slot containing the current internal state $q \in Q$; *(ii)* a *curCell* slot pointing to the cell where the head of $\mathcal{M}$ is currently located. *(iii)* a collection of *cells* representing the current state of the tape. Each cell is a complex nested record constituted by a value $v \in \Sigma$, and two pointers *prev* and *next* used to link the cell to the previous and next cells. In this way, the tape is modeled as a linked list, which initially contains a single, blank cell, and which is dynamically extended as needed. To mark the initial (resp., last) cell of the tape, we assume that its *prev* (*next*) cell is *null*.

On top of this information model, a GSM lifecyle that mimics $\mathcal{M}$ is shown in Figure 9, where, due to space constraints, only the right-shift transitions are depicted (the left-shift ones are symmetric). The schema consists of two top-level stages. *Init* stage is used to initialize the tape. *Transition* stage is instead used to mimic the execution of one of the transitions in δ. Each transition is decomposed into two sub-stages: *state update* and *head shift*. The state update is modeled by one among $k + n$ atomic sub-stages, each handling the update that corresponds to one of the transitions in δ. These stages are mutually exclusive, being $\mathcal{M}$ deterministic. Consider for example a right-shift transition $R_i = \delta(qR_i, vR_i, qR_i', vR_i', R)$ (the treatment is similar for a left-shift transition). The corresponding state update stage is opened whenever the current state is qR_i, and the value contained in the cell pointed by the head is vR_i (this can be extracted from the information model using the query $curCell.value$). The incoming arrows from the two parent's guards ensures that this condition is evaluated as soon as the parent stage is opened;

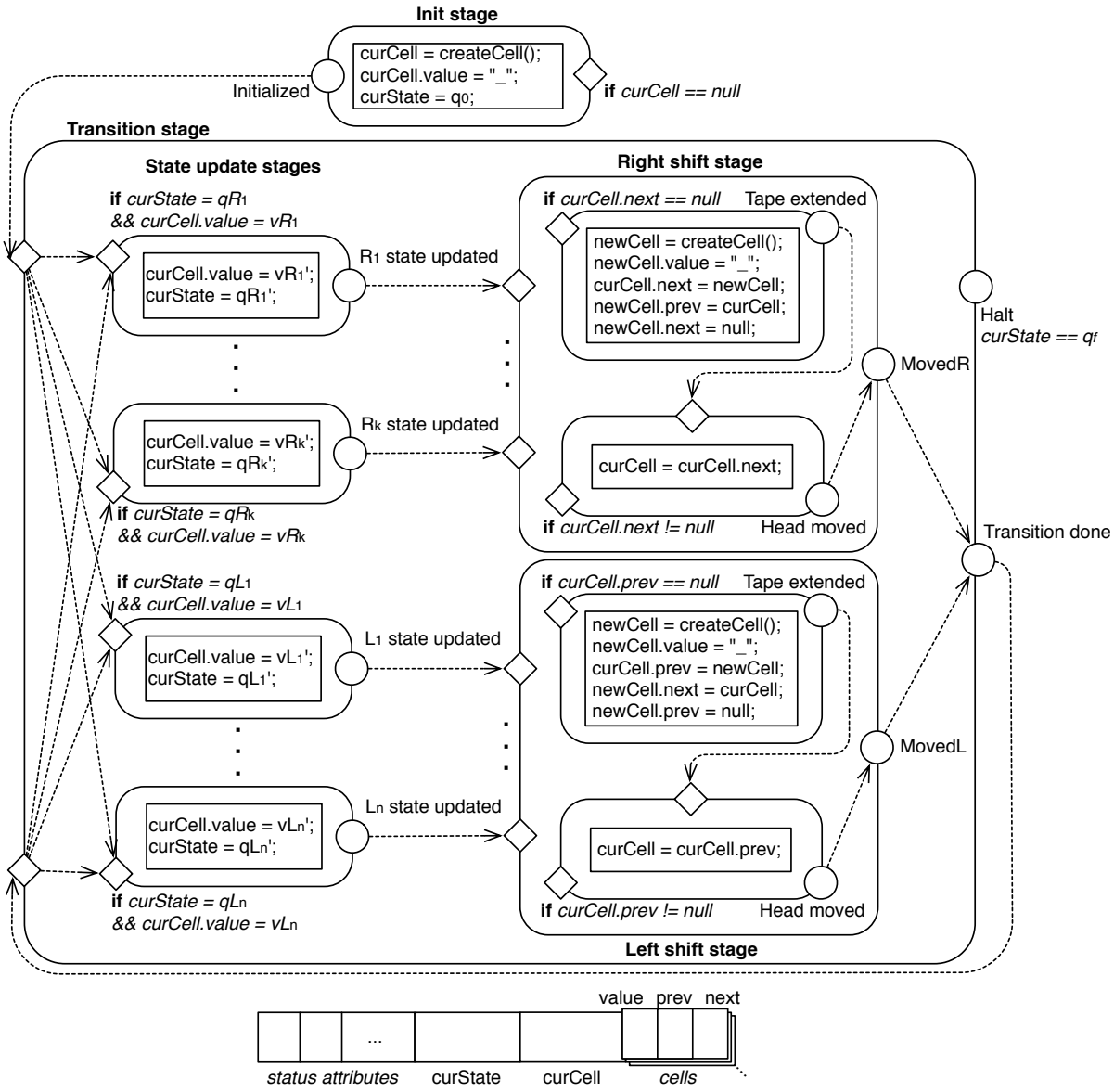


Figure 9 – GSM model of a Turing machine

hence, if the condition is true, the state update stage is immediately executed. When the state update stage is closed, the achievement of the corresponding milestone triggers one of the guards of the Right shift stage that handles the head shift. It contains two sub-stages: the first one extends the tape if the head is currently pointing to the last cell, while the second one just perform the shifting. Whenever a right or left shift stage achieves the corresponding milestone, then also the parent, transition stage is closed, achieving milestone "Transition done". This has the effect of re-opening the transition stage again, so as to evaluate the next transition to be executed. An alternative way of immediately closing the transition stage occurs when the current state corresponds to the final state q_f. In this case, milestone "Halt" is achieved, and the execution terminates (no further guards are triggered).

By considering this construction, the halting problem for $\mathcal{M}$ can be rephrased as the following verification problem: given the GSM model encoding $\mathcal{M}$, and starting from an initial state where the information model is empty, is it possible to reach a state where the "Halt" milestone is achieved? Notice that, since $\mathcal{M}$ is deterministic, the B-steps of the corresponding GSM model constitute a linear computation, which could eventually reach the "Halt" milestone or continue indefinitely. Therefore, reaching a state where "Halt" is achieved can be equivalently formulated using propositional CTL or LTL. □

7 Translating GSM into DCDSs

In this section we propose a translation procedure that takes a GSM model and produces a corresponding faithful representation in terms of DCDSs. This allows us to transfer the decidability boundaries studied for DCDSs to the GSM context. We only discuss the intuition behind the translation and provide the main results. Appendix A reports the full technical development and proofs.

The translation relies on the incremental semantics: given a GSM model $\mathcal{G}$, we encode each possible micro-step as a separate condition-action rule in the process of a corresponding DCDS system $\mathcal{S}$, such that the effect on the data and process layers of the action coincides with the effect of the corresponding micro-step in GSM. However, in order to guarantee that the transition system induced by the resulting DCDS mimics the one of the GSM model, the translation procedure should also ensure that all semantic requirements assumed for GSM [25] are modeled properly: *(i)* "one-message-at-a-time" and "toggle-once" principles, *(ii)* the finiteness of micro-steps within a B-step, and *(iii)* their order imposed by the model. We sustain these requirements by introducing into the data layer of $\mathcal{S}$ a set of auxiliary relations, suitably recalling them in the CA-rules to reconstruct the desired behaviour.

Restricting $\mathcal{S}$ to process only one incoming message at a time is implemented by the introduction of a *blocking mechanism*, represented by an auxiliary relation $R_{block}(id_R, blocked)$ for each artifact in the system, where id_R is the artifact instance identifier, and *blocked* is a boolean flag. This flag is set to *true* upon receiving an incoming message, and is then reset to $false$ at the termination of the corresponding B-step, once the outgoing events accumulated in the B-step are sent the environment. If an artifact instance has $blocked = true$, no further incoming event will be processed. This is enforced by checking the flag in the condition of each CA-rule associated to the artifact.

In order to ensure "toggle once" principle and guarantee the finiteness of sequence of micro-steps triggered by an incoming event, we introduce an *eligibility tracking mechanism*. This mechanism is represented by an auxiliary relation $R_{exec}(id_R, x_1, ..., x_c)$, where c is the total number of PAC-rules, and each x_i corresponds to a certain PAC-rule of the GSM model. Each x_i encodes whether the corresponding PAC rule is eligible to fire at a given moment in time (i.e., a particular micro-step). The initial setup of the eligibility tracking flags is performed at the beginning of a B-step, based on the evaluation of the prerequisite condition of each PAC rule. More specifically, when $x_i = 0$, the corresponding CA-rule is eligible to apply and has not yet been considered for application. When instead $x_i = 1$, then either the rule has been fired, or its prerequisite turned out to be false. This flag-based approach is used to propagate in a compact way information related to the PAC rules that have been already processed, following a mechanism that resembles *dead path elimination* in BPEL. In fact, R_{exec} is also used to enforce a firing order of CA-rules that follows the one induced by $\mathcal{G}$. This is achieved as follows. For each CA-rule $Q \mapsto \alpha$ corresponding to a given PAC rule r, condition Q is put in conjunction with a further formula, used to check whether all the PAC rules that precede r according to the ordering imposed by $\mathcal{G}$ have been already processed. Only in this case r can be considered for application, consequently applying its effect α to the current artifact snapshot. More specifically, the corresponding CA-rule becomes $Q \wedge exec(r) \mapsto \alpha$, where $exec(r) = \bigwedge_i x_i$ such that i ranges over the indexes of those rules that precede r.

Once all x_i flags are switched to 1, the B-step is about to finish: a dedicated CA-rule is enabled to send the outgoing events to the environment, and the artifact instance *blocked* flag is released.

Example 7.1. An example of a translation of a GSM PAC-rule (indexed by k) is presented in Figure 10. For simplicity, multiple parameters are compacted using an "array" notation (e.g., $x_1, \ldots, x_n$ is denoted by $\overline{x}$). In particular: (1) represents a condition part of a CA-rule, ensuring the "toggle-once" principle ($x_k = 0$), the compliant firing order ($exec(k)$) and the "one-message-at-a-time" principle ($R_{block}(id_R, true)$); (2) describes the action signature; (3) is an effect encoding the invalidation a milestone once the stage has been activated; (4) propagates

$$R_{exec}(id_R, \overline{x}) \wedge x_k = 0 \wedge exec(k) \wedge R_{block}(id_R, true) \mapsto \quad (2)$$

$$a^k_{exec}(id_R, \overline{a}', \overline{x}) : \{ \quad (3)$$

$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{S_j}_{chg}(id_R, true) \rightsquigarrow \{R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[m_j/false]\} \quad (4)$$

$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{S_j}_{chg}(id_R, true) \rightsquigarrow \{R^{m_j}_{chg}(id_R, false)\} \quad (5)$$

$$R^M_{exec}(id_R, \overline{x}) \wedge x_k = 0 \rightsquigarrow \{R^M_{exec}(id_R, \overline{x})[x_k/1]\} \quad (6)$$

$$[\mathsf{CopyMessagePools}], [\mathsf{CopyRest}] \quad \} \quad (7)$$

Figure 10 – CA-rule encoding a milestone invalidation upon stage activation

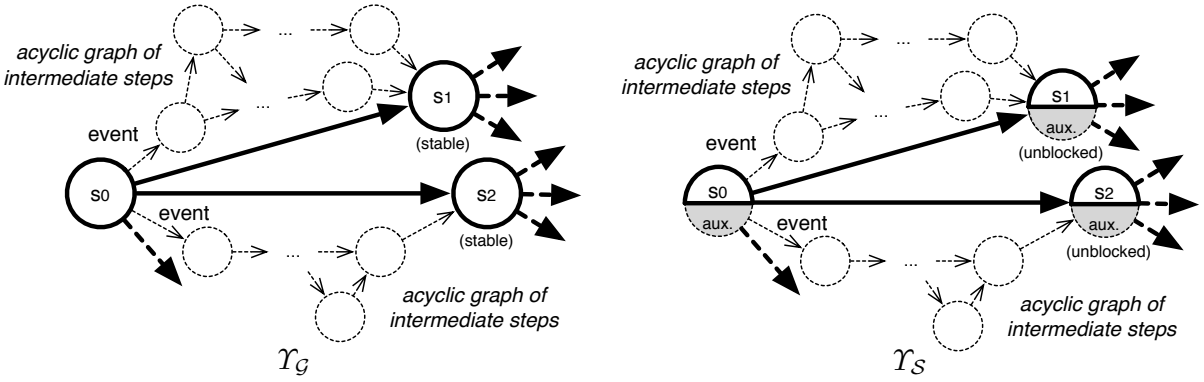


Figure 11 – Construction of the B-step transition system $\Upsilon_{\mathcal{G}}$ and unblocked-state transition system $\Upsilon_{\mathcal{S}}$, respectively for a GSM model $\mathcal{G}$ with initial snapshot s_0, and for the corresponding DCDS $\mathcal{S}$

an internal event denoting the milestone invalidation, if needed; (5) flags the encoded micro-step corresponding to PAC rule k as processed; (6) transports the unaffected data into the next snapshot.

Given a GSM model $\mathcal{G}$ with initial snapshot S_0, we denote by $\Upsilon_{\mathcal{G}}$ its *B-step transition system*, i.e., the infinite-state transition system obtained by iteratively applying the incremental GSM semantics starting from S_0 and nondeterministically considering each possible incoming event. The states of $\Upsilon_{\mathcal{G}}$ correspond to stable snapshots of $\mathcal{G}$, and each transition corresponds to a B-step. We abstract away from the single micro-steps constituting a B-step, because they represent temporary intermediate states that are not interesting for verification purposes. Similarly, given the DCDS $\mathcal{S}$ obtained from the translation of $\mathcal{G}$, we denote by $\Upsilon_{\mathcal{S}}$ its *unblocked-state transition system*, obtained by starting from S_0, and iteratively applying nondeterministically the CA-rules of the process, and the corresponding actions, in all the possible ways. As for states, we only consider those database instances where all artifact instances are not blocked; these correspond in fact to stable snapshots of $\mathcal{G}$. We then connect two such states provided that there is a sequence of (intermediate) states that lead from the first to the second one, and for which at least one artifact instance is blocked; these sequence corresponds in fact to a series of intermediate-steps evolving the system from a stable state to another stable state. Finally, we project away all the auxiliary relations introduced by the translation mechanism, obtaining a *filtered* version of $\Upsilon_{\mathcal{S}}$, which we denote as $\Upsilon_{\mathcal{S}}|_{\mathcal{G}}$. The intuition about the construction of these two transition systems is given in Figure 11. Notice that the intermediate micro-steps in the two transition systems can be safely abstracted away because: *(i)* thanks to the toggle-once principle, they do not contain any "internal" cycle; *(ii)* respecting the firing order imposed by $\mathcal{G}$, they all lead to reach the same next stable/unblocked state. We can then establish the one-to-one correspondence between these two transition systems in the following theorem (see Appendix A for a complete proof):

Theorem 7.1. *Given a GSM model $\mathcal{G}$ and its DCDS translation $\mathcal{S}$, the corresponding B-step transition system $\Upsilon_{\mathcal{G}}$ and filtered unblocked-state transition system $\Upsilon_{\mathcal{S}}|_{\mathcal{G}}$ are equivalent, i.e., $\Upsilon_{\mathcal{G}} \equiv \Upsilon_{\mathcal{S}}|_{\mathcal{G}}$.*

8 State-bounded GSM Models

We now take advantage of the key decidability results discussed in Section 2.7, and study verifiability of *state-bounded GSM models*. Observe that state-boundedness is not a too restrictive condition. It requires each state of the transition system to contain a bounded number of tuples. However, this does not mean that the system in general is restricted to encounter only a limited amount of data: infinitely many values may be distributed *across* the states (i.e. along an execution), provided that they do not accumulate in the same state. Furthermore, infinitely many executions are supported, reflecting that whenever an external event updates a slot of the information model maintained by a GSM artifact, infinitely many successor states in principle exist, each one corresponding to a specific new value for that slot. To exploit this, we have first to show that the GSM-DCDS translation preserves state-boundedness, which is in fact the case.

Lemma 8.1. Given a GSM model $\mathcal{G}$ and its DCDS translation $\mathcal{S}$, $\mathcal{G}$ is state-bounded if and only if $\mathcal{S}$ is state-bounded.

Proof. Recall that $\mathcal{S}$ contains some auxiliary relations, used to restrict the applicability of CA-rules in order to enforce the execution assumptions of GSM: *(i)* the eligibility tracking table R_{exec}, *(ii)* the artifact instance blocking flags R_{block}, *(iii)* the internal message pools $R^{msg_k}_{data}$, $R^{srv_p}_{data}$, $R^{msg_q}_{out}$, and *(iv)* the tables of status changes $R^{m_i}_{chg}$, $R^{s_j}_{chg}$. ($\Leftarrow$) This is directly obtained by observing that, if $\Upsilon_{\mathcal{S}}$ is state-bounded, then also $\Upsilon_{\mathcal{S}}|_{\mathcal{G}}$ is state-bounded. From Theorem A.4, we know that $\Upsilon_{\mathcal{S}}|_{\mathcal{G}} \equiv \Upsilon_{\mathcal{G}}$, and therefore $\Upsilon_{\mathcal{G}}$ is state-bounded as well.
($\Rightarrow$) We have to show that state boundedness of $\mathcal{G}$ implies that also all auxiliary relations present in $\Upsilon_{\mathcal{S}}$ are bounded. We discuss each auxiliary relation separately. The artifact blocking relation R_{block} keeps a boolean flag for each artifact instance, so its cardinality depends on the number of instances in the model. Since the model is state-bounded, the number of artifact instances is bounded and so is R_{block}. The eligibility tracking table R_{exec} stores for each artifact instance a boolean vector describing the applicability of a certain PAC rule. Since the number of instances is bounded and so is the set of PAC rules, then the relation R_{exec} is also bounded. Similarly, one can show the boundedness of $R^{m_i}_{chg}$, $R^{s_j}_{chg}$ due to the fact that the number of stages and milestones is fixed a-priori. Let us now analyze internal message pools. By construction, $\mathcal{S}$ may contain at most one tuple in $R^{msg_k}_{data}$ and $R^{srv_p}_{data}$ for each artifact instance. This is enforced by the blocking mechanism R_{block}, which blocks the artifact instance at the beginning of a B-step and prevents the instance from injecting further events in internal pools. The outgoing message pool $R^{msg_q}_{out}$ may contain as much tuples per artifact instance as the amount of atomic stages in the model, which is still bounded. However, neither incoming nor outgoing messages are accumulated in the internal pool along the B-steps execution, since the final micro-step of the B-step is designed not to propagate any of the internal message pools to the next snapshot. Therefore, $\Upsilon_{\mathcal{S}}$ is state-bounded. □

By combining the decidability result stating that verification of $\mu\mathcal{L}_P$ properties over state-bounded DCDSs is decidable, with Theorem A.4 and Lemma 8.1, we directly obtain:

Theorem 8.2. *Verification of $\mu\mathcal{L}_P$ properties over state-bounded GSM models is decidable, and can be reduced to finite-state model checking of propositional μ-calculus.*

Obviously, in order to guarantee verifiability of a given GSM model, we need to understand whether it is state-bounded or not. However, as discussed in Section 2.7, state-boundedness is a "semantic" condition, which is undecidable to check. We mitigate this problem by isolating a class of GSM models that is guaranteed to be state-bounded. We show however that even very simple GSM models (such as the one showin in Figure 12), are not state-bounded, and thus we provide some modelling strategies to lift an arbitrary GSM model into a state-bounded model.

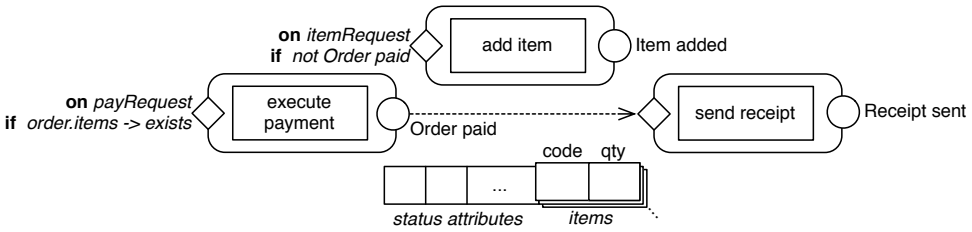


Figure 12 – GSM model of a simple order management process

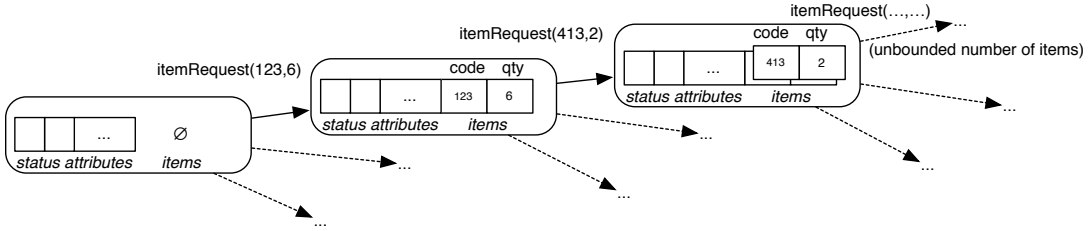


Figure 13 – Unbounded execution of the GSM model in Fig. 12

8.1 GSM Models without Artifact Creation

We investigate the case of GSM models that do not contain any *create-artifact-instance* tasks. Without loss of generality, we assimilate the creation of nested datatypes (such as those created by the "add item" task in Figure 12) to the creation of new artifacts. From the formal point of view, we can in fact consider each nested datatype as a simple artifact with an empty lifecycle, and its own information model including a connection to its parent artifact.

Corollary 1. *Verification of $\mu\mathcal{L}_P$ properties over GSM models without* create-artifact-instance *tasks is decidable.*

Proof. Let $\mathcal{G}$ be a GSM model without *create-artifact-instance* tasks. At each stable snapshot Σ_k, $\mathcal{G}$ can either process an event representing an incoming one-way message, or the termination of a task. We claim that the only source of state-unboundedness can be caused by service calls return related to the termination of *create-artifact-instance* tasks. In fact, one-way incoming messages, as well as other service call returns, do not increase the size of the data stored in the GSM information model, because the payload of such messages just substitutes the values of the corresponding data attributes, according to the signature of the message. Similarly, by an inspection of the proof of Lemma 8.1, we know that across the micro-steps of a B-step, status attributes are modified but their size does not change. Furthermore, a bounded number of outgoing events could be accumulated in the message pools, but this information is then flushed at the end of the B-step, thus bringing the size of the overall information model back to the same size present at the beginning of the B-step. Therefore, without *create-artifact-instance* tasks, the size of the information model in each stable state is constant, and corresponds to the size of the initial information model. We can then apply Theorem 8.2 to get the result. □

8.2 Arbitrary GSM Models

The types of models studied in paragraph above are quite restrictive, because they forbid the possibility of extending the number of artifacts during the execution of the system. On the other hand, as soon as this is allowed, even very simple GSM models, as the one shown in Fig. 12, may become state unbounded. In that example, the source of state unboundedness lies in the stage containing the "add item" task, which could be triggered an unbounded number of times due to continuous *itemRequest* incoming events, as pointed out in Fig. 13. This, in turn, is caused by the fact that the modeler left the GSM model underspecified, without providing any hint about the maximum number of items that can be included in an order. To

overcome this issue, we require the modeler to supply such information (stating, e.g., that each order is associated to at most 10 items). Technically, the GSM model under study has to be parameterized by an arbitrary but finite number N_{max}, which denotes the maximum number of artifact instances that can coexist in the same execution state. We call this kind of GSM model *instance bounded*. A possible policy to provide such bound is to allocate available "slots" for each artifact type of the model, i.e. to specify a maximum number N_{A_i} for each artifact type A_i, then having $N_{max} = \sum_i N_{A_i}$. In order to incorporate the artifact bounds into the execution semantics, we proceed as follows. First, we pre-populate the initial snapshot of the considered GSM instance with N_{max} blank artifact instances (respecting the relative proportion given by the local maximum numbers for each artifact type). We refer to one such blank artifact instance as *artifact container*. Along the system execution, each container may be: *(i)* filled with concrete data carried by an actual artifact instance of the corresponding type, or *(ii)* flushed to the initial, blank state. To this end, each artifact container is equipped with an auxiliary flag fr_i, which reflects its current state: fr_i is false when the container stores a concrete artifact instance, true otherwise. Then, the internal semantics of *create-artifact-instance* is changed so as to check the availability of a blank artifact container. In particular, when the corresponding service call is to be invoked with the new artifact instance data, the calling artifact instance selects the next available blank artifact container, sets its flag fr_i to $false$, and fills it with the payload of the service call. If all containers are occupied, the calling artifact instance waits until some container is released. Symmetrically to artifact creation, the deletion procedure for an artifact instance is managed by turning the corresponding container flag fr_i to true. Details on the DCDS CA-rules formalizing creation/deletion of artifact instances according to these principles can be found in Appendix A.

We observe that, following this container-based realization strategy, the information model of an instance-bounded GSM model has a fixed size, which polinomially depends on the total maximum number N_{max}. The new implementation of *create-artifact-instance* does not really change the size of the information model, but just suitably changes its content. Therefore, Corollary 1 directly applies to instance-bounded GSM models, guaranteeing decidability of their verification. Finally, notice that infinitely many different artifact instances can be created and manipulated, provided that they do not accumulate in the same state (exceeding N_{max}).

Part III
KAB Instantiation: Artifact Systems with Semantic Layer

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 (this is, e.g., the case of GSM). A rich, conceptual and well-founded modeling of the data component is yet to come. Furthermore, a suitable balance between these two coexisting aspects is often missing, leading to artifacts that 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.

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

By leveraging on the recent, extensive work on ontology-based data access (see e.g., [42, 11, 37]), the primary goal of this part is to overcome such limitations by proposing a framework for the semantic enrichment, governance and management of artifact-centric systems, consequently discussing concrete counterparts for the abstract framework of semantically-governed artifact systems and the notion of ACSI semantic layer (cf. Section 4).

Even though the notion of semantic layer is orthogonal to the artifact/process modeling language of choice, we show how this idea can be concretely exploited by grounding it in the recently proposed GSM (Guard-Stage-Milestone) artifact modeling language ([35, 25]). This is possible not only because GSM is associated to a concrete modeling and execution environment, but also because (domain) ontologies have a formal underpinning in Description Logics (DLs), which in turn come with a plethora of reasoning services and corresponding techniques[8]. These reasoning services do not only cover design-time tasks such as satisfiability and consistency of the ontology, but also query answering, consequently allow for a run-time, live exploitation of ontologies, which goes far beyond their usage for modeling purposes only.

More specifically, the contributions of this part can be summarized as follows:

- We provide a formal definition of the *semantic GSM* model, providing a concrete counterpart for the formal framework of KBDSs (cf. Section 3)[9]. 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 constitute 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 ([38, 27]) and ontology-based data

[8] See, for instance, the services offered by the DL-based reasoners presented in [33, 44, 49, 43].

[9] The correspondence between semantic GSM and KBDSs can be established by exploiting the correspondence between standard GSM and DCDSs (cf. Part II).

access ([42]) 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. This corresponds to a concrete instantiation of semantically-governed artifact systems, and more in general of ACSI semantic layer (cf. Section 4).

- We discuss in details a running example within the ACSI Energy use case, leveraging on what presented in Section 5.3.

We argue that our framework is parametric with respect to the language used for representing the ontology and for querying it, as well as with respect to the (various) forms of mappings that can be adopted to link the ontology with external data management systems. It is out from the scope of this discussion to investigate specific choices for these languages, and the computational problems that consequently arise.

9 DL Ontologies: a Recap

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 ([38, 27]). The discussion extends what already discussed in Section 3.1 for lightweight ontologies that rely on the DL-Lite family, and ECQ queries. We will also extensively use the notion of mappings to link data with ontologies, following what discussed in Section 4.2.

9.1 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. DL expressions are built starting from an alphabet Γ, which 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 different kinds of TBox and/or ABox assertions. Formally, a DL ontology $\mathcal{O}$ over an alphabet Γ is a pair $\langle T, A\rangle$, where T is a TBox and 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. [4]). 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 T alone.

Various reasoning services can be performed over DL ontologies, and are supported by state-of-the-art automated reasoners (see, e.g., [33, 44, 49]). 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 the following, we do not refer to a specific ontology language, being our contributions applicable to any DL ontology. For the sake of simplicity we provide only graphical representation of ontologies (or better of their approximation) given through ER diagrams.

9.2 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 $q(\vec{x})$ be an $\mathcal{L}$-query with free (a.k.a. distinguished) variables $\vec{x}$ over an ontology $\mathcal{O}$. Then, a tuple $\vec{t}$ of constants from Γ_C is a *certain answer* for $q(\vec{x})$ if $\mathcal{O} \models q(\vec{t})$, where $q(\vec{t})$ is the closed formula, i.e., a sentence, obtained by substituting $\vec{x}$ with $\vec{t}$. 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.

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 ([1]), 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., [17, 31]).

In the rest of this part, we consider, as a query language over a DL ontology $\mathcal{O}$, *ECQ*s (cf. Section 3.1) extended with built-in operators:

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

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. [16]). 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.

10 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 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

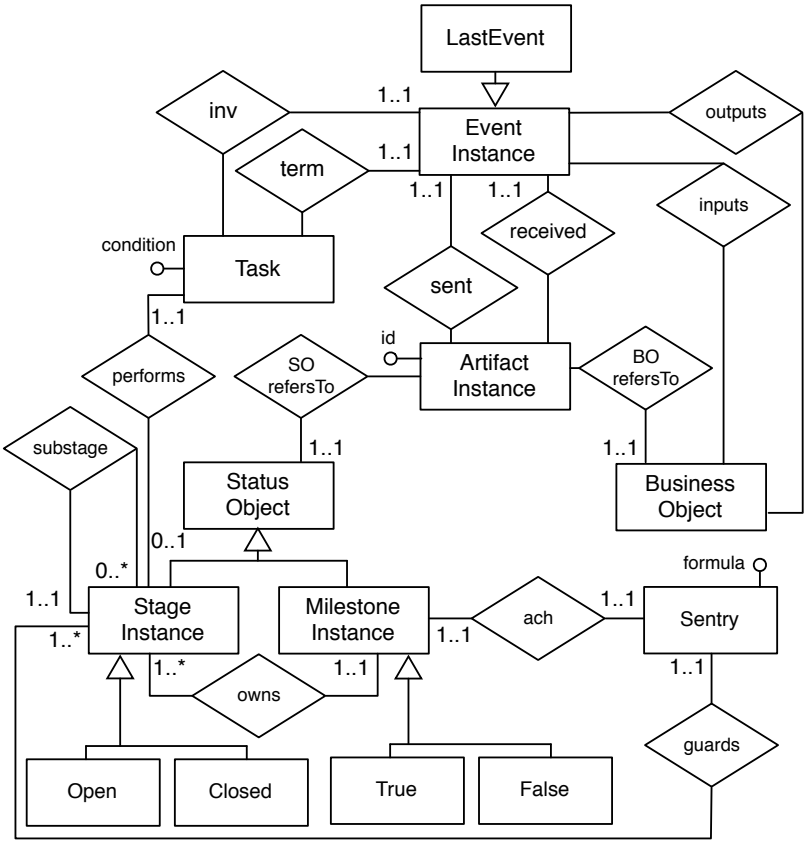


Figure 14 – Graphical representation of the lifecycle TBox for Example 10.1

are waiting for a task termination event from the environment, or which composite stages have an achieved milestone, i.e., 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 in order to represent the schema of a specific process. Figure 14 shows a graphical representation of the upper ontology, in which ArtifactInstance is the central concept connected to the three main (macro-)entities of a GSM model: *(i)* event (EventInstance concept); *(ii)* status attributes, here called StatusObjects and *(iii)* data attributes, called BusinessObjects. An event has inputs and outputs (which are business objects) and can be sent or received by an artifact instance. 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., allowing for only one instance, given that a GSM system processes one event at a time. A status object is either a milestone or a stage and such a 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. 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 a GSM schema. Concepts model the domain by specializing BusinessObject, while the lifecycle schema is defined by specializing StatusObject. Moreover, EventInstance generalizes all event type instances required by the process and the same holds for Task. Also roles can be specialized in order 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 T over Γ containing the upper ontology in Figure 14.*

Then, the notion of snapshot for a semantic GSM schema is given below.

Definition 4. *A* semantic snapshot *for a semantic data schema T is an ABox A such that:*

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

Intuitively, (2) and (3) corresponds to GSM-1 and GSM-2 invariants, respectively, explained in Section 5.

The conditional language used for specifying semantic sentries is a query language $\mathcal{L}$ for T with alphabet Γ. Differently from classic GSM, now $\mathcal{L}$-formulas 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 $-\mathsf{Stg}(s) \equiv \mathsf{Stg}(s) \wedge \mathsf{Close}(s) \wedge \mathsf{Open}'(s)$ 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 temporal formulas that refers to two semantic snapshots (A, A'). The intuitive semantics is similar to that presented in Section 5, 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 $\mathcal{L}$-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.

Example 10.1. We model the energy process described in Example 5.1 with a semantic GSM schema T. Figure 16 shows a graphical representation of the portion of T describing the domain of interest. All concepts in the figure are intended to specialize the BusinessObject concept of the upper ontology in Figure 14 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.

Notice that such a TBox contains other assertion which are not rendered graphically, but which can be captured by the ontology language. As an example, each CPA is identified by the month, the year and the control point it refers to.

In Figure 16, a partial fragment of 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

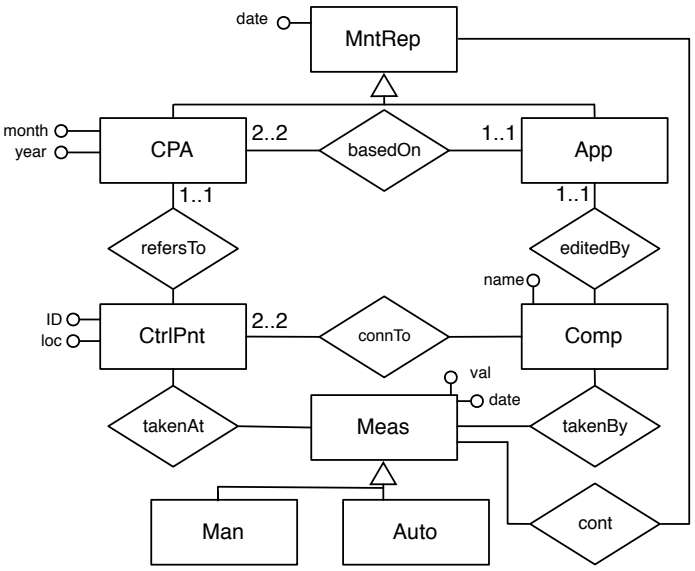


Figure 15 – Fragment of the semantic GSM schema for the energy process in Example 5.1 describing the domain.

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.

Table 5 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 length due to joins, they are easier to manage than the ones in Table 3. 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 Figure 7. 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.[10]

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

Finally, queries can now 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.

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

■

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,

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

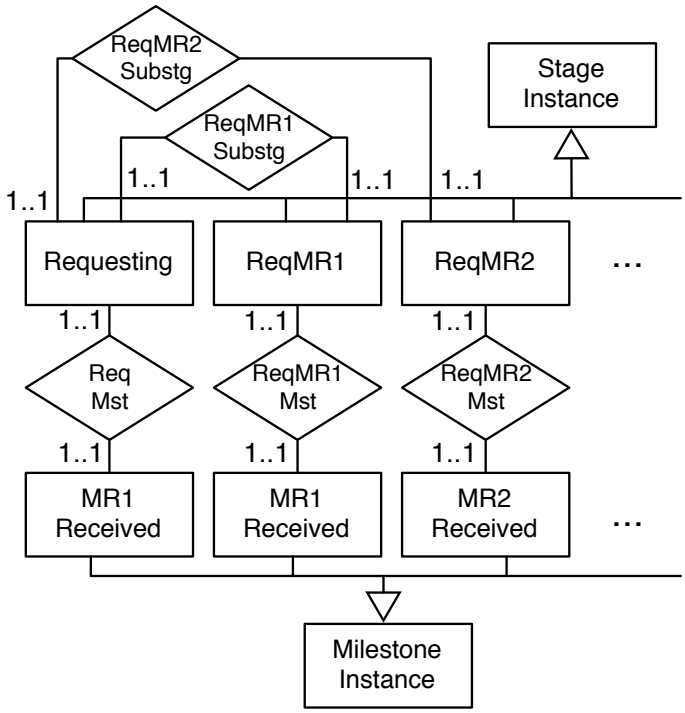
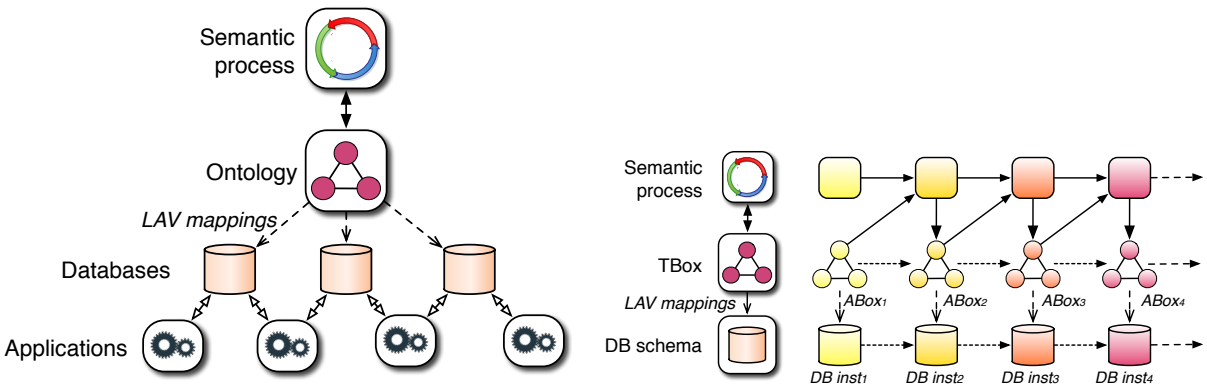


Figure 16 – Fragment of the semantic GSM schema for the energy process in Example 5.1 partially describing the process' lifecycle.

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 logical implication), and on the other hand, due to the unified view of the schema, user queries can now directly refer to both data and lifecycle.

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	$\exists ap1, ap2, v1, v2.App(ap1) \neq App(ap2) \wedge v1 \neq v2 \wedge$ $[\exists a, m1, m2, d.basedOn(a, ap1) \wedge basedOn(a, ap2) \wedge cont(ap1, m1)$ $\wedge cont(ap2, m2) \wedge date(m1, d) \wedge date(m2, d) \wedge val(m1, v1) \wedge val(m2, v2) \wedge$ $((Man(m1) \wedge Man(m2)) \vee (Auto(m1) \wedge Auto(m2)))]$

Table 5 – Guards for semantic Example 10.1.



(a) Architectural view (b) Runtime view

Figure 17 – Semantic GSM with LAV mappings exploited by multiple front-end applications

11 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 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 11.1. 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 17(a) 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 Figure 17(a) 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 Section 10. 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 Section 9 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 $R(\vec{x}) \rightsquigarrow \psi(\vec{x})$, which actually expresses that $R(\vec{x})$ is a (local) view constructed on top of the query $\psi(\vec{x})$ 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. Section 9). 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).

Figure 17(b) 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.

11.1 Data Transfer

In data transfer approach, some data maintained by the ontology are now replicated in the local database, similarly to data exchange (cf. [36]). 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 11.1. 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.

11.2 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 $R(\vec{x}) \rightsquigarrow \psi(\vec{x})$, the relation R is a view corresponding to the certain answers of $\psi(\vec{x})$ 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. [19] 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 [2, 15, 26, 30, 34]).

Example 11.2. 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 the control point assessment to be executed (cf. Figure 7 and Figure 8). 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 Figure 17(a).

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 $R(id, y, m, d, ms) \rightsquigarrow Q_1(id, y, m, d, v)$, where Q_1 is the ontology query in Example 10.1. ■

12 Semantic Monitoring and Governance of Relational Artifacts

We discuss now an architectural solution that complements the one discussed in Section 11, 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 manipulated are typically scattered around into several (typically relational) data sources with different schemas.

Example 12.1. 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.

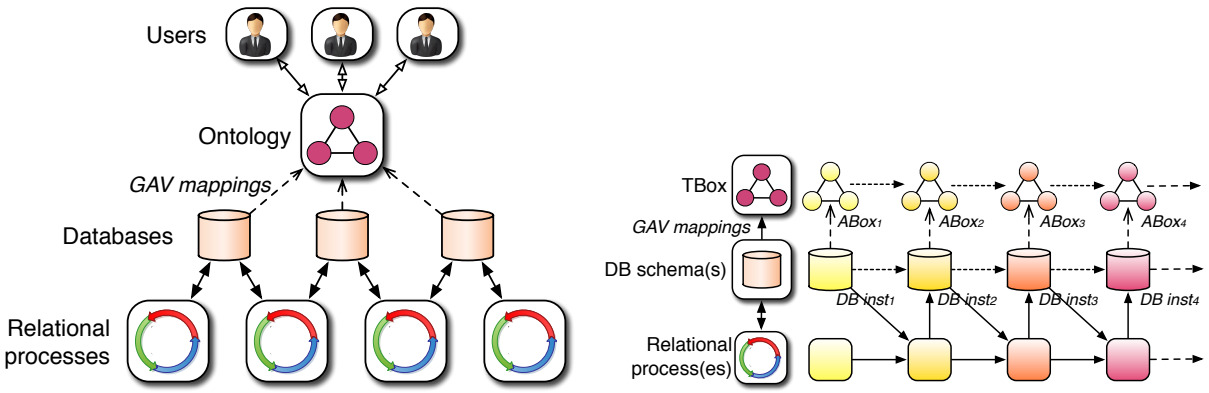
Since in this case the purpose is to understand data in the data sources through the ontology, i.e., (virtually) transfer data from the source schemas to the conceptual schema, the most natural form of mapping to adopt to interconnect the two layers is the one of GAV. In fact, to define a concept N in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

As recalled in Section 9, ontology-based data access (OBDA) techniques have been extensively employed to enable the concrete usage of such 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, thus focusing on a concrete instantiation of semantically-governed artifact systems (cf. Section 4). Figure 18(a) 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 Figure 18(b), 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 new, resulting snapshot can then be queried at the conceptual level through the ontology itself.

Example 12.2. Consider again our running example on the ACSI energy use case. Assume now that the control point assessment artifact does not materialize data it receives from the companies, but it has suitable mappings towards the databases locally used by such companies to store their reports sent monthly to the system operator (i.e., their applications). For example, the company C uses the following two relational tables to store such reports containing claimed measures on a daily basis for a certain control point in a certain month:

$$\mathsf{R1}(\mathsf{MRA\_ID}, \mathsf{CP\_ID}, \mathsf{month}, \mathsf{year})$$
$$\mathsf{R2}(\mathsf{MSR\_ID}, \mathsf{MRA\_ID}, \mathsf{date}, \mathsf{time}, \mathsf{value}, \mathsf{type}).$$

The 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 to 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'`.



(a) Architectural view (b) Runtime view

Figure 18 – Relational processes with GAV mappings and a unifying ontology

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 and date together form 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 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 Figure 15. ■

Analogously to what we did in Section 11, 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.

12.1 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 fruitfully exploited to support external audits. In fact, part of an audit is typically dedicated to analyse (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 [50]. See the discussion about the "semantic event log", provided below.

12.2 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 is 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. [42] and [43] provide notable examples in which the above rewritings can be expressed in SQL, thus allowing for deleting its evaluation to the underlying relational data management systems.

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 Section 10. 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 Figures 14 and 16. These concepts and relationships can be easily attached 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 10.1.
- 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. This scenario instantiates the one of Figure 4 . Obviously, it requires a mechanism to evaluate the action effects before effectively enforcing them, triggering an exceptional behaviour 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, Figure 20 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, as defined in the context of ACSI interoperation hubs to formally regulate to which pieces of information the various stakeholders participating to the hub share an access.

12.3 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 [50].

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 broaden including also a plethora of operational decision support tasks that are exploited at run-time, i.e., by considering live data of running process

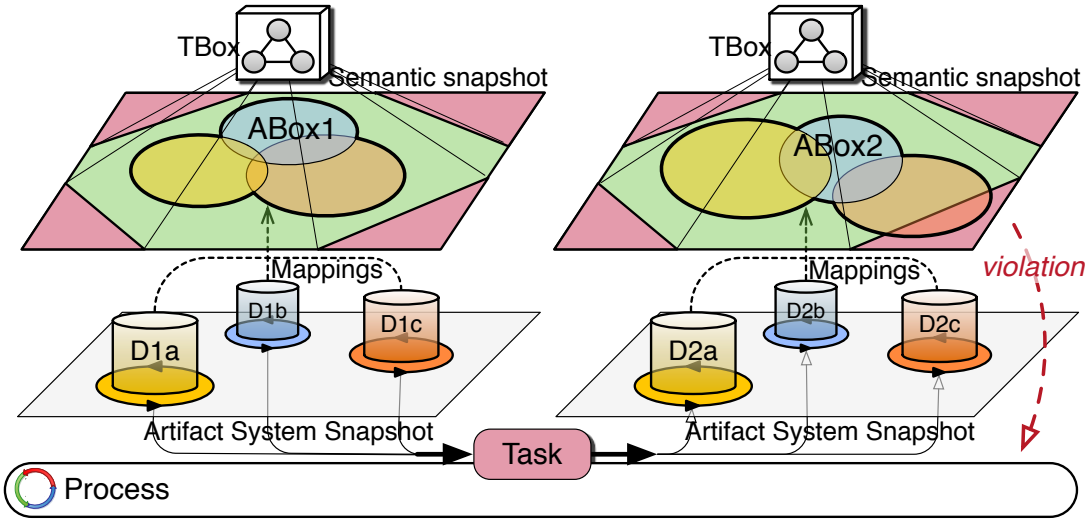


Figure 19 – Ontology-based governance with propagation of violations from the Semantic Layer down to the Artifact Layer

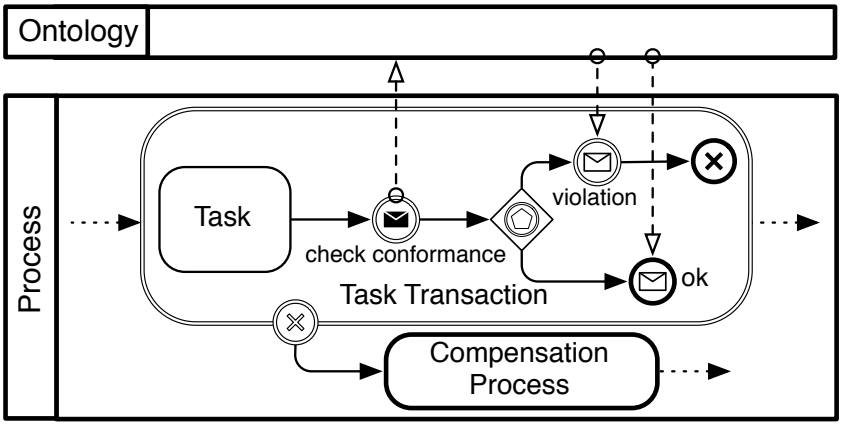


Figure 20 – Meta-model of an ontology-governed BPMN task

instances.

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 to 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., Figure 21).
- 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-portem 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

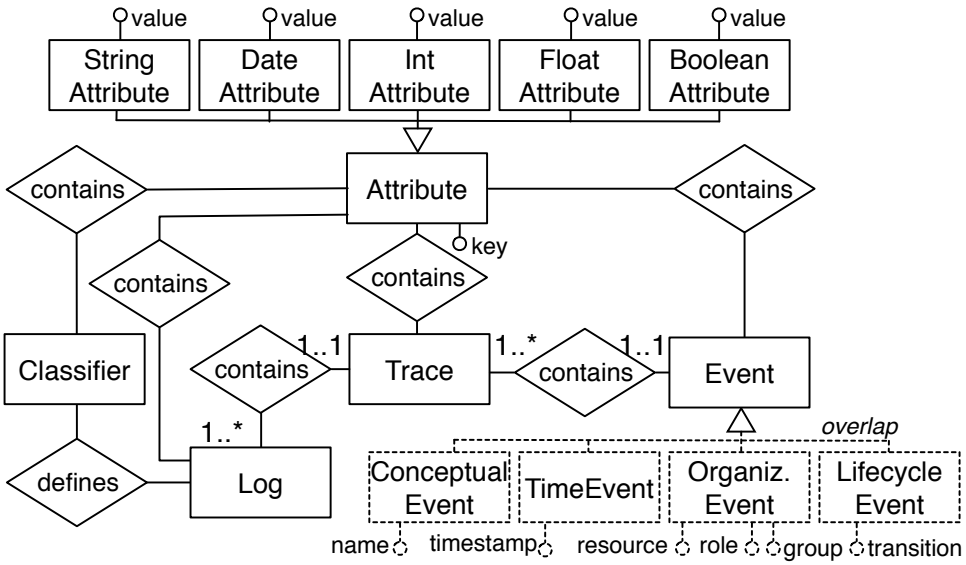


Figure 21 – E-R diagram capturing a portion of the XES meta-model; the dashed part is reported for clarity, but is concretely realized in XES through the notion of extension and corresponding required attributes for the events.

event log, and consequently checking whether a process execution trace is currently respecting or violating certain business rules, in the style of [23, 40].

Part IV
Model Checking GSM with Semantic Layer

In Section 4, we have introduced SASs as a formal framework for representing artifact systems with a Semantic Layer. In particular, we have focused our attention to the usage of lightweight Description Logics, belonging to the DL-Lite family, to conceptually capture the relevant domain entities and relationships at the Semantic Layer. At the same time, we have shown that, thanks to the FO rewritability of DL-Lite, verification of temporal properties and dynamic laws over the evolution of an artifact system understood through the lens of the Semantic Layer can be faithfully reduced to verification of properties directly carried out at the Artifact Layer.

With a different perspective but relying on the same approach, in Section 12 we have discussed possible concrete usages of this framework.

We now continue the investigation of artifact systems equipped with a Semantic Layer, showing how the techniques developed for KBDSs can be exploited to effectively attack the verification problem, leveraging on:

- *Ontop*[11], a JAVA-based framework for OBDA, and in particular the Quest reasoner, which is the component dedicate to handle query rewriting and unfolding;
- the *GSMC model checker*, developed within ACSI to verify GSM-based artifact-centric systems against temporal/dynamic properties [32].

In particular, we report on the development of *OBGSM*, a JAVA-based tool that, given a temporal property specified over the ontology that captures the Semantic Layer of the system under study, together with mapping assertions whose language is suitably shaped to work with GSMC, automatically rewrites and unfolds the property by producing a corresponding property that can be directly fed into GSMC. This rewriting and unfolding procedure cannot be done by solely relying on the functionalities provided by *Ontop*, for two reasons:

- OBGSM deals with temporal properties specified in a fragment of $\mu\mathcal{L}_P^{\text{EQL}}$, and not just (local) ECQs;
- the mapping assertions are shaped so as to reflect the specific query language supported by GSMC, guaranteeing that the rewriting and unfolding process produces a temporal property expressed in the input language of GSMC.

13 OBGSM System Specification

The OBGSM tool takes a *conceptual temporal property* specified over the Semantic Layer of a GSM-based artifact-system, producing a corresponding temporal property that can be directly verified by GSMC [32] over the GSM specification, without involving the Semantic Layer anymore. More specifically, OBGSM has three inputs:

1. a conceptual temporal property Φ (provided in a `*.prop` file);
2. an OWL 2 QL [12] TBox (provided in a `*.owl` file);
3. a mapping specification $\mathcal{M}$ (provided in a `*.obgsm` file).

We detail in the following the languages used to specify Φ and $\mathcal{M}$.

[11] `http://ontop.inf.unibz.it`

[12] `http://www.w3.org/TR/2008/WD-owl2-profiles-20081008/#OWL_2_QL`. OWL 2 QL is the OWL2 profile that closely corresponds to the DL-Lite family of Description Logics.

13.1 Specification of Conceptual Temporal Properties

The input conceptual temporal property is specified in a file with the extension `*.prop`. As far as the temporal component is concerned, the verification language relies on CTL, in accordance to the input verification language of GSMC [32][13]. Remember that CTL is subsumed by μ-calculus. As far as the local queries over the ontology are concerned, the language relies on SPARQL, in accordance to the query language supported by Ontop. More specifically, the syntax is as follows:

```
formula ::= [ query ]
          | (formula)
          | formula AND formula
          | formula OR formula
          | formula -> formula
          | ! formula
          | AG formula
          | EG formula
          | AF formula
          | EF formula
          | AX formula
          | EX formula
          | A (formula UNTIL formula)
          | E (formula UNTIL formula)
          | FORALL Var . forallQuantification
          | EXISTS Var . existsQuantification

forallQuantification ::= [ query ] -> formula
                       | FORALL Var . forallQuantification
                       | [ query ]

existsQuantification ::= [ query ] AND formula
                       | EXISTS Var . existsQuantification
                       | [ query ]
```

where [query] is a SPARQL 1.1[14] SELECT query. SELECT queries, in turn, obey to the following grammar:

```
query ::= PrefixDeclarations SELECT Var WHERE {Triples FILTER(filter)}

filter ::= filterExpression
         | filterExpression && filter
```

[13] GSMC also supports epistemic operators, which are not considered here.

[14] `http://www.w3.org/TR/sparql11-query/`

```
filterExpression ::= var_const < var_const
                   | var_const <= var_const
                   | var_const > var_const
                   | var_const >= var_const
                   | var_const = var_const
                   | var_const ! = var_const

var_const ::= Var
            | integer
            | "string"
            | "string"^^http://www.w3.org/2001/XMLSchema#string
            | "string"^^http://www.w3.org/2001/XMLSchema#integer
            | "string"^^http://www.w3.org/2001/XMLSchema#decimal
            | "string"^^http://www.w3.org/2001/XMLSchema#double
            | "string"^^http://www.w3.org/2001/XMLSchema#dateTime
            | "string"^^http://www.w3.org/2001/XMLSchema#boolean
            | "string"^^http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal
```

where:

- **Var** is a variable that obeys to the pattern `?([a-z]|[A-Z])+`;
- **Triples** and **PrefixDeclarations** follow the usual triple patterns and prefix declarations of SPARQL 1.1;
- `integer` and `string` are the standard integer and string built-in domains.

Additionally, we require that all variables present in the SELECT clause of the query also appear in the WHERE clause, and vice-versa; in other words, all variables in the query must be answer variables.

The semantics of the temporal operators is the one of CTL [9]. As far as first-order quantification is concerned, we impose the following restrictions:

- Only closed temporal formulae are supported for verification.
- Each first-order quantifier must be "guarded", in such a way that it ranges over individuals present in the current active domain. This active domain quantification is in line with GSMC, and also with the $\mu\mathcal{L}_A^{\text{EQL}}$ and $\mu\mathcal{L}_P^{\text{EQL}}$ logics introduced before in this document. As attested by the grammar above, this is syntactically guaranteed by requiring quantified variables to appear in a [`query`] according to the following guidelines:

$$\forall \vec{x}.\texttt{query}(\vec{x}) \rightarrow \phi$$
$$\exists \vec{x}.\texttt{query}(\vec{x}) \wedge \phi$$

- Quantified variables must obey to specific restrictions, depending on whether they quantify over object terms or values. This can be syntactically recognized by checking whether the variable appears in the second component of an attribute (in this case, it ranges over values) or not. The restriction is as follows: for each variable y ranging over values, there must be at least one variable x that ranges over object terms and that appears in the first component of the corresponding attribute (i.e., $Attr(x, y)$ is present in the query, with $Attr$ being an attribute of the TBox), such that x is quantified "before" y. For example, $\forall x.C(x) \implies \exists y.Attr(x.y)$ satisfies this condition, whereas $\exists y \exists x.Attr(x, y)$ does not.

These restrictions have been introduced so as to guarantee that the conceptual temporal property can be translated into a corresponding GSMC temporal property. In fact, GSMC poses several restrictions on the way values (i.e., attributes of artifacts) can be accessed.

Example 13.1. We consider a simple university information system similar to the one in [18]. The following TBox is used to capture the relevant concepts and relations of the university domain at the Semantic Layer:

$$\begin{array}{rcl@{\qquad}rcl} Bachelor & \sqsubseteq & Student & \delta(MNum) & \sqsubseteq & Student \\ Master & \sqsubseteq & Student & \delta(HasAge) & \sqsubseteq & Student \\ Graduated & \sqsubseteq & Student & \exists Attend & \sqsubseteq & Student \\ & & & \exists Attend^{-} & \sqsubseteq & Course \end{array}$$

The Artifact Layer contains the following artifact types:

1. `ENROLLEDSTUDENT`, whose instances represent the enrolled students. For each enrolled student, these data attributes are maintained: `ID, MNum, Name, Age, Type`, where `ID, Name` and `Type` are of type String, while `Age` and `MNum` are of type Integer.
2. `GRAD`, whose instances represent those students who have been graduated. The following data attributes are maintained: `ID, MNum`.
3. `COURSE`, whose instances represent the courses offered by the university. They have the following data attributes: `ID, CourseName`.

An example of interesting temporal property specified over the Semantic Layer is:

```
EF FORALL ?x. ([PREFIX : <http://acsi/example/student/student.owl#>
                PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                SELECT ?x WHERE {  ?x rdf:type :Bachelor }] ->
               [PREFIX : <http://acsi/example/student/student.owl#>
                SELECT ?x WHERE {  ?x rdf:type :Graduated }]
          );
```

which says that "eventually there is a state in the future where all bachelor students are graduated". Another example is:

```
EF FORALL ?x. ([PREFIX : <http://acsi/example/student/student.owl#>
                PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
                SELECT ?x WHERE {?x rdf:type :Master; :HasAge "26"^^xsd:integer}]
               -> [PREFIX : <http://acsi/example/student/student.owl#>
                   SELECT ?x WHERE {  ?x rdf:type :Graduated }]
          );
```

It states that "eventually in the future there is a state where all bachelor students who are at least 26 years old are graduated". ■

Notice that in the second temporal property of Example 13.1, the typed value `"26"^^xsd:integer` is used to denote the age of students. More in general, according to the current implementation of Ontop, there is support for the following type of values:

- xsd:string
- xsd:integer
- xsd:decimal
- xsd:double
- xsd:dateTime
- xsd:boolean

- rdf:Literal

where "`xsd:`" and "`rdf:`" are predefined prefixes, respectively defined as"`xsd: http://www.w3.org/2001/XMLSchema#`" and "`rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#`". Whenever an input value is not typed, we consider it to be, by default, of type `rdf:Literal`.

13.2 Specification of the Input Mapping

The specification of input mapping assertions is provided inside a file with extension `*.obgsm`. The structure of our mapping language is borrowed from the one of Ontop. More specifically, the expected file format is:

```
[PrefixDeclaration]
...

[ClassDeclaration] @collection [[
...
]]

[ObjectPropertyDeclaration] @collection [[
...
]]

[DataPropertyDeclaration] @collection [[
...
]]

[MappingDeclaration] @collection [[
...
]]
```

In the following, we detail the different parts of this format.

Prefix, Class, Object Property and Data Property Declaration

The `[PrefixDeclaration]` part contains the definition of the URI (Uniform Resource Identifier) prefixes that will be used in the remainder of the file.

Example 13.2. We provide a simple prefix declaration that could be contained in a mapping specification file:

```
[PrefixDeclaration]
xsd: http://www.w3.org/2001/XMLSchema#
rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
: http://acsi/example/student/student.owl#
```

■

Parts `[ClassDeclaration]`, `[ObjectPropertyDeclaration]`, and `[DataPropertyDeclaration]`, respectively contain the declaration of *concepts*, *roles*, and *attributes* name that will be mentioned in the mapping declarations. They are also specified in terms of URIs, where each entry is separated by comma.

Example 13.3. We provide three sample declarations for a class, an object property, and a data property, respectively:

```
[ClassDeclaration] @collection [[
:Student, :Bachelor, :Graduated, :Master, :Course
]]

[ObjectPropertyDeclaration] @collection [[
:Attend
]]

[DataPropertyDeclaration] @collection [[
:MNum, :HasAge
]]
```

■

Mapping Declaration

The `[MappingDeclaration]` contains the declaration of mapping assertions (cf. Section 4.2). When constructing object terms starting from artifact identifiers in the Artifact Layer, we require that only unary function symbols are used. As in Ontop, such unary function symbols are in turn represented by *URI templates* (i.e., a preset format for URIs). For example, the object term $stud(x)$ is represented as `<http://www.acsi-project.eu/example/#stud{x}>`.

Each mapping assertion is then described by three components:

1. `mappingId`, which provides a unique identifier for the mapping assertion.
2. `target`, which contains the *target query* (i.e., the head of the mapping). Technically, a target query is a CQ over the vocabulary of the ontology. For the specification of such target query, we adopt the Ontop syntax, which is in turn based on the Turtle[15] syntax to represent RDF triples. Each atom in the CQ is in fact represented as an RDF-like triple template. There are three kinds of possible atoms in the target query:
   (a) Concepts, expressed as

   `[URI_Template] rdf:type [ConceptName]`

   where `[ConceptName]` is an URI, and `rdf:` is the prefix `rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#`. For example, to represent the atom

   `ConceptName(c(x))`

   (where *ConceptName* is a concept name in the ontology), the following notation is used:

   `<"&:;c{$x}"> rdf:type :ConceptName`

   where ":" is a predefined prefix.
   (b) Roles, again expressed as triples:

   `[URI_Template] [RoleName] [URI_Template]`

   where `[RoleName]` is an URI. For example, the atom

   `RoleName(r1(x), r2(y))`

   (where *RoleName* is a role name in the ontology) is represented as:

   `<"&:;r1{$x}"> :RoleName <"&:;r2{$y}">`

[15] `http://www.w3.org/TR/turtle/`

where ":" is a predefined prefix.

(c) Attributes, whose definition resembles the one of roles:

`[URI_Template] [AttributeName] [TypedOrUntypedVariable]`

where `[AttributeName]` is an URI. For example,

`AttributeName(att(x), integer(y))`

is represented as:

`<"&:;att{$x}"> :AttributeName $y^^xsd:integer .`

where ":" and "`xsd:`" are predefined prefixes, and "`xsd:`" is defined as "`xsd: http://www.w3.org/2001/XMLSchema#`". It is worth noting that the second component of an attribute is a value. We assume that it originates from a value attribute contained inside the information model of an artifact, we use dedicated function symbols to wrap the value into an object term, ensuring that this choice does not overlap with any function symbol chosen for "real" object terms. In the example above, we use "`http://www.w3.org/2001/XMLSchema#integer`", but in general, Ontop supports all the following special data types:

- `http://www.w3.org/2001/XMLSchema#string`
- `http://www.w3.org/2001/XMLSchema#integer`
- `http://www.w3.org/2001/XMLSchema#decimal`
- `http://www.w3.org/2001/XMLSchema#double`
- `http://www.w3.org/2001/XMLSchema#dateTime`
- `http://www.w3.org/2001/XMLSchema#boolean`
- `http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal`

3. `source`, which describes the *source query*, i.e., the body of the mapping. The grammar of the source query is borrowed from the grammar of the GSMC input language[32], with extensions that allow to "extract" artifact identifiers and their value attributes, so as to link them to the ontology. The extended syntax is:

$$
\begin{aligned}
\texttt{expression} ::= \ & \texttt{constant} \\
| \ & \texttt{expression} == ?variable \\
| \ & \texttt{expression}\ \textbf{aop}\ \texttt{expression} \\
| \ & \texttt{expression}\ \textbf{lop}\ \texttt{expression} \\
| \ & \{variable./path/attributeID\} \\
| \ & \textbf{GSM.isStageActive}('variable','stageID') \\
| \ & \textbf{GSM.isMilestoneAchieved}('variable','milestoneID') \\
| \ & variable.attributeID1 \text{ -> } \textbf{exists}(attributeID2 = \texttt{expression})
\end{aligned}
$$

$$
\begin{aligned}
\texttt{formula} ::= \ & \texttt{expression} \\
| \ & (\texttt{formula}) \\
| \ & \texttt{formula}\ \text{AND}\ \texttt{formula} \\
| \ & \texttt{formula}\ \text{OR}\ \texttt{formula} \\
| \ & !\ \texttt{formula} \\
| \ & \textbf{exists}('variable',\ 'artifactID')(\texttt{formula}) \\
| \ & \textbf{forall}('variable',\ 'artifactID')(\texttt{formula}) \\
| \ & \textbf{get}('variable',\ 'artifactID')(\texttt{formula})
\end{aligned}
$$

The two key additional features rely in the possibility of introducing a variable assigning it to an expression (see the second line in the grammar definition), and the possibility of "getting" a variable representing an instance of the specified artifact (see the last line in the grammar definition). These variables are considered to be free in the specified query, and can be consequently used to "transport" values and artifact identifiers into the Semantic Layer, respectively as attributes and object terms.

Example 13.4. Consider again the Artifact and Semantic Layer introduced in Example 13.1. We specify the following mapping assertions to link the three artifacts and their information models to the ontology present in the Semantic Layer:

```
[MappingDeclaration] @collection [[

mappingId   BachelorStudent
target      <"&:;stud/{$x}/"> rdf:type :Bachelor .
source      get('x', 'ENROLLEDSTUDENT')({x./ENROLLEDSTUDENT/Type} == "Bachelor")

mappingId   MasterStudent
target      <"&:;stud/{$x}/"> rdf:type :Master .
source      get('x', 'ENROLLEDSTUDENT')({x./ENROLLEDSTUDENT/Type} == "Master")

mappingId   MatriculationNumber
target      <"&:;stud/{$x}/"> :MNum $y^^xsd:integer .
source      get('x','ENROLLEDSTUDENT')({x./ENROLLEDSTUDENT/MNum} == ?y)

mappingId   GraduatedStudent
target      <"&:;stud/{$x}/"> rdf:type :Graduated .
source      get('x', 'ENROLLEDSTUDENT')(exists('y', 'GRAD')(
           {x./ENROLLEDSTUDENT/MNum} == {y./GRAD/MNum}))

mappingId   Age
target      <"&:;stud/{$x}/"> :HasAge $y^^xsd:integer .
source      get('x', 'ENROLLEDSTUDENT')(x./ENROLLEDSTUDENT/Age == ?y)

mappingId   AttendingCourse
target      <"&:;stud/{$x}/"> :Attend <"&:;course/{$y}/"> .
source      get('x', 'ENROLLEDSTUDENT')(get('y','COURSE')
           (x./ENROLLEDSTUDENT/AttendedCourses->exists(ID == {y./COURSE/ID})))

]]
```

The first two mapping assertions are used to populate bachelor and master students in the Semantic Layer, by extracting information from artifact instances of type `ENROLLEDSTUDENT`, respectively selecting those instances whose `Type` field corresponds to the string "Bachelor" or "Master". Notice that the artifact instance identifier x is used to create the corresponding student object term $stud(x)$ in the ontology.

The following three mapping assertions are used to populate attributes in the Semantic Layer, starting from specific artifacts and fields in their information models. According to the previously discussed restrictions, the first component of attributes is always associated to an object term constructed starting from an artifact instance identifier, and the second from a value in its information model.

The last mapping assertions is used to populate a relation in the Semantic Layer, starting from pairs of artifact identifiers in the Artifact Layer. In particular, the source query is used to extract all pairs of artifact instances of type `ENROLLEDSTUDENT` and `COURSE`, such that the course artifact instance is among the attended courses by the student artifact instance (notice the navigation `x./ENROLLEDSTUDENT/AttendedCourses` to select all attended courses, and the consequent join used to check whether the considered course instance is among the attended ones). In this case, both the first and the second component of the association are object terms constructed from artifact instance identifiers. ■

13.3 OBGSM Workflow and Components

As depicted in the Fig. 22, the workflow of OBGSM is as follows.

1. The tool reads and parses the input conceptual temporal property Φ, the input ontology T, and the input mapping declaration $\mathcal{M}$.
2. The tool *rewrites* the input conceptual temporal property Φ based on the input ontology (TBox) T, in order to compile away the TBox. This step produces *rewritten temporal property* $rew(\Phi)$.
3. The rewritten property $rew(\Phi)$ is *unfolded* by exploiting $\mathcal{M}$. The final temporal property $\Phi_{GSM} = unfold(rew(\Phi))$ obeys to the syntax expected by GSMC, and is such that verifying Φ over the transition system of the GSM model under study after projecting its states into the Semantic Layer through $\mathcal{M}$, is equivalent to verifying Φ_{GSM} directly over the GSM model (without considering the Semantic Layer).
4. GSMC is invoked by passing Φ_{GSM} together with the specification file of the GSM model under study.

Notice that the correctness of the translation is guaranteed by the fact OBGSM manipulates the local components of the query Φ according to the standard rewriting and unfolding algorithms, while maintaining untouched the temporal structure of the property. This has been proven to be the correct way of manipulating the property (cf. Section 4.6). The proof has been done for $\mu\mathcal{L}_P^{\text{EQL}}$ and $\mu\mathcal{L}_A^{\text{EQL}}$, and since first-order CTL with active domain quantification is a fragment of $\mu\mathcal{L}_A^{\text{EQL}}$, the result directly applies also in our setting. This result also shows that OBGSM can largely rely on state-of-the-art existing rewriting and unfolding techniques to manipulate the temporal properties. Indeed, OBGSM exploits Ontop to accomplish this task, adding a last step to deal with the constructs that have been introduced for the mapping assertions, but that are not directly supported by GSMC. In particular, "get" statements are turned into corresponding existential quantifications, whereas variable assignments are managed by directly applying the constraints involving these variables to the attributes of the artifact information models, finally removing such additional variables.

OBGSM consists of the following main components:

1. *Temporal Property Parser and Validator*. This component parses and validates a conceptual temporal property, checking its well-formedness and whether it guarantees the required restrictions or not. The parser for the temporal part of the property is implemented using Antlr 4.0 [16]. For parsing the local queries in the temporal property, the SPARQL 1.1 parser component from Ontop is extensively used, together with the Apache Jena™ library[17].
2. *Ontology Parser*. This component, entirely provided by Ontop, reads and parse an OWL 2 QL ontology.
3. *Mapping Parser*. This component reads and parses the file containing mapping assertions. As for the implementation, the parser already present in Ontop is reused and complemented with the additional features by using Antlr 4.0.
4. *Temporal Property Rewriter*. When the input temporal property and the input ontology have been parsed, OBGSM *rewrites* the parsed temporal property based on the given ontology, using this component. The implementation of the query rewriting functionality is fully inherited from Ontop.
5. *Unfolder*. This component takes the specification of mapping assertions and the property produced by the rewriter, producing the final unfolded property. To do so, it extends the base unfolding functionality already present in Ontop.

[16] `http://www.antlr.org`
[17] `http://jena.apache.org`

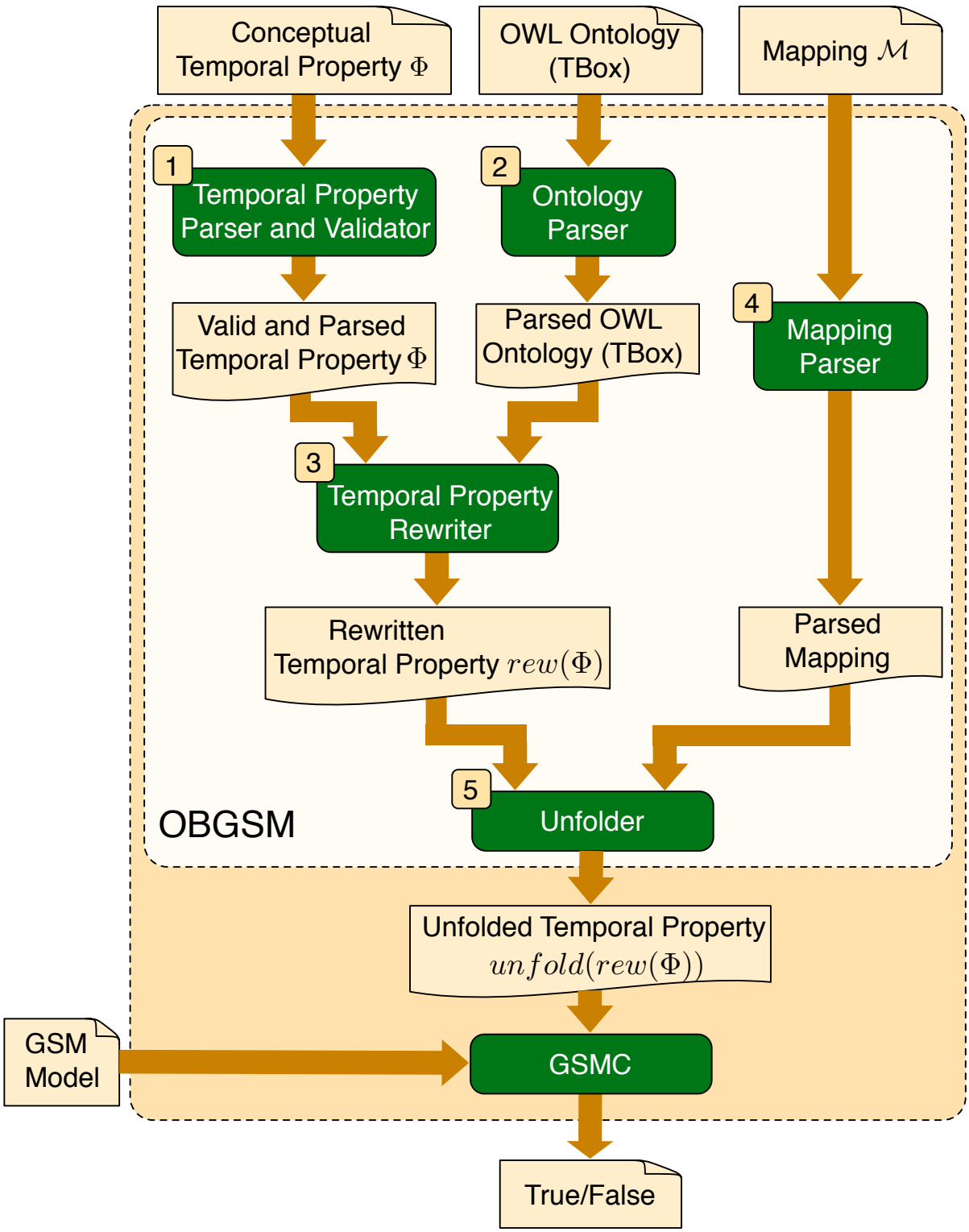


Figure 22 – OBGSM System Architecture

13.4 Running the OBGSM

OBGSM is distributed as a JAR application, which can be directly run from the command line. It can be downloaded from: `http://www.inf.unibz.it/~asantoso/obgsm/`. The three command line options are:

1. `-o <ontology file path (*.owl file)>`
2. `-p <temporal property file path (*.prop file)>` property file path.
3. `-m <mapping file path (*.obgsm file)>`

A sample invocation of the tool follows:

```
java -jar OBGSM.jar -o ex/ontology.owl -p ex/tempProperty.prop -m ex/mappings.obgsm
```

14 An Example from the ACSI Energy Use Case

This section describes the application of OBGSM to a fragment of the ACSI Energy use case. We show how the Semantic Layer can be exploited to facilitate the specification of temporal properties of interest, and discuss how they are automatically translated into properties that can be directly verified by GSMC over the Energy GSM model.

14.1 ACSI Energy Use Case at a Glance

We sketch here the main aspects of the Energy use case, referring the interested reader to [47, 46, 48] for a detailed treatment.

The ACSI Energy use case focuses on the e electricity supply exchange between electric companies inside the distribution network in Spain. The electricity exchange between companies occurs at *control points*. Within a control point, a measurement of electricity supply exchange takes place in order to calculate the fair remuneration that the participating companies in the control point should receive. The measurement is done by a *meter reader company*, which corresponds to one of the companies pertaining to that certain control point. The measurement result is then submitted to the *system operator*, which is in charge of publishing it. If the company has any objection concerning the published measurement, the company can raise an objection. Once the objection is resolved, the report is closed.

The GSM Model for ACSI Energy Use Case

To implement the sketched scenario, we consider two artifacts:

- *Control Point Monthly Report* (*CPMR*). This artifact contains the information about hourly measurements done in a control point within a certain month. The lifecycle of an instance of this artifact is started when the MDM system provides the hourly measurements, and runs until the liquidation for the CP measurements is started. This artifact consists of three root stages:
  - *CPMRInitialization*, activated when a new instance of the CPMR artifact is created.
  - *Claiming*. This stage handles the submission of measurements, and the consequent reviewing stage, where objections may be raised. Five sub-stages are used to deal with this lifecycle in a fine-grained way: *Drafting*, *Evaluating*, *Reviewing*, *Closing*, and *CreateObjection*.
  - *MeasurementUpdating*, activated when there is an event requesting for updating the measurement results.

- *CPMR Monthly Accumulator* (*CPMRMA*), responsible of the measurement files. It submits measurements to the system operator, and receives back from the operator the value for the corresponding official measurements. It consists of four root stages:
  - *CPMRMAInitialization*, which handles the generation of measurement files.
  - *SubmittingMeasurementFile*, which submits the measurement files to the system operator.
  - *EvaluatingMeasurementFile*, which waits the official measurements from the system operator.
  - *ProcessingOfficialMeasurement*, which calculates the differences between the official measurements and the submitted measurements, and consequently notify the CPMR instances about the differences. The *CalculatingDifferences* and *NotifyingOfficialMeasurement* sub-stages are used to handle this portion of the lifecycle.

Figure 23 shows the GSM model for the two artifacts described above.

14.2 The Semantic Layer

We provide a Semantic Layer on top of the GSM model for the ACSI Energy use case, restricting our attention to the Published Control Point Measurement Report (CPMR).

The Ontology

An UML model for the ontology of the Semantic Layer is depicted in Figure 23. A control point measurement report can be either:

- a finished CPMR (when the milestone *CPMRFinished* is achieved),
- a reviewed CPMR (after finishing the review inside the *Reviewing* stage)
- an accepted CPMR (when the milestone *PublishedOK* is achieved)
- an objected CPMR.

We formalize the UML model in *DL-Lite*$_{\mathcal{A}}$:

$$\begin{aligned} FinishedReport &\sqsubseteq ControlPointReport \\ ReviewedReport &\sqsubseteq ControlPointReport \\ AcceptedReport &\sqsubseteq ControlPointReport \\ ObjectedReport &\sqsubseteq ControlPointReport \\ \exists contains &\sqsubseteq ControlPointReportCollection \\ \exists contains^{-} &\sqsubseteq ControlPointReport \\ \delta(controlPointID) &\sqsubseteq ControlPointReport \\ \rho(controlPointID) &\sqsubseteq String \end{aligned}$$

This ontology is shown visually in Figure 24.

The Mapping Assertions

We use the following mapping assertions in order to link the information model in the GSM model to the Semantic Layer. The assertions are written in the mapping specification language of OBGSM.

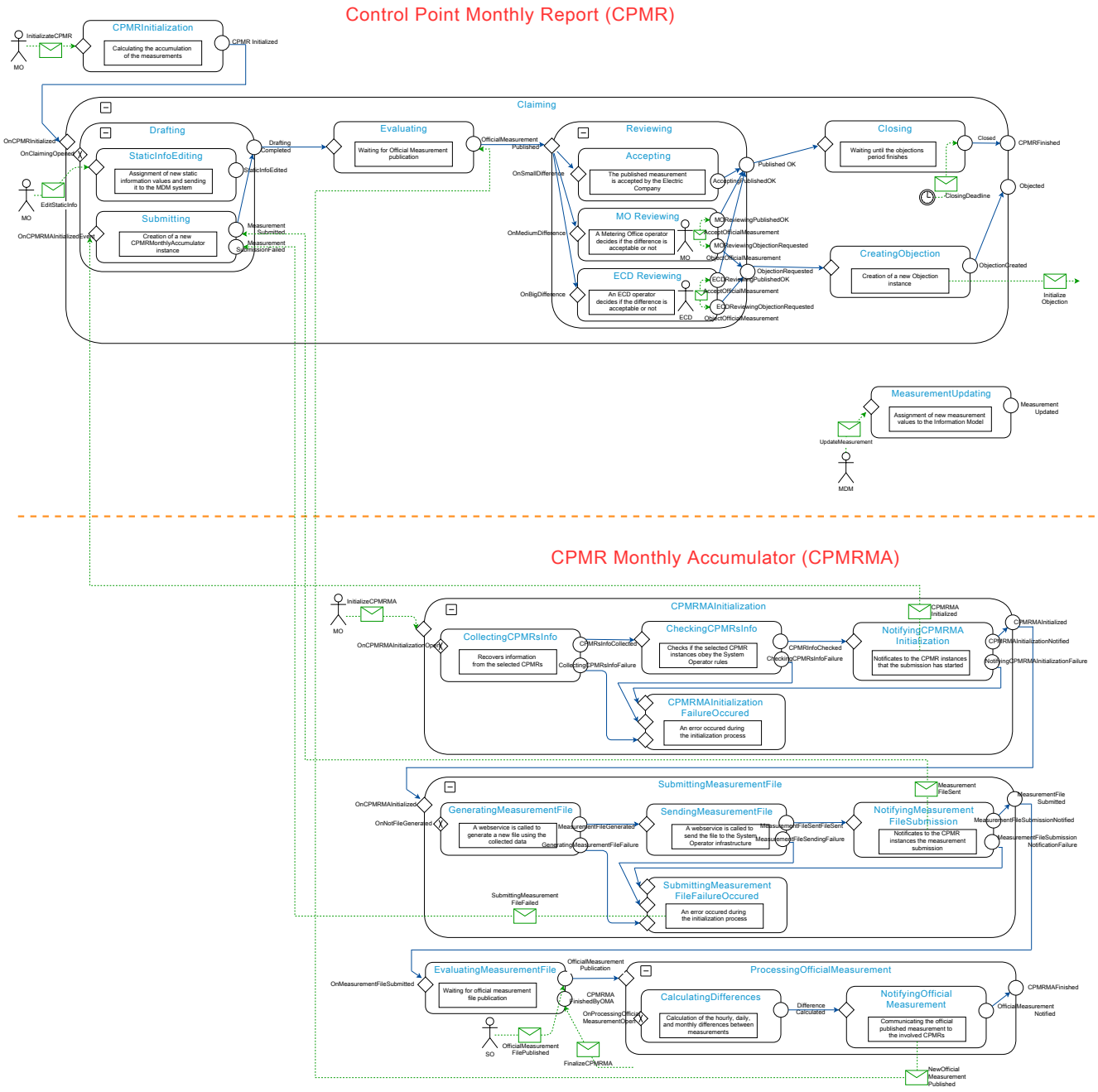


Figure 23 – GSM Model for ACSI Energy Use Case

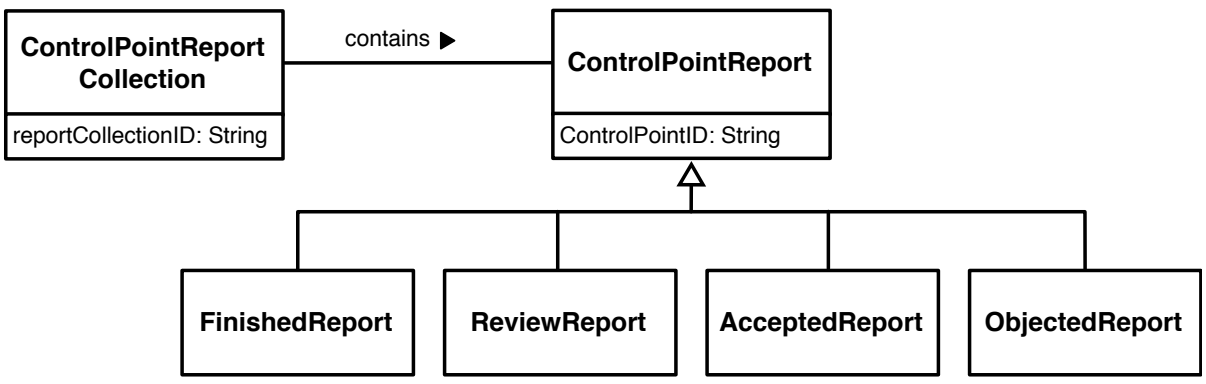


Figure 24 – Ontology for the CPMR reviewing process in ACSI Energy Use Case

```
[PrefixDeclaration]
  xsd: http://www.w3.org/2001/XMLSchema#
  owl: http://www.w3.org/2002/07/owl#
  : http://acsi/example/ACSIEnergy/ACSIEnergy.owl#
  rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#

[ClassDeclaration] @collection [[
  owl:Thing, :ControlPointReportCollection, :ControlPointReport,
  :ObjectedReport, :AcceptedReport, :ReviewedReport, :FinishedReport
]]

[ObjectPropertyDeclaration] @collection [[
  :contains
]]

[DataPropertyDeclaration] @collection [[
  :hasControlPointID
]]


  mappingId  ControlPointReportCollectionMapping
  target     <"&:;cpmrma/{$x}/"> rdf:type :ControlPointReportCollection .
  source     get('x', 'CPMRMA')(TRUE)

  mappingId  ControlPointIDMapping
  target     <"&:;cpmr/{$x}/"> :hasControlPointID $y^^xsd:string .
  source     get('x', 'CPMR')({x./CPMR/CPID} == ?y)

  mappingId  CPMRMAContainsCPRMMapping
  target     <"&:;cpmrma/{$x}/"> :contains <"&:;cpmr/{$y}/"> .
  source     get('x', 'CPMRMA')(get('y', 'CPMR')(
                        x./CPMRA/CPMRDATA->exists(CPMRID == {y./CPMR/ID})))


  mappingId  ReviewedReportMapping
  target     <"&:;cpmr/{$x}/"> rdf:type :ReviewedReport .
  source     get('x','CPMR')(GSM.isMilestoneAchieved('x','AcceptingPublishedOK') OR
                        GSM.isMilestoneAchieved('x','MOReviewingPublishedOK') OR
                        GSM.isMilestoneAchieved('x','ECDReviewingPublishedOK'))

  mappingId  AcceptedReportMapping
  target     <"&:;cpmr/{$x}/"> rdf:type :AcceptedReport .
  source     get('x','CPMR')(GSM.isMilestoneAchieved('x','PublishedOK'))

  mappingId  ObjectedReportMapping
  target     <"&:;cpmr/{$x}/"> rdf:type :ObjectedReport .
  source     get('x','CPMR')(GSM.isMilestoneAchieved('x','Objected') OR
                        GSM.isMilestoneAchieved('x','ObjectionRequested') OR
                        GSM.isMilestoneAchieved('x','ObjectionCreated') OR
                        GSM.isMilestoneAchieved('x','MOReviewingObjectionRequested') OR
                        GSM.isMilestoneAchieved('x','ECDReviewingObjectionRequested'))

  mappingId  FinishedReportMapping
  target     <"&:;cpmr/{$x}/"> rdf:type :FinishedReport .
  source     get('x','CPMR')(GSM.isMilestoneAchieved('x','CPMRFinished'))
]]
```

Where ":" is a prefix declared as `http://acsi/example/energy/energy.owl#`. The intuition of some mapping assertions above is as follows:

- `ControlPointReportMapping` and `ControlPointReportCollectionMapping` populate the concepts *ControlPointReport* and *ControlPointReportCollection* with the CPMR and CPMRMA artifact instances respectively.
- `CPMRMAContainsCPMRMapping` populates the *contains* role, which relates CPMRMA with the CPMRs it contains.
- `ControlPointIDMapping` populates the attribute *hasControlPointID* by relating *ControlPointReport* with its control point ID.
- `ReviewedReportMapping` populates the *ReviewedReport* concept with the CPMR artifact instance, given the achievement of one of the three milestones *AcceptingPublishedOK MOReviewingPublishedOK* or *ECDReviewingPublishedOK*. In the Semantic Layer, *ReviewedReport* intuitively represents a CPMR that has been reviewed. In the Artifact Layer, this corresponds to the situation in which the CPMR has been reviewed and accepted either by the electric company, or by the metering office (MO), or by the Electric Control Department (ECD). This example show how such details can be hidden from the Semantic Layer, which does not show the fact that a *ReviewedReport* is obtained by a (possibly complex) chaining of achieved milestones in the underlying GSM model.
- `AcceptedReportMapping` populates the *AcceptedReport* concept with the CMPR artifact instance, when milestone *PublishedOK* is achieved. This intuitively means that the *AcceptedReport* is a published CPMR that has been approved.
- `ObjectedReportMapping` populates the *ObjectedReport* concept with an objected CMPR artifact instance. This situation is recognized, at the Artifact Layer, by combining different related milestones.
- `FinishedReportMapping` populates the *FinishedReport* concept with the finished CMPRs, i.e., those that have achieved milestone *CPMRFinished*.

14.3 Verification

In this section, we demonstrate how the presence of the Semantic Layer can help in the specification of temporal properties of interests. We consider in particular the following properties:

1. All control point reports will eventually will be finished.
2. All control point reports that are accepted must have been reviewed. This property is used to ensure that there is no way to achieve a state in which a certain CPMR is accepted, without going through the review for that CPMR. Notice that "having being reviewed" is considered to be a permanent property of CPMRs.
3. All control point reports that are finished must not be objected control point reports. This property ensures that control point reports cannot be classified as finished as long as they are still objected.
4. All objected control point reports must not be finished control point reports.

Such four properties can be expressed as conceptual temporal properties over the Semantic Layer. In particular, we encode them using the language provided by OBGSM:

1. AG(FORALL x . ([$ControlPointReport(x)$] -> EF[$FinishedReport(x)$]))
2. AG(FORALL x . ([$AcceptedReport(x)$] -> [$ReviewedReport(x)$]))
3. AG(FORALL x . ([$FinishedReport(x)$] -> ![$ObjectedReport(x)$]))
4. AG(FORALL x . ([$ObjectedReport(x)$] -> ![$FinishedReport(x)$]))

We now show how this high-level property are compiled by OBGSM into underlying temporal properties that can be fed into the GSMC model checker. We stress that, without the presence of the Semantic Layer, the user would be forced to write this low-level properties manually.

1. The rewriting step for the first conceptual temporal property produces the following formula, which "embeds" the constraints of the ontology present at the Semantic Layer:

$$\begin{aligned}\text{AG}(\text{FORALL } x \text{ . } ((\text{EXISTS } y \text{ . } [hasControlPointID(x,y)]) \text{ OR}\\ [ObjectedReport(x)] \text{ OR}\\ [AcceptedReport(x)] \text{ OR}\\ [ControlPointReport(x)] \text{ OR}\\ [ReviewedReport(x)] \text{ OR}\\ (\text{EXISTS } z \text{ . } [contains(z,x)]) \text{ OR}\\ [FinishedReport(x)] \text{ OR}\\ \text{-> EF}[FinishedReport(x)]))\end{aligned}$$

   The expansion of the queries contained in the property is done by embedding the following cases, reflected by the ontology constraints:

   - Those objects that have a control point ID (i.e., are in the domain of the attribute $hasControlPointID$), are instances of $ControlPointReport$.
   - $ObjectedReport$ is a $ControlPointReport$.
   - $AcceptedReport$ is a $ControlPointReport$.
   - $ReviewedReport$ is a $ControlPointReport$.
   - Those objects that are in the range of the role $contains$ are instances of $ControlPointReport$.
   - $FinishedReport$ is a $ControlPointReport$.

   By exploiting the mapping assertions, OBGSM unfolds the rewritten property into this final result:

$$\begin{aligned}\text{AG } (\textbf{forall}('x',\ 'CPMR')(!(&GSM.isMilestoneAchieved('x','Objected') \text{ OR}\\ &GSM.isMilestoneAchieved('x','ObjectionRequested') \text{ OR}\\ &GSM.isMilestoneAchieved('x','ObjectionCreated') \text{ OR}\\ &GSM.isMilestoneAchieved('x','MOReviewingObjectionRequested') \text{ OR}\\ &GSM.isMilestoneAchieved('x','ECDReviewingObjectionRequested') \text{ OR}\\ &GSM.isMilestoneAchieved('x','PublishedOK') \text{ OR}\\ &(false) \text{ OR}\\ &GSM.isMilestoneAchieved('x','AcceptingPublishedOK') \text{ OR}\\ &GSM.isMilestoneAchieved('x','MOReviewingPublishedOK') \text{ OR}\\ &GSM.isMilestoneAchieved('x','ECDReviewingPublishedOK') \text{ OR}\\ &\textbf{exists}('y','CPMRMA')(y./CPMRA/CPMRDATA \text{ ->}\\ &\qquad \textbf{exists}(CPMRID == x./CPMR/ID)) \text{ OR}\\ &GSM.isMilestoneAchieved('x','CPMRFinished')) \text{ OR}\\ &EF(GSM.isMilestoneAchieved('x','CPMRFinished'))))\end{aligned}$$

   Notice that the absence of a mapping assertion for the concept $ControlPointReport$ results into a $false$ disjunct in the unfolding.

2. The translation of the second conceptual temporal property produces this final result:

$$\begin{aligned}&\text{AG } \textbf{forall}('x',\ 'CPMR')(!(GSM.isMilestoneAchieved('x','PublishedOK')) \text{ OR } (\\ &\quad GSM.isMilestoneAchieved('x','AcceptingPublishedOK') \text{ OR}\\ &\quad GSM.isMilestoneAchieved('x','MOReviewingPublishedOK') \text{ OR}\\ &\quad GSM.isMilestoneAchieved('x','ECDReviewingPublishedOK'))\\ &\quad)\end{aligned}$$

3. The translation of the third conceptual temporal property produces this final result:

$$\begin{aligned}&\text{AG } \textbf{forall}('x',\ 'CPMR')(!GSM.isMilestoneAchieved('x',CPMRFinished') \text{ OR}\\ &\quad !(GSM.isMilestoneAchieved('x','Objected') \text{ OR}\\ &\quad GSM.isMilestoneAchieved('x','ObjectionRequested') \text{ OR}\\ &\quad GSM.isMilestoneAchieved('x','ObjectionCreated') \text{ OR}\\ &\quad GSM.isMilestoneAchieved('x','MOReviewingObjectionRequested') \text{ OR}\\ &\quad GSM.isMilestoneAchieved('x','ECDReviewingObjectionRequested'))\\ &\quad)\end{aligned}$$

4. The translation of the fourth conceptual temporal property produces this final result:

$$\begin{aligned}&\text{AG } \textbf{forall}('x',\ 'CPMR')(!(GSM.isMilestoneAchieved('x','Objected') \text{ OR}\\ &\quad GSM.isMilestoneAchieved('x','ObjectionRequested') \text{ OR}\\ &\quad GSM.isMilestoneAchieved('x','ObjectionCreated') \text{ OR}\\ &\quad GSM.isMilestoneAchieved('x','MOReviewingObjectionRequested') \text{ OR}\\ &\quad GSM.isMilestoneAchieved('x','ECDReviewingObjectionRequested')) \text{ OR}\\ &\quad !GSM.isMilestoneAchieved('x',CPMRFinished')\\ &\quad)\end{aligned}$$

By comparing the properties specified over the Semantic Layer and their corresponding translations, it is apparent that, even in this simple case study, the presence of the Semantic Layer hides low-level details, helps the modeler in focusing on the domain under study, and allows for using the vocabulary he/she is familiar with.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[2] P. Adjiman, P. Chatalic, F. Goasdoué, M.-C. Rousset, and L. Simon. Distributed reasoning in a peer-to-peer setting: Application to the Semantic Web. *Journal of Artificial Intelligence Research*, 25:269–314, 2006.

[3] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The *DL-Lite* family and relations. *Journal of Artificial Intelligence Research*, 36:1–69, 2009.

[4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2nd edition, 2007.

[5] B. Bagheri Hariri, D. Calvanese, G. De Giacomo, R. De Masellis, P. Felli, and M. Montali. Verification of description logic knowledge and action bases. In L. De Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. J. F. Lucas, editors, *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 103–108. IOS Press, 2012.

[6] B. Bagheri Hariri, D. Calvanese, G. De Giacomo, R. De Masellis, P. Felli, and M. Montali. Description logic knowledge and action bases. *Journal of Artificial Intelligence Research*, 46:651–686, 2013.

[7] B. Bagheri Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of relational data-centric dynamic systems with external services. CoRR Technical Report arXiv:1203.0024, arXiv.org e-Print archive, 2012.

[8] B. Bagheri Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of relational data-centric dynamic systems with external services. In *Proc. of the 32nd ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2013)*. ACM Press and Addison Wesley, 2013.

[9] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[10] J. Bradfield and C. Stirling. Modal mu-calculi. In *Handbook of Modal Logic*, volume 3, pages 721–756. Elsevier, 2007.

[11] A. Calì, G. Gottlob, and T. Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. In *Proc. of the 28th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2009)*, pages 77–86, 2009.

[12] D. Calvanese, G. De Giacomo, B. Bagheri Hariri, R. De Masellis, D. Lembo, and M. Montali. Techniques and tools for KAB, to manage action linkage with the Artifact Layer – Iteration 1. Deliverable ACSI-D2.4.1, ACSI Consortium, May 2012.

[13] D. Calvanese, G. De Giacomo, D. L., M. Lenzerini, A. Poggi, M. Rodríguez-Muro, and R. Rosati. Ontologies and databases: The *DL-Lite* approach. In S. Tessaris and E. Franconi, editors, *Reasoning Web. Semantic Technologies for Informations Systems – 5th Int. Summer School Tutorial Lectures (RW 2009)*, volume 5689 of *Lecture Notes in Computer Science*, pages 255–356. Springer, 2009.

[14] D. Calvanese, G. De Giacomo, D. Lembo, R. De Masellis, P. Felli, D. F. Savo, M. Montali, and F. Patrizi. The complete ACSI Artifact Paradigm. Deliverable ACSI-D1.3, ACSI Consortium, Jan. 2012.

[15] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. What to ask to a peer: Ontology-based query reformulation. In *Proc. of the 9th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 469–478, 2004.

[16] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. EQL-Lite: Effective first-order query processing in description logics. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 274–279, 2007.

[17] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[18] D. Calvanese, G. De Giacomo, D. Lembo, M. Montali, M. Ruzzi, and A. Santoso. Techniques and tools for KAB, to manage data linkage with the Artifact Layer. Deliverable ACSI-D2.3, ACSI Consortium, May 2012.

[19] D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Logical foundations of peer-to-peer data integration. In *Proc. of the 23rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2004)*, pages 241–251, 2004.

[20] D. Calvanese, G. De Giacomo, M. Montali, and F. Patrizi. Verification and synthesis in description logic based dynamic systems. In *Proc. of the 7th Int. Conf. on Web Reasoning and Rule Systems (RR 2013)*. Springer, 2013.

[21] D. Calvanese, E. Kharlamov, M. Montali, A. Santoso, and D. Zheleznyakov. Verification of inconsistency-aware knowledge and action bases. In *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013)*. AAAI Press/The MIT Press, 2013.

[22] D. Calvanese and M. Montali. Evolvability of the interoperation hub – Iteration 2. Deliverable ACSI-D2.5.2, ACSI Consortium, May 2013.

[23] F. Chesani, P. Mello, M. Montali, F. Riguzzi, M. Sebastianis, and S. Storari. Checking compliance of execution traces to business rules. In D. Ardagna, M. Mecella, and J. Yang, editors, *Proc. of the BPM 2008 Workshops, 4th Int. Workshop on Business Intelligence (BPI 2008)*, volume 17 of *Lecture Notes in Business Information Processing*, pages 134–145. Springer, 2009.

[24] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. The MIT Press, 1999.

[25] E. Damaggio, R. Hull, and R. Vaculín. On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. *Information Systems*, 38(4):561–584, 2013.

[26] G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. On reconciling data exchange, data integration, and peer data management. In *Proc. of the 26th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2007)*, pages 133–142, 2007.

[27] A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[28] E. A. Emerson. Model checking and the mu-calculus. In *Descriptive Complexity and Finite Models*, 1996.

[29] E. Franconi, G. Kuper, A. Lopatenko, and L. Serafini. A robust logical and computational characterisation of peer-to-peer database systems. In *Proc. of the VLDB International Workshop On Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2003)*, 2003.

[30] A. Fuxman, P. G. Kolaitis, R. J. Miller, and W. C. Tan. Peer data exchange. *ACM Trans. on Database Systems*, 31(4):1454–1498, 2005.

[31] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. *Journal of Artificial Intelligence Research*, 31:151–198, 2008.

[32] P. Gonzales, A. Griesmayer, and A. Lomuscio. Model checking tool for artifact interoperations (MOCAI) – Iteration 3. Deliverable ACSI-D2.2.3, ACSI Consortium, May 2013.

[33] V. Haarslev and R. Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.

[34] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *Proc. of the 19th IEEE Int. Conf. on Data Engineering (ICDE 2003)*, pages 505–516, 2003.

[35] R. Hull, E. Damaggio, R. De Masellis, F. Fournier, M. Gupta, F. T. Heath, III, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. N. Sukaviriya, and R. Vaculin. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *Proc. of DEBS*. ACM, 2011.

[36] P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proc. of the 24th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2005)*, pages 61–75, 2005.

[37] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to query answering in *DL-Lite*. In *Proc. of the 12th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2010)*, pages 247–257, 2010.

[38] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 21th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)*, pages 233–246, 2002.

[39] M. Linehan. GSM expression language. Technical report, IBM Research, Jenuary 2011. Available on request.

[40] M. Montali, F. M. Maggi, F. Chesani, P. Mello, and W. M. P. van der Aalst. Monitoring business constraints with the event calculus. *ACM Transactions on Intelligent Systems and Technology*, 2013.

[41] D. M. R. Park. Finiteness is Mu-ineffable. *Theoretical Computer Science*, 3(2):173–181, 1976.

[42] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.

[43] M. Rodriguez-Muro and D. Calvanese. High performance query answering over *DL-Lite* ontologies. In *Proc. of the 13th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2012)*, pages 308–318, 2012.

[44] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *J. of Web Semantics*, 5(2):51–53, 2007.

[45] C. Stirling. *Modal and Temporal Properties of Processes*. Springer, 2001.

[46] D. Toribio Gomez, C. Murphy O Connor, P. De Leenheer, P. Malarme, J. De Vos, S. Christiaens, F. Fournier, and L. Limonad. Energy and FRIS "as-is" assessment. Deliverable ACSI-D5.2, ACSI Consortium, May 2011.

[47] D. Toribio Gomez, C. Murphy O Connor, P. De Leenheer, P. Malarme, J. De Vos, F. Fournier, D. Boaz, B. van Dongen, D. Fahland, M. de Leoni, and M. Dumas. Energy and FRIS use case definition and requirements. Deliverable ACSI-D5.1, ACSI Consortium, Mar. 2010.

[48] D. Toribio Gómez, C. Murphy-O Connor, P. D. Leenheer, and P. Malarme. Deployment and evaluation of pilots using the ACSI Hub System – Iteration 1. Deliverable ACSI-D5.3.1, ACSI Consortium, June 2012.

[49] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, pages 292–297, 2006.

[50] W. M. P. van der Aalst, A. Adriansyah, A. K. A. de Medeiros, F. Arcieri, T. Baier, T. Blickle, R. P. J. C. Bose, P. van den Brand, R. Brandtjen, J. C. A. M. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. de Leoni, P. Delias, B. F. van Dongen, M. Dumas, S. Dustdar, D. Fahland, D. R. Ferreira, W. Gaaloul, F. van Geffen, S. Goel, C. W. Günther, A. Guzzo, P. Harmon, A. H. M. ter Hofstede, J. Hoogland, J. E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. La Rosa, F. M. Maggi, D. Malerba, R. S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H. R. Motahari Nezhad, M. zur Muehlen, J. Muñoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H. S. Pérez, R. S. Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K. D. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen,

G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard, and M. T. Wynn. Process mining manifesto. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Proc. of the BPM 2011 Workshops, 7th Int. Workshop on Business Intelligence (BPI 2011)*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer, 2011.

Part V
Appendix

A Translating GSM into the DCDS Framework

Here we detail the steps to translate a GSM model R into a corresponding DCDS $\mathcal{S}_R = \langle \mathcal{D}, \mathcal{P} \rangle$, which faithfully reproduces the dynamics of the original GSM model. We attack this problem by first discussing how the information model of R is translated into the data layer $\mathcal{D}$ of $\mathcal{S}_R$, and then focusing on the translation of the lifecycle of R into the process layer $\mathcal{P}$ of $\mathcal{S}_R$.

A.1 Data layer

Given an artifact type $(R, x, Att_{data} \cup Att_{status}, Typ, Stg, Mst, Lcyc)$,
a lifecycle $Lcyc = (Substages, Task, Owns, Guards, Ach, Inv)$[18],
a set of finite sets **MSG** of message types and **SRV** of (2-way) services,
a corresponding set of PAC-rules Γ_{PAC};
the corresponding data layer $\mathcal{D} = \langle \mathcal{C}, \mathcal{R}, \mathcal{E}, \mathcal{I}_I \rangle$ will have the following form:

- $\mathcal{C} = \bigcup_i \mathbf{DOM}(Typ(Att_i))$,
- $\mathcal{R} = \{R_{att}\} \cup \{R^{m_i}_{chg} \mid m_i \in Mst\} \cup \{R^{s_j}_{chg} \mid s_j \in Stg\} \cup$
  $\{R^{msg_k}_{data} \mid msg_k \in \mathbf{MSG}$ and msg_k is incoming 1-way message$\} \cup$
  $\{R^{srv_p}_{data} \mid srv_p \in \mathbf{SRV}$ and srv_p is a service call return$\} \cup$
  $\{R^{msg_q}_{out} \mid msg_q \in \mathbf{MSG}$ and msg_q is outgoing 1-way message$\} \cup$
  $\{R_{exec}, R_{block}\}$,
  where:
  - $R_{att} = (id_A, fr, a_1, ..., a_n, s_1, ..., s_m, m_1, ..., m_k)$, where id_A ranges along the artifact IDs, $n = |Att_{data}|, m = |Stg|, k = |Mst|$. Stores the attributes of an artifact. Each s_i and m_i store the status of a stage and milestone, respectively.
  - $R^{m_i}_{chg} = (id^{origin}_R, newstate)$ – stores the fact of a milestone m_i has been recently achieved or invalidated; id^{origin}_R is the id of the artifact where it happened and $newstate$ stores the new value. This relation is used to model the pool of internal events concerning milestones.
  - $R^{s_j}_{chg} = (id^{origin}_R, newstate)$ – same as previous but about stages being opened or closed.
  - $R^{msg_k}_{data} = (id^{dest}_R, p_1, ..., p_l)$, where msg_k is a 1-way incoming message from the environment, $(p_1, ..., p_l)$ – its signature and id^{dest}_R is the id of a destination artifact (or $null$ if message is indirected)[19].

    Used to model the immediate effect of an incoming message. The idea behind it is that, together with propagating changes of involved attributes, message is passed to the inner 'data-pool' so that all the sentries which use this message in the event expression, could react properly.
  - $R^{srv_p}_{data} = (id^{caller}_R), p_1, ..., p_l$, where srv_p is an external service call return, $(p_1, ..., p_l)$ – its signature and id^{dest}_R is the id of a caller artifact (basicaly the id of a destination artifact for service call return, but it can't be $null$). This is the same as for the incoming message.

[18] Without loss of generality, for the sake of simplicity we consider $Substages = Inv = \emptyset$.
[19] Q: Broadcast messages? A: They use queries to address specific artifacts.

  - $R_{out}^{msg_q} = (id_R^{dest}, a_1, ..., a_l)$ – stores eventual outgoing messages to be sent to the environment after finishing the B-step.
  - $R_{exec} = (id_R, x_1, ..., x_c)$, where x_i are flags that keep information on which PAC rules have been taken in consideration and $c = |\Gamma_{PAC}|$.
  - $R_{block} = (id_R, blocked)$, keeps information whether an artifact instance may receive an incoming message / service call return, or is currently still processing the previous one.
  - $SHELF = (index, id_R)$, implements a proposed methodology of keeping the ACS data-bounded by restricting the number of artifact instances within one snapshot (the number of instances along the execution path may still be infinite).

    Represents a physical storage for artifact instances, where one shelf may contain only one artifact instance. When there is no artifact instance on the shelf - the ID of stored instance is -1.

- $\mathcal{E} = \{\mathcal{E}_i \mid \mathcal{E}_i \text{ is some integrity constraint}\}$
- $\mathcal{I}_0 = \emptyset$.

A.2 Process layer

Given an artifact type $(R, x, Att_{data} \cup Att_{status}, Typ, Stg, Mst, Lcyc)$,
a lifecycle $Lcyc = (Substages, Task, Owns, Guards, Ach, Inv)$,
a set of finite sets **MSG** of message types and **SRV** of (2-way) services,
a corresponding set of PAC-rules Γ_{PAC};
the corresponding process layer $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \varrho \rangle$ will have the following form:

- $\mathcal{F} = \{f^{genID}\} \cup \{f^{srv_i} \mid srv_i \in \mathbf{SRV}\} \cup$
  $\{f^{msg_i} \mid msg_i \in \mathbf{MSG} \text{ and } msg_i \text{ is 1-way message from environment}\} \cup$
  $\{f_{out}^{msg_i} \mid msg_i \in \mathbf{MSG} \text{ and } msg_i \text{ is 1-way outgoing message}\}$,
  where
  - f^{genID} is a function with generates IDs for newly created artifact instances.
  - $srv_i(\overline{x}) = (f^{srv_i}(\overline{x}, 1), ..., f^{srv_i}(\overline{x}, n))$, where n is cardinality of service output signature.
  - $msg_i = (f^{msg_i}(1), ..., f^{msg_i}(n))$, where n is cardinality of message signature.
- $\mathcal{A} = \{\alpha_i\}$ is a set of actions, where $i = \overrightarrow{1, ..., N_A}$ and
  $N_A = |\Gamma_{PAC}| + |MSG_{in}| + |SRV| + 1 + 1 + 1$, i.e. there is
  - one action for every PAC-rule of the given GSM model
  - one action for each incoming message (to describe immediate effect);
  - one action for each service call (for immediate effect of the call return);
  - one action to send outgoing messages after each B-step;
  - one action to create an artifact instance;
  - one action to remove the artifact.
- A process ϱ, which is a set of condition-action rules is described below, where for each action $\alpha_i \in \mathcal{A}$ there is one and only condition-action rule defined.

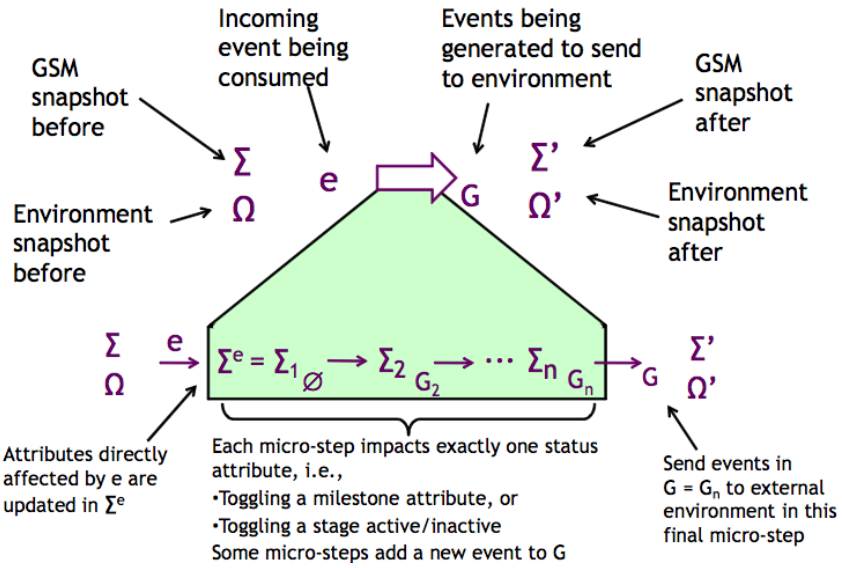


Figure 25 – Incremental formulation of a B-step [25]

B-step in DCDS

When modeling a GSM model as a DCDS system, what we would like to do is to mimic an incremental semantics of GSM, i.e. we encode each micro-step of the B-step as a separate condition-action rule in DCDS system, such that the effect on the data and process layer of the ACS of this action coincides with the effect of corresponding micro-step in GSM.

Recall the structure of a B-step in GSM represented on Figure 25. According to the incremental formulation of GSM, each B-step consists of an initial micro-step which incorporates incoming event into current snapshot, a sequence of micro-steps executing all applicable PAC-rules, and finally a micro-step sending a set of generated events at the termination of the B-step. The translation relies on the incremental semantics: given a GSM model $\mathcal{G}$, we encode each possible micro-step as a separate condition-action rule in the process of a corresponding DCDS system $\mathcal{S}$, such that the effect on the data and process layers of the action coincides with the effect of the corresponding micro-step in GSM. However, in order to guarantee that the transition system induced by a resulting DCDS mimics the one of the GSM model, the translation procedure should also ensure that all semantic assumption of GSM are modeled properly: (i) "one-message-at-a-time" and "toggle-once" principles, (ii) the finiteness of micro-steps within a B-step, and (iii) their order imposed by the model. We sustain these requirements by introducing into the data layer a set of auxiliary relations, suitably recalling them in the CA-rules to reconstruct the desired behaviour.

Thus, when performing the translation we rely on the following assumptions:

1. Restricting $\mathcal{S}$ to process only one incoming message at a time is implemented by the introduction of a *blocking mechanism*, represented by an auxiliary relation $R_{block}(id_R, blocked)$ for each artifact in the system, where id_R is the artifact instance identifier, and *blocked* is a boolean flag. This flag is set to *true* upon receiving an incoming message, and is then reset to $false$ at the termination of the corresponding B-step, once the outgoing events accumulated in the B-step are sent the environment. If an artifact instance has $blocked = true$, no further incoming event will be processed. This is enforced by checking the flag in the condition of each CA-rule associated to the artifact.

2. In order to ensure "toggle once" principle and guarantee the finiteness of sequence of micro-steps triggered by an incoming event, we introduce an *eligibility tracking mechanism*. This mechanism is represented by an auxiliary relation $R_{exec}(id_R, x_1, ..., x_c)$, where c is the total number of PAC-rules, and each x_i corresponds to a certain PAC-rule of the GSM model. Each x_i encodes whether the corresponding PAC rule is eligible to fire at a given

moment in time (i.e., a particular micro-step). The initial setup of the eligibility tracking flags is performed at the beginning of a B-step, based on the evaluation of the prerequisite condition of each PAC rule. More specifically, when $x_i = 0$, the corresponding CA-rule is eligible to apply and has not yet been considered for application. When instead $x_i = 1$, then either the rule has been fired, or its prerequisite turned out to be false.

3. The same flag-based approach is used to propagate in a compact way information related to the PAC rules that have been already processed, following a mechanism that resembles *dead path elimination* in BPEL. In fact, R_{exec} is also used to enforce a firing order of CA-rules that follows the one induced by $\mathcal{G}$. This is achieved as follows. For each CA-rule $Q \mapsto \alpha$ corresponding to a given PAC rule r, condition Q is put in conjunction with a further formula, used to check whether all the PAC rules that precede r according to the ordering imposed by $\mathcal{G}$ have been already processed. Only in this case r can be considered for application, consequently applying its effect α to the current artifact snapshot. More specifically, the corresponding CA-rule becomes $Q \wedge exec(r) \mapsto \alpha$, where $exec(r) = \bigwedge_i x_i$ such that i ranges over the indexes of those rules that precede r. Once all x_i flags are switched to 1, the B-step is about to finish: a dedicated CA-rule is enabled to send the outgoing events to the environment, and the artifact instance *blocked* flag is released.

The general translation algorithm is as follows:

1. For each incoming message m_i, construct a CA-rule that:
   - implements immediate effects of an incoming message;
   - puts a block on the artifact instance to perform the B-step.
   - sets up the eligibility flags based on the current snapshot, i.e., for each PAC-rule checks the prerequisite part. If $\pi = false$, then set the boolean flag of the corresponding micro-step 1 (which will basically mean that we have already considered this rule).
2. For each PAC-rule r_i, construct a CA-rule that:
   - contains a check whether the action has been already executed or has been marked as irrelevant (simply checks the boolean flag);
   - contains a check whether all the PAC rules that precede r_i according to the ordering imposed by $\mathcal{G}$ have been already processed;
   - if relevant and eligible and if the antecedent part is true (the query to populate the effect is not empty), performs the required change of the status attribute
   - if relevant and eligible, marks the corresponding boolean flag as *true*.
3. Construct a CA-rule that will send outgoing messages and unblock the artifact instance:
   - check whether all the PAC-rules have been taken into consideration ($\forall x_k \in R_{block} : x_k = 1$) and whether the artifact instance is still blocked;
   - in the action part – release the block;
   - in the action part – flush the eligibility flags.
4. If *create artifact* or *remove artifact* tasks are present, add micro-steps dealing with it (a particular type of the immediate effect micro-steps described later).

Now let us get down to translating each of the possible micro-steps.

Translation 1 (Immediate effect of 1-way incoming message).
Assume an incoming message type M, its associated artifact type R and its signature $(a_1 : Typ(a_1), \ldots, a_k : Typ(a_k))$, where $a_i \in Att_{data}$.
Assume also a set of PAC-rules $\{(\pi_i, \alpha_i, \gamma_i)\}$.

Then an immediate effect of a message of type M on some artifact instance A of type R may be modeled by the following condition-action rule:

$$\exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{block}(id_R, false) \mapsto$$
$$\alpha_M^{ImmEff}(id_R) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[a_1/f^M(1), ..., a_k/f^M(k)]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R^M_{data}(id_R, f^M(1), ..., f^M(k))$
(3) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{block}(id_R, true)$
(4) for each i :
$R^M_{exec}(id_R, x_1, ..., x_q) \wedge \pi_i(id_R) \rightsquigarrow R^M_{exec}(id_R, x_1, ..., x_q)[x_i/0]$
$R^M_{exec}(id_R, x_1, ..., x_q) \wedge \neg\pi_i(id_R) \rightsquigarrow R^M_{exec}(id_R, x_1, ..., x_q)[x_i/1]$
(5) [CopyRest] }

(1) substitutes attributes' values with the payload of the message; (2) propagates values to the message hub; (3) blocks the artifact instance; (4) initializes the eligibility flags for each PAC-rule, where $\pi_i(id_R)$ is a prerequisite of the i-th PAC-rule.

Translation 2 (**Immediate effect of 2-way service call generated by artifact instance).** Assume a 2-way service call type F within an atomic stage S_p, its associated artifact type R, input signature $(b_1 : Typ(b_1), \ldots, b_l : Typ(b_l))$ and output signature $(a'_1 : Typ(a'_1), \ldots, a'_k : Typ(a'_k))$ where $a'_i \in Att_{data}$.
Assume also a set of PAC-rules $\{(\pi_i, \alpha_i, \gamma_i)\}$.
Then a service call and an immediate effect of a service call return of type F on some artifact instance A of type R may be modeled by the following condition-action rule:

$$\exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge S_p = true \wedge R_{block}(id_R, false) \mapsto$$
$$\alpha_F^{call}(id_R) :$$
$\{$(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[a_1/f^F(\overline{b}, 1), ..., a_k/f^F(\overline{b}, k)]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R^F_{data}(id_R, f^F(\overline{b}, 1), ..., f^F(\overline{b}, k))$
(3) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{block}(id_R, true)$
(4) for each i :
$R^F_{exec}(id_R, x_1, ..., x_q) \wedge \pi_i(id_R) \rightsquigarrow R^F_{exec}(id_R, x_1, ..., x_q)[x_i/0]$
$R^F_{exec}(id_R, x_1, ..., x_q) \wedge \neg\pi_i(id_R) \rightsquigarrow R^F_{exec}(id_R, x_1, ..., x_q)[x_i/1]$
(5) [CopyRest]
$\}$, where

(1) substitutes attributes' values with the result of the service call; (2) propagates values to the message hub; (3) blocks the artifact instance; (4) initializes the eligibility flags for each PAC-rule, where $\pi_i(id_R)$ is a prerequisite of the i-th PAC-rule.

Translation 3 (**PAC-1 rule (Activating a stage)).**
Assume a stage S_j and its activating guard $g_j = [$ **on** $\xi(x)$ **if** $\phi(x)]$, where $\xi(x)$ is a triggering event and $\phi(x)$ is a condition. Then activating a stage S_j by validating g_j can be modeled by the following condition-action rule:
NB: Include term $(S' = true)$ if S' is parent of S_j.
NB: Note that prerequisite is checked on the stage of implementing immediate effect and, if not validated, will lead to marking x_k as 1, so will lead to skipping this CA-rule.
NB: Include effect propagating the outgoing message to the outgoing hub, if the stage to be activated is atomic and contains an action of sending a one-way message O with a signature $(b_1 : Typ(b_1), \ldots, b_k : Typ(b_k))$, where $b_i \in Att_{data}$.

$R_{exec}(id_R, \overline{x}) \wedge x_k = 0 \wedge exec(k) \wedge R_{block}(id_R, true) \mapsto$

$\alpha^k_{exec}(id_R, \overline{a'}, \overline{x}) : \{$

(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_\xi(id_R, \overline{a'}) \wedge S' = true \wedge \phi(id_R) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[S_j/true]$

(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_\xi(id_R, \overline{a'}) \wedge S' = true \wedge \phi(id_R) \rightsquigarrow R^{S_j}_{chg}(id_R, true)$

(3) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_\xi(id_R, \overline{a'}) \wedge S' = true \wedge \phi(id_R) \rightsquigarrow R^{O}_{out}(id_R, b_1, ..., b_k)$

(4) $R_{exec}(id_R, \overline{x}) \wedge x_k = 0 \rightsquigarrow R_{exec}(id_R, \overline{x})[x_k/1]$

(5) [CopyMessagePools]

(6) [CopyRest] $\}$,

where $exec(k) = \bigwedge_k x_k$ such that $r_k <_{PDG} r_a$

$R_\xi(id_R, \overline{a'}) = R_M(id_R, \overline{a'})$ if the guard contains incoming message event
or $R^{att}_{chg}(id_R, status_{new})$) if the guard contains internal event.

(1) activates a stage on a condition; (2) propagates internal event of opening a stage on a condition; (3) prepares eventual outgoing message for sending; (4) flags the microstep as performed.

Translation 4 (**PAC-2 rule (Milestone achiever)).**
Assume a stage S_j and its milestone m_j with achieving sentry [**on** $\xi(x)$ **if** $\phi(x)$], where $\xi(x)$ is a triggering event and $\phi(x)$ is a condition. Then achieving a milestone m_j can be modeled by the following condition-action rule:

$R^M_{exec}(id_R, \overline{x}) \wedge x_k = 0 \wedge exec(k) \wedge R_{block}(id_R, true) \mapsto$

$\alpha^k_{exec}(id_R, \overline{a'}, \overline{x}) : \{$

(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_\xi(id_R, \overline{a'}) \wedge S' = true \wedge \phi(id_R) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[m_j/true]$

(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_\xi(id_R, \overline{a'}) \wedge S' = true \wedge \phi(id_R) \rightsquigarrow R^{m_j}_{chg}(id_R, true)$

(3) $R_{exec}(id_R, \overline{x}) \wedge x_k = 0 \rightsquigarrow R_{exec}(id_R, \overline{x})[x_k/1]$

(4) [CopyMessagePools]

(5) [CopyRest] $\}$,

where $exec(k) = \bigwedge_k x_k$ such that $r_k <_{PDG} r_a$

$R_\xi(id_R, \overline{a'}) = R_M(id_R, \overline{a'})$ if the guard contains incoming message event
or $R^{att}_{chg}(id_R, status_{new})$) if the guard contains internal event.

(1) achieves a milestone on a condition; (2) propagates internal event of achieving a milestone on a condition; (3) flags the microstep as performed.

Translation 5 (**PAC-3 rule (Milestone invalidator)).**
Assume a stage S_j and its milestone m_j with invalidating sentry [**on** $\xi(x)$ **if** $\phi(x)$], where $\xi(x)$ is a triggering event and $\phi(x)$ is a condition. Then invalidating a milestone m_j can be modeled

by the following condition-action rule:

$$R^{M}_{exec}(id_R, \overline{x}) \wedge x_k = 0 \wedge exec(k) \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^{k}_{exec}(id_R, \overline{a'}, \overline{x}) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{\xi}(id_R, \overline{a'}) \wedge S' = true \wedge \phi(id_R) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[m_j/false]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{\xi}(id_R, \overline{a'}) \wedge S' = true \wedge \phi(id_R) \rightsquigarrow R^{m_j}_{chg}(id_R, false)$
(3) $R_{exec}(id_R, \overline{x}) \wedge x_k = 0 \rightsquigarrow R_{exec}(id_R, \overline{x})[x_k/1]$
(4) [CopyMessagePools]
(5) [CopyRest] $\}$,
where $exec(k) = \bigwedge_k x_k$ such that $r_k <_{PDG} r_a$

$R_{\xi}(id_R, \overline{a'}) = R_M(id_R, \overline{a'})$ if the guard contains incoming message event
or $R^{att}_{chg}(id_R, status_{new})$) if the guard contains internal event.

(1) invalidates a milestone on a condition; (2) propagates internal event of invalidating a milestone on a condition; (3) flags the microstep as performed.

Translation 6 (**PAC-4 rule (Opening stage invalidating milestone)).**
Assume a stage S_j and its milestone m_j. Then invalidating a milestone m_j caused by opening a stage can be modeled by the following condition-action rule:

$$R_{exec}(id_R, \overline{x}) \wedge x_k = 0 \wedge exec(k) \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^{k}_{exec}(id_R, \overline{a'}, \overline{x}) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{S_j}_{chg}(id_R, true)) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[m_j/false]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{S_j}_{chg}(id_R, true)) \rightsquigarrow R^{m_j}_{chg}(id_R, false)$
(3) $R^{M}_{exec}(id_R, \overline{x}) \wedge x_k = 0 \rightsquigarrow R^{M}_{exec}(id_R, \overline{x})[x_k/1]$
(4) [CopyMessagePools]
(5) [CopyRest] $\}$,
where $exec(k) = \bigwedge_k x_k$ such that $r_k <_{PDG} r_a$

(1) invalidates a milestone if the stage was open; (2) propagates internal event of invalidating a milestone; (3) flags the microstep as performed.

Translation 7 (**PAC-5 rule (Closing a stage on achieving milestone)).**
Assume a stage S_j and its milestone m_j. Then closing a stage S_j caused by achieving a milestone m_j can be modeled by the following condition-action rule:

$$R^{M}_{exec}(id_R, \overline{x}) \wedge x_k = 0 \wedge exec(k) \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^{k}_{exec}(id_R, \overline{a}, \overline{x}) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{m_j}_{chg}(id_R, true)) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[S_j/false]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{m_j}_{chg}(id_R, true)) \rightsquigarrow R^{S_j}_{chg}(id_R, false)$
(3) $R^{M}_{exec}(id_R, \overline{x}) \wedge x_k = 0 \rightsquigarrow R^{M}_{exec}(id_R, \overline{x})[x_k/1]$
(4) [CopyMessagePools]
(5) [CopyRest] $\}$,
where $exec(k) = \bigwedge_k x_k$ such that $r_k <_{PDG} r_a$

(1) closes a stage if the milestone was achieved; (2) propagates internal event of closing a stage; (3) flags the microstep as performed.

Translation 8 (**PAC-6 rule (No activity in a closed stage)).**
Assume a stage S_j and its parent stage S'. Then closing a stage S_j caused by closing its parent stage S' can be modeled by the following condition-action rule:

$$R^{M}_{exec}(id_R, \overline{x}) \wedge x_k = 0 \wedge exec(k) \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^{k}_{exec}(id_R, \overline{a'}, \overline{x}) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{S'}_{chg}(id_R, false)) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[S_j/false]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{S_j}_{chg}(id_R, false)) \rightsquigarrow R^{S_j}_{chg}(id_R, false)$
(3) $R^{M}_{exec}(id_R, \overline{x}) \wedge x_k = 0 \rightsquigarrow R^{M}_{exec}(id_R, \overline{x})[x_k/1]$
(4) [CopyMessagePools]
(5) [CopyRest] $\}$,
where $exec(k) = \bigwedge_k x_k$ such that $r_k <_{PDG} r_a$

(1) closes a stage if the parent stage is closed; (2) propagates internal event of closing a stage; (3) flags the microstep as performed.

Translation 9 (**Sending outgoing messages to the environment and flushing the message hubs).**
Assume a set of 1-way outgoing message types O_j obtained after all the PAC rules have been already taken into consideration. Then the conclusive part of a B-step, involving sending one-way outgoing messages and flushing the system message hubs may be modeled by the following CA-rule:

$$\exists R_{exec}(id_R, \overline{x}) \wedge \forall i\ x_i = 1 \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^{send}_{flush}(id_R) : \{$$
(1) $R_{exec}(id_R, \overline{x}) \wedge \forall i\ x_i = 1 \rightsquigarrow R_{block}(id_R, false)$
(2) $R_{exec}(id_R, \overline{x}) \wedge \forall i\ x_i = 1 \rightsquigarrow R_{exec}(id_R, \overline{0})$
(3) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})$
(4) for each j :
$$R^{O_j}_{out}(id_R, b_1, ..., b_k) \rightsquigarrow R_{result}(id_R, f^{O_j}(b_1, ..., b_k))$$
(5) [CopyRest] $\}$

(1) unblocks the artifact instance; (2) flushes the eligibility flags; (3) copies data; (4) sends outgoing messages to the environment.

Translation 10 (**Create artifact service call).**
Assume a particular kind of a 2-way service call - create artifact service call, within an atomic stage S_p. Assume also a set of PAC-rules $\{(\pi_i, \alpha_i, \gamma_i)\}$.
Then a create artifact service call and its immediate effect of a service call return may be

modeled by the following condition-action rule:

$$\exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge S_p = true \wedge R_{block}(id_R, false)$$
$$\mapsto \alpha_A^{create}(id_R, num) :$$
$$\{(1)\ \ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{att}(f_A^{create}(\tilde{a}), \tilde{a}, \overline{0}, \overline{0})$$
$$(3)\ \ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{data}^{srv-newA}(f_A^{create}(\tilde{a}))$$
$$(4)\ \ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{block}(id_R, true)$$
$$(5)\ \ \text{for each } i :$$
$$R_{exec}^F(id_R, x_1, ..., x_q) \wedge \pi_i(id_R) \rightsquigarrow R_{exec}^F(id_R, x_1, ..., x_q)[x_i/0]$$
$$R_{exec}^F(id_R, x_1, ..., x_q) \wedge \neg\pi_i(id_R) \rightsquigarrow R_{exec}^F(id_R, x_1, ..., x_q)[x_i/1]$$
$$(6)\ \ [\text{CopyRest}]\ \ \},$$
where $f_A^{create}(\tilde{a})$ returns ID of newly create artifact.

Translation 11 (**Remove artifact service call).**
Assume a particular kind of a 2-way service call - remove artifact service call, within an atomic stage S_p. Assume also a set of PAC-rules $\{(\pi_i, \alpha_i, \gamma_i)\}$.
Then a remove artifact service call and its immediate effect of a service call return may be modeled by the following condition-action rule:

$$\exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge S_p = true \wedge R_{block}(id_R, false)$$
$$\mapsto \alpha_A^{remove}(id_R, num) :$$
$$(2)\ \ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{data}^{srv-remA}(f_A^{remove}(\overline{a}))$$
$$(3)\ \ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{block}(id_R, true)$$
$$(5)\ \ \text{for each } i :$$
$$R_{exec}^F(id_R, x_1, ..., x_q) \wedge \pi_i(id_R) \rightsquigarrow R_{exec}^F(id_R, x_1, ..., x_q)[x_i/0]$$
$$R_{exec}^F(id_R, x_1, ..., x_q) \wedge \neg\pi_i(id_R) \rightsquigarrow R_{exec}^F(id_R, x_1, ..., x_q)[x_i/1]$$
$$(6)\ \ [\text{CopyRest}]\ \ \},$$
where $f_A^{remove}(\overline{a})$ returns the outcome of deletion.

A.3 An Example

Let us consider a process *Func* which is as simple as calculating a sum $a + b$, given that $a \neq b$. The GSM concrete model of such process is, in fact, represented by the first stage in Figure 26.

Then the corresponding artifact type has the form

$$(R, x, Att_{data} \cup Att_{status}, Typ, Stg, Mst, Lcyc),$$

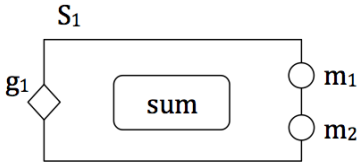


Figure 26 – GSM model of $(a + b)$

where[20]

$$\begin{aligned} Att_{data} &= \{A_{ID}, a, b, c\} \\ Att_{status} &= \{s_1, m_1, m_2\}, \text{ all Boolean} \\ Type &= \{(a, Float), (b, Float), (c, Float)\} \\ S &= \{s_1\}, \quad M = \{m_1, m_2\} \end{aligned}$$

Consequently, lifecycle *Lcyc* has the following form:

$$\begin{aligned} &(Substages, Task, Owns, Guards, Ach, Inv), \text{ where} \\ Substages &= \emptyset \\ Task &= \{(s_1, Sum)\} \\ Owns &= \{(s_1, \{m_1, m_2\})\} \\ Guards &= \{(s_1, \{\widetilde{g}_1\})\} \\ Ach &= \{(m_1, \{\widetilde{m}_1\}), (m_2, \{\widetilde{m}_2\})\} \\ Inv &= \emptyset \end{aligned}$$

Corresponding sentries for guards and milestones are given below:

$$\begin{aligned} \widetilde{g}_1 &: \text{on } x.Func^{call}(a, b) \text{ if } a \neq b \\ \widetilde{m}_1 &: \text{on } x.Sum^{return}(c) \text{ if } c \geq 0 \\ \widetilde{m}_2 &: \text{if } c < 0 \end{aligned}$$

Intuitively, the workflow of the given process is the following:

- after receiving a request from the environment, if the operands are not equal, the first stage is activated;
- the task associated with the first atomic stage is executed, calling the external service *Sum* with given parameters and obtaining the result;
- upon completing the request, if the result is positive, then milestone m_1 is achieved;
- if attribute storing the result of operation is negative, then milestone m_2 is achieved. Note that the corresponding sentry $\widetilde{m}_2$ doesn't contain event expression of receiving service call return. Thus, if the stage gets activated with $c < 0$, this milestone will be achieved immediately.

PAC rules

Type	ID	Prerequisite	Antecedent	Consequent
PAC-1	x1	$\neg x.s_1$	**on** $x.Func^{call}(a, b)$ **if** $a \neq b$	$+x.s_1$
PAC-2	x2	$x.s_1$	**on** $x.Sum^{return}(c)$ **if** $c \geq 0$	$+x.m_1$
PAC-2	x3	$x.s_1$	**if** $c < 0$	$+x.m_2$
PAC-4	x4	$x.m_1$	**on** $+x.s_1$	$-x.m_1$
PAC-4	x5	$x.m_2$	**on** $+x.s_1$	$-x.m_2$
PAC-5	x6	$x.s_1$	**on** $+x.m_1$	$-x.s_1$
PAC-5	x7	$x.s_1$	**on** $+x.m_2$	$-x.s_1$

[20]For the sake of simplicity, we will omit attributes as *mostRecEventType* and *mostRecEventTime*. Similarly, we will omit attributes like $m^{mostRecentUpdate}$ and $active_S^{mostRecentUpdate}$.

DCDS Translation

Data layer

The corresponding data layer $\mathcal{D} = \langle \mathcal{C}, \mathcal{R}, \mathcal{E}, \mathcal{I}_0 \rangle$ will have the following form:

- $\mathcal{C} = \bigcup_i \mathbf{DOM}(Typ(Att_i))$,
- $\mathcal{R} = \{R_{att}\} \cup \{R^{m_i}_{chg} \mid m_i \in Mst\} \cup \{R^{S_j}_{chg} \mid S_j \in Stg\} \cup$
  $\{R^{msg_k}_{data} \mid msg_k \in \mathbf{MSG}$ and msg_k is incoming 1-way message$\} \cup$
  $\{R^{srv_p}_{data} \mid srv_p \in \mathbf{SRV}$ and srv_p is a service call return$\} \cup$
  $\{R^{msg_q}_{out} \mid msg_q \in \mathbf{MSG}$ and msg_q is outgoing 1-way message$\} \cup$
  $\{R_{exec}, R_{block}\}$,

  where R_{att} stores the attributes of an artifact, R_{exec} keeps information on which PAC rules have been taken in consideration while other relations are used to model the incoming / outgoing message pool:

  - $R_{att} = (id_A, a, b, c, s_1, m_1, m_2)$,
  - $R_{exec} = (id_A, x1, x2, x3, x4, x5, x6, x7)$,
  - $R_{block} = (id_A, blocked)$,
  - $R^{s_1}_{chg} = (id_A, newstate)$,
  - $R^{m_1}_{chg} = (id_A, newstate)$,
  - $R^{m_2}_{chg} = (id_A, newstate)$,
  - $R^{Func}_{data} = (id_A, a, b)$,
  - $R^{Sum}_{data} = (id_A, c)$.
- $\mathcal{E} = \emptyset$.
- $\mathcal{I}_0 = \emptyset$.

Immediate effect rules

Incoming message $Func$:

$$\exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{block}(id_R, false) \mapsto$$
$$\alpha^{ImmEff}_{Func}(id_R) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[a/f^{Func}(1), b/f^{Func}(2)]$

(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R^{Func}_{data}(id_R, f^{Func}(1), f^{Func}(2))$

(3) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{block}(id_R, true)$

(4) $R_{exec}(id_R, \overline{x}) \wedge \neg S_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x1/0, x2/1, x3/1, x6/1, x7/1]$
$R_{exec}(id_R, \overline{x}) \wedge S_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x1/1, x2/0, x3/0, x6/0, x7/0]$

$R_{exec}(id_R, \overline{x}) \wedge m_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x4/0]$
$R_{exec}(id_R, \overline{x}) \wedge \neg m_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x4/1]$

$R_{exec}(id_R, \overline{x}) \wedge m_2 \rightsquigarrow R_{exec}(id_R, \overline{x})[x5/0]$
$R_{exec}(id_R, \overline{x}) \wedge \neg m_2 \rightsquigarrow R_{exec}(id_R, \overline{x})[x5/1]$

(5) [CopyRest] $\}$

Service call return Sum:

$$\exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge S_1 = true \wedge R_{block}(id_R, false) \mapsto$$
$$\alpha^{call}_{Sum}(id_R) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[c/f^{Sum}(a, b, 1)]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R^{Sum}_{data}(id_R, f^{Sum}(a, b, 1))$
(3) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{block}(id_R, true)$
(4) $R_{exec}(id_R, \overline{x}) \wedge \neg S_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x1/0, x2/1, x3/1, x6/1, x7/1]$
$R_{exec}(id_R, \overline{x}) \wedge S_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x1/1, x2/0, x3/0, x6/0, x7/0]$
$R_{exec}(id_R, \overline{x}) \wedge m_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x4/0]$
$R_{exec}(id_R, \overline{x}) \wedge \neg m_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x4/1]$
$R_{exec}(id_R, \overline{x}) \wedge m_2 \rightsquigarrow R_{exec}(id_R, \overline{x})[x5/0]$
$R_{exec}(id_R, \overline{x}) \wedge \neg m_2 \rightsquigarrow R_{exec}(id_R, \overline{x})[x5/1]$
(5) [CopyRest] }

PAC rules
PAC-1 rule x_1:

$$R_{exec}(id_R, \overline{x}) \wedge x_1 = 0 \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^{1}_{exec}(id_R, a, b, \overline{x}) : \{$$
(1) $R^{Func}_{data}(id_R, a, b) \wedge R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge a \neq b \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[S_1/true]$
(2) $R^{Func}_{data}(id_R, a, b) \wedge R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge a \neq b \rightsquigarrow R^{S_1}_{chg}(id_R, true)$
(3) $R^{M}_{exec}(id_R, \overline{x}) \wedge x_1 = 0 \rightsquigarrow R^{M}_{exec}(id_R, \overline{x})[x_1/1]$
(4) [CopyMessagePools]
(5) [CopyRest] }

PAC-2 rule x_2:

$$R_{exec}(id_R, \overline{x}) \wedge x_2 = 0 \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^{2}_{exec}(id_R, c, \overline{x}) : \{$$
(1) $R^{Sum}_{data}(id_R, c) \wedge R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge c \geq 0 \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[m_1/true]$
(2) $R^{Sum}_{data}(id_R, c) \wedge R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge c \geq 0 \rightsquigarrow R^{m_1}_{chg}(id_R, true)$
(3) $R_{exec}(id_R, \overline{x}) \wedge x_2 = 0 \rightsquigarrow R_{exec}(id_R, \overline{x})[x_2/1]$
(4) [CopyMessagePools]
(5) [CopyRest] }

PAC-2 rule x_3:

$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_3 = 0 \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^{3}_{exec}(id_R, \overline{a}, \overline{x}) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge c < 0 \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[m_2/true]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge c < 0 \rightsquigarrow R^{m_2}_{chg}(id_R, true)$
(3) $R_{exec}(id_R, \overline{x}) \wedge x_3 = 0 \rightsquigarrow R_{exec}(id_R, \overline{x})[x_3/1]$
(4) [CopyMessagePools]
(5) [CopyRest] }

PAC-4 rule x_4:

$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_4 = 0 \wedge x_1 = 1 \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^4_{exec}(id_R, \overline{a}, \overline{x}) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{S_1}_{chg}(id_R, true)) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[m_1/false]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{S_1}_{chg}(id_R, true)) \rightsquigarrow R^{m_1}_{chg}(id_R, false)$
(3) $R_{exec}(id_R, \overline{x}) \wedge x_4 = 0 \rightsquigarrow R_{exec}(id_R, \overline{x})[x_4/1]$
(4) [CopyMessagePools]
(5) [CopyRest] }

PAC-4 rule x_5:

$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_5 = 0 \wedge x_1 = 1 \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^5_{exec}(id_R, \overline{a}, \overline{x}) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{S_1}_{chg}(id_R, true)) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[m_2/false]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{S_1}_{chg}(id_R, true)) \rightsquigarrow R^{m_2}_{chg}(id_R, false)$
(3) $R_{exec}(id_R, \overline{x}) \wedge x_5 = 0 \rightsquigarrow R_{exec}(id_R, \overline{x})[x_5/1]$
(4) [CopyMessagePools]
(5) [CopyRest] }

PAC-5 rule x_6:

$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_6 = 0 \wedge x_2 = 1 \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^6_{exec}(id_R, \overline{a}, \overline{x}) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{m_1}_{chg}(id_R, true)) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[S_1/false]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{m_1}_{chg}(id_R, true)) \rightsquigarrow R^{S_1}_{chg}(id_R, false)$
(3) $R_{exec}(id_R, \overline{x}) \wedge x_6 = 0 \rightsquigarrow R_{exec}(id_R, \overline{x})[x_6/1]$
(4) [CopyMessagePools]
(5) [CopyRest] }

PAC-5 rule x_7:

$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_7 = 0 \wedge x_3 = 1 \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^7_{exec}(id_R, \overline{a}, \overline{x}) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{m_2}_{chg}(id_R, true)) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[S_1/false]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{m_2}_{chg}(id_R, true)) \rightsquigarrow R^{S_1}_{chg}(id_R, false)$
(3) $R_{exec}(id_R, \overline{x}) \wedge x_7 = 0 \rightsquigarrow R_{exec}(id_R, \overline{x})[x_7/1]$
(4) [CopyMessagePools]
(5) [CopyRest] }

Sending outgoing messages and unblocking the artifact instance

$$\exists R_{exec}(id_R, \overline{x}) \wedge \forall i\ x_i = 1 \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^{send}_{flush}(id_R) : \{$$
(1) $R_{exec}(id_R, \overline{x}) \wedge \forall i\ x_i = 1 \rightsquigarrow R_{block}(id_R, false)$
(2) $R_{exec}(id_R, \overline{x}) \wedge \forall i\ x_i = 1 \rightsquigarrow R_{exec}(id_R, \overline{0})$
(3) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})$
(4) for each j :
$$R^{O_j}_{out}(id_R, b_1, ..., b_k) \rightsquigarrow R_{result}(id_R, f^{O_j}(b_1, ..., b_k))$$
(5) [CopyRest] }

Execution Semantics

Let us now follow the construction of a transition system resulting from the obtained translation. We start with an initial state I_0 such that:

- $s_1 = m_1 = m_2 = 0$
- $R_{block}(id_R, false)$
- $R_{exec}(id_R, \overline{0})$
- $R^{Sum}_{data} = R^{Func}_{data} = \emptyset$

Let us evaluate the condition part of all CA-rules and mark with $(*)$ applicable ones:

$(*)$ $\exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{block}(id_R, false) \mapsto \{\}$
$\exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge S_1 = true \wedge R_{block}(id_R, false) \mapsto \{\}$
$R_{exec}(id_R, \overline{x}) \wedge x_1 = 0 \wedge R_{block}(id_R, true) \mapsto \{\}$
$R_{exec}(id_R, \overline{x}) \wedge x_2 = 0 \wedge R_{block}(id_R, true) \mapsto \{\}$
$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_3 = 0 \wedge R_{block}(id_R, true) \mapsto \{\}$
$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_4 = 0 \wedge x_1 = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$
$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_5 = 0 \wedge x_1 = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$
$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_6 = 0 \wedge x_2 = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$
$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_7 = 0 \wedge x_3 = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$
$\exists R_{exec}(id_R, \overline{x}) \wedge \forall i\ x_i = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$

So, only the first rule is applicable, let us check what it looks like:

$$\exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{block}(id_R, false) \mapsto$$
$$\alpha^{ImmEff}_{Func}(id_R) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[a/f^{Func}(1), b/f^{Func}(2)]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R^{Func}_{data}(id_R, f^{Func}(1), f^{Func}(2))$
(3) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{block}(id_R, true)$
(4) $R_{exec}(id_R, \overline{x}) \wedge \neg S_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x1/0, x2/1, x3/1, x6/1, x7/1]$
$R_{exec}(id_R, \overline{x}) \wedge S_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x1/1, x2/0, x3/0, x6/0, x7/0]$
$R_{exec}(id_R, \overline{x}) \wedge m_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x4/0]$
$R_{exec}(id_R, \overline{x}) \wedge \neg m_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x4/1]$
$R_{exec}(id_R, \overline{x}) \wedge m_2 \rightsquigarrow R_{exec}(id_R, \overline{x})[x5/0]$
$R_{exec}(id_R, \overline{x}) \wedge \neg m_2 \rightsquigarrow R_{exec}(id_R, \overline{x})[x5/1]$
(5) [CopyRest] }

The first effect is to obtain input from the environment by calling the function f^{Func}. Second effect propagates obtained data to the inner message pool. (3) Blocks the artifact instance. Effects from (4) decide which CA-rules are relevant. For our current state, we get that: $x1 = 0$ and all the rest $x_i = 1$. Which means that only $x1$ is relevant.

Thus, we get the following situation:

- $s_1 = m_1 = m_2 = 0$
- $R_{block}(id_R, true)$
- $R_{exec}(id_R, (0, 1, ..., 1))$
- $R^{Sum}_{data} = \emptyset, R^{Func}_{data} = (a, b)$

Let us evaluate again the condition part of all CA-rules and mark with $(*)$ applicable ones:

$$
\begin{array}{ll}
 & \exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{block}(id_R, false) \mapsto \{\} \\
 & \exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge S_1 = true \wedge R_{block}(id_R, false) \mapsto \{\} \\
(*) & R_{exec}(id_R, \overline{x}) \wedge x_1 = 0 \wedge R_{block}(id_R, true) \mapsto \{\} \\
 & R_{exec}(id_R, \overline{x}) \wedge x_2 = 0 \wedge R_{block}(id_R, true) \mapsto \{\} \\
 & R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_3 = 0 \wedge R_{block}(id_R, true) \mapsto \{\} \\
 & R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_4 = 0 \wedge x_1 = 1 \wedge R_{block}(id_R, true) \mapsto \{\} \\
 & R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_5 = 0 \wedge x_1 = 1 \wedge R_{block}(id_R, true) \mapsto \{\} \\
 & R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_6 = 0 \wedge x_2 = 1 \wedge R_{block}(id_R, true) \mapsto \{\} \\
 & R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_7 = 0 \wedge x_3 = 1 \wedge R_{block}(id_R, true) \mapsto \{\} \\
 & \exists R_{exec}(id_R, \overline{x}) \wedge \forall i\ x_i = 1 \wedge R_{block}(id_R, true) \mapsto \{\}
\end{array}
$$

So, only the third rule is applicable, let us check what it looks like:

$$
\begin{array}{l}
R_{exec}(id_R, \overline{x}) \wedge x_1 = 0 \wedge R_{block}(id_R, true) \mapsto \\
\quad \alpha^1_{exec}(id_R, a, b, \overline{x}) : \{ \\
\quad (1)\ \ R^{Func}_{data}(id_R, a, b) \wedge R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge a \neq b \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[S_1/true] \\
\quad (2)\ \ R^{Func}_{data}(id_R, a, b) \wedge R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge a \neq b \rightsquigarrow R^{S_1}_{chg}(id_R, true) \\
\quad (3)\ \ R^{M}_{exec}(id_R, \overline{x}) \wedge x_1 = 0 \rightsquigarrow R^{M}_{exec}(id_R, \overline{x})[x_1/1] \\
\quad (4)\ \ [\mathsf{CopyMessagePools}] \\
\quad (5)\ \ [\mathsf{CopyRest}]\ \ \}
\end{array}
$$

At this point there are 2 possible cases: when $a \neq b$ and $a = b$.

If $a = b$ the first 2 effects are ignored and we just mark $x_1 = 1$ and are done. Then we go to the last rule, which unblocks the artifact instance.

If $a \neq b$, then we open a stage s_1 and propagate this event to the inner message pool. We also mark $x_1 = 1$. So, we have the following situation:

- $s_1 = 1, m_1 = m_2 = 0$
- $R_{block}(id_R, true)$
- $R_{exec}(id_R, (1, 1, ..., 1))$
- $R^{Sum}_{data} = \emptyset, R^{Func}_{data} = (a, b)$

Let us evaluate again the rules:

$$\exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{block}(id_R, false) \mapsto \{\}$$
$$\exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge S_1 = true \wedge R_{block}(id_R, false) \mapsto \{\}$$
$$R_{exec}(id_R, \overline{x}) \wedge x_1 = 0 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$R_{exec}(id_R, \overline{x}) \wedge x_2 = 0 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_3 = 0 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_4 = 0 \wedge x_1 = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_5 = 0 \wedge x_1 = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_6 = 0 \wedge x_2 = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_7 = 0 \wedge x_3 = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$(*)\ \exists R_{exec}(id_R, \overline{x}) \wedge \forall i\ x_i = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$$

The last rule is applicable:

$$\exists R_{exec}(id_R, \overline{x}) \wedge \forall i\ x_i = 1 \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^{send}_{flush}(id_R) : \{$$
$$(1)\ \ R_{exec}(id_R, \overline{x}) \wedge \forall i\ x_i = 1 \rightsquigarrow R_{block}(id_R, false)$$
$$(2)\ \ R_{exec}(id_R, \overline{x}) \wedge \forall i\ x_i = 1 \rightsquigarrow R_{exec}(id_R, \overline{0})$$
$$(3)\ \ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})$$
$$(4)\ \ \text{for each } j:$$
$$R^{O_j}_{out}(id_R, b_1, ..., b_k) \rightsquigarrow R_{result}(id_R, f^{O_j}(b_1, ..., b_k))$$
$$(5)\ \ [\text{CopyRest}]\ \ \}$$

So we have:

- $s_1 = 1, m_1 = m_2 = 0$
- $R_{block}(id_R, false)$
- $R_{exec}(id_R, (0, 0, ..., 0))$
- $R^{Sum}_{data} = \emptyset, R^{Func}_{data} = \emptyset$

Let us evaluate again the rules:

$$(*)\ \exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{block}(id_R, false) \mapsto \{\}$$
$$(*)\ \exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge S_1 = true \wedge R_{block}(id_R, false) \mapsto \{\}$$
$$R_{exec}(id_R, \overline{x}) \wedge x_1 = 0 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$R_{exec}(id_R, \overline{x}) \wedge x_2 = 0 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_3 = 0 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_4 = 0 \wedge x_1 = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_5 = 0 \wedge x_1 = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_6 = 0 \wedge x_2 = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{exec}(id_R, \overline{x}) \wedge x_7 = 0 \wedge x_3 = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$$
$$\exists R_{exec}(id_R, \overline{x}) \wedge \forall i\ x_i = 1 \wedge R_{block}(id_R, true) \mapsto \{\}$$

Here comes a non-deterministic choice of the rule to apply. Let us, say, apply the first one:

$$\exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{block}(id_R, false) \mapsto$$
$$\alpha_{Func}^{ImmEff}(id_R) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[a/f^{Func}(1), b/f^{Func}(2)]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{data}^{Func}(id_R, f^{Func}(1), f^{Func}(2))$
(3) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{block}(id_R, true)$
(4) $R_{exec}(id_R, \overline{x}) \wedge \neg S_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x1/0, x2/1, x3/1, x6/1, x7/1]$
$R_{exec}(id_R, \overline{x}) \wedge S_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x1/1, x2/0, x3/0, x6/0, x7/0]$
$R_{exec}(id_R, \overline{x}) \wedge m_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x4/0]$
$R_{exec}(id_R, \overline{x}) \wedge \neg m_1 \rightsquigarrow R_{exec}(id_R, \overline{x})[x4/1]$
$R_{exec}(id_R, \overline{x}) \wedge m_2 \rightsquigarrow R_{exec}(id_R, \overline{x})[x5/0]$
$R_{exec}(id_R, \overline{x}) \wedge \neg m_2 \rightsquigarrow R_{exec}(id_R, \overline{x})[x5/1]$
(5) [CopyRest] $\}$

Then what we get is:

- $s_1 = 1, m_1 = m_2 = 0$
- $R_{block}(id_R, true)$
- $R_{exec}(id_R, (1, 0, 0, 1, 1, 0, 0))$
- $R_{data}^{Sum} = \emptyset, R_{data}^{Func} = (a, b)$

So we have to apply $x2, x3, x6, x7$.

A.4 Correctness of the Translation

The proof plan:

1. Prove that for each micro-step of the GSM model, the corresponding DCDS CA-rule results in the same state (pre-snapshot) of the model, w.r.t. data and status attributes.
2. Prove that for each GSM B-step (certain path in the resulting transition system) there exists a corresponding execution in the DCDS transition system.
3. Prove that for each execution path in the DCDS transition system, there exists a corresponding one in GSM.

Lemma A.1. *For each micro-step, consisting of applying a ground PAC rule $(\pi_k, \alpha_k, \gamma_k)$ to a pre-snapshot Σ_j, the corresponding translation of this rule – a DCDS condition-action rule $Q_k \mapsto \alpha_k(p_1, \ldots, p_q) : \{e_1, \ldots, e_m\}$, results in the same pre-snapshot Σ_{j+1} w.r.t. data and status attributes.*

Proof. First, we have to prove that the CA-rule, corresponding to computing the $\Sigma_1 = \mathit{ImmEffect}(\Sigma, e, t)$, results in the same pre-snapshot as $\mathit{ImmEffect}(\Sigma, e, t)$.

By definition of the immediate effect of an incoming event $e = M(A_1 : c_1, \ldots, A_n : c_n)$, $\mathit{ImmEffect}(\Sigma, e, t)$ is a pre-snapshot Σ' obtained from Σ by modifying the corresponding artifact instance I in the following way[21]: for each data attribyte A_i, set $I.A_i := c_i$.

[21] As mentioned earlier, we omit *mostRecEventType* and *mostRecEventTime*.

Let us now consider the corresponding CA-rule:

$$\exists \overline{a}, \overline{s}, \overline{m}\ R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_{block}(id_R, false) \mapsto$$
$$\alpha_M^{ImmEff}(id_R) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[a_1/f^M(1), ..., a_k/f^M(k)]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R^M_{data}(id_R, f^M(1), ..., f^M(k))$
(3) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \rightsquigarrow R_{block}(id_R, true)$
(4) for each i :
$$R^M_{exec}(id_R, x_1, ..., x_q) \wedge \pi_i(id_R) \rightsquigarrow R^M_{exec}(id_R, x_1, ..., x_q)[x_i/0]$$
$$R^M_{exec}(id_R, x_1, ..., x_q) \wedge \neg\pi_i(id_R) \rightsquigarrow R^M_{exec}(id_R, x_1, ..., x_q)[x_i/1]$$
(5) [CopyRest] }

The condition part of this CA-rule selects an artifact instance and checks whether it is able to process the message, i.e. is not busy with processing another one. Then, Effect (1) of the action does exactly what is required by the definition of *ImmEffect* – it changes the data attributes affected by the payload of e. Effect (2) propagates the values of the incoming event to the system message hub, which is used by the DCDS engine to encode the execution of a model. Thus, we can abstract from it, as well from Effect (3), which blocks the artifact instance from receiving other messages until the current message is fully processed. Also Effect (4) may be abstracted away, since it is also a system information. It should be noted, though, that the Effect (4) implements the step of selecting applicable CA-rules according to the incremental semantics of GSM. For those CA-rules whose prerequisite is valid ($\pi_i(id_R) == true$), the corresponding CA-rule is marked as 0, i.e., to be taken into consideration. Not eligible rules are marked with 1, i.e. already taken into consideration. Therefore, since:

- it is assumed that in GSM incoming messages "are processed by the artifact instances one at a time",
- corresponding CA-rule uses its own blocking mechanism to ensure this;
- the only effect changing data attributed is Effect (1), which is applied whenever an action is fired (i.e., without any condition),
- Effect (1) strictly corresponds to the definition of the *ImmEffect* in GSM,
- no other effect involves either data or status attributes,

then it may be claimed that the DCDS pre-snapshot obtained after firing the corresponding CA-rule coincides with the GSM pre-snapshot $\Sigma_1 = ImmEffect(\Sigma, e, t)$ w.r.t. to data and status attributes. Similarly it can be shown for the case of service call return. The only difference would be a condition of an atomic stage to be activated in order to enable service call.

Now let us get down to proving correspondence between PAC rules and their translations.

Consider, for instance, a PAC-1 rule $(\pi_k, \alpha_k, \gamma_k)$ corresponding to a certain micro-step in the incremental foundation. We have to prove that a corresponding DCDS CA-rule is eligible to apply the effect γ if and only if the PAC-1 rule is eligible to apply the effect γ and that the effect of firing this rule will result in the coinciding pre-snapshot w.r.t. to data and status attributes.

The PAC-1 rule is eligible to apply the effect γ if and only if each of the following holds:

- $\Sigma \vDash \pi_k$, i.e. the prerequisite is met.
- $\Sigma_j \vDash \alpha_k$, for $\alpha_k = [$ **on** $\xi(x)$ **if** $\phi(x)]$, i.e. the antecedent is satisfied.
- the ordering implied by $PDG(\Gamma)$ is respected, i.e. for each pair (r, r') of ground rules with abstract actions $\odot R.s$ and $\odot' R'.s'$, respectively, if $\odot R.s < \odot' R'.s'$, then the rule r must be considered for firing before the rule r' is considered for firing.

Now let us consider the translation of the PAC-1 rule to DCDS CA-rule and show that the conditions for applying the effect γ coincide with those listed above. The corresponding CA-rule looks like the following:

$$R_{exec}(id_R, \overline{x}) \wedge x_k = 0 \wedge exec(k) \wedge R_{block}(id_R, true) \mapsto$$
$$\alpha^k_{exec}(id_R, \overline{a'}, \overline{x}) : \{$$
(1) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_\xi(id_R, \overline{a'}) \wedge S' = true \wedge \phi(id_R) \rightsquigarrow R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[S_j/true]$
(2) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_\xi(id_R, \overline{a'}) \wedge S' = true \wedge \phi(id_R) \rightsquigarrow R^{S_j}_{chg}(id_R, true)$
(3) $R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R_\xi(id_R, \overline{a'}) \wedge S' = true \wedge \phi(id_R) \rightsquigarrow R^{O}_{out}(id_R, b_1, ..., b_k)$
(4) $R_{exec}(id_R, \overline{x}) \wedge x_k = 0 \rightsquigarrow R_{exec}(id_R, \overline{x})[x_k/1]$
(5) [CopyMessagePools]
(6) [CopyRest] $\}$,

where $exec(k) = \bigwedge_k x_k$ such that $r_k <_{PDG} r_a$

$R_\xi(id_R, \overline{a'}) = R_M(id_R, \overline{a'})$ if the guard contains incoming message event
or $R^{att}_{chg}(id_R, status_{new})$) if the guard contains internal event.

The condition part of this CA-rule obtains the current state of the execution plan for this event and insures that this CA-rule has not yet been taken into consideration ($x_k = 0$) and that all the preceding CA-rules have been taking into consideration ($exec(k)$).

Assume that $\Sigma \nvDash \pi_k$. Then, since each micro-step is preceded by the *ImmEffect*, then the CA-rule implementing the immediate effect of the event has already been fired. Since it has been fired then for each i either of the effects has been applied:

$$R^F_{exec}(id_R, x_1, \ldots, x_q) \wedge \pi_i(id_R) \rightsquigarrow R^F_{exec}(id_R, x_1, \ldots, x_q)[x_i/0]$$
$$R^F_{exec}(id_R, x_1, \ldots, x_q) \wedge \neg\pi_i(id_R) \rightsquigarrow R^F_{exec}(id_R, x_1, \ldots, x_q)[x_i/1]$$

Since $\Sigma \nvDash \pi_k$, then it is the second effect that has been applied, so $x_k = 1$, which prevents our CA-rule to fire, since the condition part is not met.

Now let us assume that the prerequisite holds, but the ordering implied by $PDG(\Gamma)$ is not yet respected, i.e. there exists a rule r', such that $\odot' R'.s' < +R.active_S$ and that it has not yet been considered. Then the executability flag x' will be equal to 0, which will prevent rule from firing, since the condition part of the CA-rule contains a check $x' = 1$.

Now let us assume that $\Sigma \vDash \pi_k$, the ordering is respected, but $\Sigma_j \nvDash \alpha_k$, so either **on** $\xi(x)$ hasn't happened or $\Sigma \nvDash \phi_k$. Then the CA-rule will be eligible to fire, however, the effects (1-3) will not be applied, since the query for instantiating them will be empty:

- in case $\Sigma_j \nvDash \phi_k$ – it is obvious.
- in case **on** $\xi(x)$ hasn't happened, this means that the corresponding record hasn't been put into the R_ξ, therefore R_ξ will be empty.

Then the only effects that will possible take place are (3)–(6), which do not deal with any data or status attributes and, however, mark this PAC-rule as considered, so that rules dependent on this one could proceed.

Not let us assume that $\Sigma \vDash \pi_k$, $\Sigma_j \vDash \alpha_k$ and the ordering is respected. Then the effect will be applied and will result in toggling the status attribute $R.active_S$ to *true*. None of the remaining effects deal with data or status attribute, so the resulting DCDS pre-snapshot will coincide to that of GSM micro-step w.r.t. to data and status attributes.

The proof for other PAC rules can be formulated similarly to PAC-1.

The proof for CA-rule of sending a set of outgoing one-way messages once all the PAC rules have been taken into consideration, can be formulated similarly to the *ImmEffect* rule. □

We now have to prove the second and the third statement of the proof plan.

Lemma A.2. *Given an artifact instance A_R for each possible GSM B-step (i.e. a sequence of micro-steps preceded by ImmEff and followed by the step of sending outgoing messages to the environment) there exists a corresponding execution in the DCDS transition system.*

Proof. In order to prove this statement we have to prove that the mechanism used by the DCDS translation to restrict the possible sequences of CA-rules results in the same order imposed by the PDG graph of the GSM model.

Let us assume we have a sequence of PAC-rules $\Gamma_{PAC} = \{r_i = (\pi_i, \alpha_i, \gamma_i)\}$, preceded by the *ImmEffect* micro-step, which respect the order imposed by the PDG graph constructed for the given GSM model. We now have to prove that for the set of corresponding CA-rules $\{tr(r_i)\}$ the following holds:

$$\forall r_m, r_n \in \Gamma_{PAC} \text{ if } r_m <_{PDG} r_n \text{ then for any path in DCDS transition system,}$$
$$tr(r_m) \text{ is considered for firing before } tr(r_n) \text{ is considered for firing.}$$

Assume $r_n = (\pi, \alpha, \odot R.s)$ and $r_m = (\pi', \alpha', \odot' R'.s')$. Then $\odot R.s < \odot' R'.s'$. This means that, by construction of PDG, α' contains $\odot R.s$ as a triggering event (or contains $R.s$ in its condition).

Let us now consider corresponding DCDS CA-rules:

$$tr(r_m) = Q \mapsto act$$
$$tr(r_n) = Q' \mapsto act'$$

By definition of DCDS translation, $exec(n) \in Q'$ where:

for each PAC rule $r_k = (\pi_k, \alpha_k, \gamma_k)$ the expression $exec(k)$ for the corresponding CA-rule $tr(r_k)$ is defined as follows:

$$exec(k) = \bigwedge_j x_j \text{ such that } r_j <_{PDG} r_k \text{ (i.e. } \gamma_j \in \alpha_k),$$

where x_j is a boolean flag that shows whether the CA-rule $tr(r_j)$ has been taken into consideration. By construction of the DCDS translation, the boolean flag x_j is only affected in the CA-rule $tr(r_j)$ and in the first micro-step incorporating the immediate effect. The $ImmEff$ micro-step checks the prerequisite of a PAC rule in order to set the value of x_j. Assume it is set to 1, which means that prerequisite is not satisfied and therefore r_j cannot influence α_k, so r_k can fire, which is totally valid. Assume now it is set to 0, which means that r_j is applicable and should be taken into consideration. Then, the only place in the translation it may be affected is the $tr(r_j)$ and it will be, in fact, changed to 1, whenever this CA-rule will be nondeterministically chosen to fire. Till then it will be 0, which will prevent $tr(r_k)$ from firing.

Therefore $tr(r_n)$ will not be taken into consideration unless $tr(r_m)$ has been taken into consideration. □

Lemma A.3. *Given an artifact instance A_R, its GSM model and a corresponding DCDS translation, for each possible execution in DCDS starting with Immediate Effect rule, there exists a corresponding B-step in GSM model, which results in the same next pre-snapshot Σ_{j+1} w.r.t. data and status attributes.*

Proof. The proof is done by construction of CA-rules, see Section A.2. □

Given a GSM model $\mathcal{G}$ with initial snapshot S_0, we denote by $\Upsilon_{\mathcal{G}}$ its *B-step transition system*, i.e., the infinite-state transition system obtained by iteratively applying the incremental GSM semantics starting from S_0 and nondeterministically considering each possible incoming event. The states of $\Upsilon_{\mathcal{G}}$ correspond to stable snapshots of $\mathcal{G}$, and each transition corresponds to a B-step. We abstract away from the single micro-steps constituting a B-step, because they represent temporary intermediate states that are not interesting for verification purposes. Similarly, given

the DCDS $\mathcal{S}$ obtained from the translation of $\mathcal{G}$, we denote by $\Upsilon_{\mathcal{S}}$ its *unblocked-state transition system*, obtained by starting from S_0, and iteratively applying nondeterministically the CA-rules of the process, and the corresponding actions, in all the possible ways. As for states, we only consider those database instances where all artifact instances are not blocked; these correspond in fact to stable snapshots of $\mathcal{G}$. We then connect two such states provided that there is a sequence of (intermediate) states that lead from the first to the second one, and for which at least one artifact instance is blocked; these sequence corresponds in fact to a series of intermediate-steps evolving the system from a stable state to another stable state. Finally, we project away all the auxiliary relations introduced by the translation mechanism, obtaining a *filtered* version of $\Upsilon_{\mathcal{S}}$, which we denote as $\Upsilon_{\mathcal{S}}|_{\mathcal{G}}$. The intuition about the construction of these two transition systems is given in Figure 11. Notice that the intermediate micro-steps in the two transition systems can be safely abstracted away because: (i) thanks to the toggle-once principle, they do not contain any "internal" cycle; (ii) respecting the firing order imposed by $\mathcal{G}$, they all lead to reach the same next stable/unblocked state.

We can then establish the one-to-one correspondence between these two transition systems by applying subsequently results obtained from Lemmas A.1 to A.3 to prove the following theorem:

Theorem A.4 (Soundness and completeness)**.** *Given a GSM model $\mathcal{G}$ and its translation into a corresponding DCDS $\mathcal{S}$, the corresponding B-step transition system $\Upsilon_{\mathcal{G}}$ and filtered unblocked-state transition system $\Upsilon_{\mathcal{S}}|_{\mathcal{G}}$ are equivalent, i.e., $\Upsilon_{\mathcal{G}} \equiv \Upsilon_{\mathcal{S}}|_{\mathcal{G}}$.*