Verification of Artifact-Centric Systems: Decidability and Modeling Issues

Dmitry Solomakhin[1], Marco Montali[1],
Sergio Tessaris[1], and Riccardo De Masellis[2]

[1] Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy
{solomakhin,montali,tessaris}@inf.unibz.it
[2] Sapienza Università di Roma, Via Ariosto, 25, 00185 Rome, Italy
demasellis@dis.uniroma1.it

Abstract. Artifact-centric business processes have recently emerged as an approach in which processes are centred around the evolution of business entities, called *artifacts*, giving equal importance to control-flow and data. The recent Guard-State-Milestone (GSM) framework provides means for specifying business artifacts lifecycles in a declarative manner, using constructs that match how executive-level stakeholders think about their business. However, it turns out that formal verification of GSM is undecidable even for very simple propositional temporal properties. We attack this challenging problem by translating GSM into a well-studied formal framework. We exploit this translation to isolate an interesting class of "state-bounded" GSM models for which verification of sophisticated temporal properties is decidable. We then introduce some guidelines to turn an arbitrary GSM model into a state-bounded, verifiable model.

Keywords: artifact-centric systems, guard-stage-milestone, formal verification.

1 Introduction

In the last decade, a plethora of graphical notations (such as BPMN and EPCs) have been proposed to capture business processes. Independently from the specific notation at hand, formal verification has been generally considered as a fundamental tool in the process design phase, supporting the modeler in building correct and trustworthy process models [17]. Intuitively, formal verification amounts to check whether possible executions of the business process model satisfy some desired properties, like generic correctness criteria (such as deadlock freedom or executability of activities) or domain-dependent constraints. To enable formal verification and other forms of reasoning support, business process models are translated into an equivalent formal representation, which typically relies on variants of Petri nets [1], transition systems [2], or process algebras [19]. Properties are then formalized using temporal logics, using model checking techniques to actually carry out verification tasks [9].

A common drawback of classical process modeling approaches is being *activity-centric*: they mainly focus on the control-flow perspective, lacking the connection between the process and the data manipulated during its executions. This reflects also in the corresponding verification techniques, which often abstract away from the data component. This "data and process engineering divide" affects many contemporary process-aware information systems, increasing the risk of introducing redundancies and

S. Basu et al. (Eds.): ICSOC 2013, LNCS 8274, pp. 252–266, 2013.

potential errors in the development phase [13,8]. To tackle this problem, the artifact-centric paradigm has recently emerged as an approach in which processes are guided by the evolution of business data objects, called *artifacts* [18,10]. A key aspect of artifacts is coupling the representation of data of interest, called *information model*, with *lifecycle constraints*, which specify the acceptable evolutions of the data maintained by the information model. On the one hand, new modeling notations are being proposed to tackle artifact-centric processes. A notable example is the Guard-State-Milestone (GSM) graphical notation [11], which corresponds to way executive-level stakeholders conceptualize their processes [7]. On the other hand, the formal foundations of the artifact-centric paradigm are being investigated in order to capture the relationship between processes and data and to support formal verification [12,5,4,3]. Two important issues arise. First, verification formalisms must go beyond propositional temporal logics, and incorporate first-order formulae to express constraints about the evolution of data and to query the artifact information models. Second, verification tasks become undecidable in general.

In this work, we tackle the problem of *automated verification of GSM models*. First of all, we show that verifying GSM models is indeed a very challenging issue, being undecidable in general even for simple propositional reachability properties. We then provide a sound and complete encoding of GSM into Data-Centric Dynamic Systems (DCDSs), a recently developed formal framework for data- and artifact-centric processes [4]. This encoding enables the to transfer in the GSM context the decidability and complexity results recently established for DCDSs with bounded information models (*state-bounded DCDSs*). These are DCDSs where the number of tuples does not exceed a given maximum value. This does not mean that the system must contain an overall bounded amount of data: along a run, infinitely many data can be encountered and stored into the information model, provided that they do not accumulate in the same state. We lift this property in the context of GSM, and show that verification of state-bounded GSM models is decidable for a powerful temporal logic, namely a variant of first-order μ-calculus supporting a restricted form of quantification [14]. We then isolate an interesting class of GSM models for which state-boundedness is guaranteed, introducing guidelines that help to make GSM models state-bounded and, in turn, verifiable.

The rest of the paper is organized as follows. Section 2 gives an overview of GSM and provides a first undecidability result. Section 3 introduces DCDSs and presents the GSM-DCDS translation. Section 4 introduces "state-bounded" GSM models and provides key decidability results. Discussion and conclusion follow.

2 GSM Modeling of Artifact-Centric Systems

The foundational character of artifact-centric business processes is the combination of static properties; i.e., the data of interest, and dynamic properties of a business process, i.e., how it evolves. *Artifacts*, the key business entities of a given domain, are characterized by *(i)* an *information model* that captures business-relevant data, and *(ii)* a *lifecycle model* that specifies how the artifact progresses through the business. In this work, we focus on the Guard-Stage-Milestone (GSM) approach for artifact-centric modeling, recently proposed by IBM [11] and included by the Object Management Group (OMG) into the new standard for Case Management Model and Notation (CMMN) [22].

For the sake of simplicity here we provide a general overview of the GSM methodology and we refer an interested reader to [6] for more detailed and formal definitions.

GSM is a declarative modelling framework that has been designed with the goal of being executable and at the same time enough high-level to result intuitive to executive-level stakeholders. The GSM information model uses (possibly nested) attribute/value pairs to capture the domain of interest. The key elements of a lifecycle model are *stages*, *milestones* and *guards* (see Example 1). Stages are (hierarchical) clusters of activities (*tasks*) intended to update and extend the data of the information model. They are associated to milestones, business operational objectives to be achieved when the stage is under execution. Guards control the activation of stages and, like milestones, are described in terms of data-aware expressions, called *sentries*, involving events with associated data (called *payload*) and conditions over the artifact information model. Sentries have the form **on** e **if** $cond$, where e is an event and $cond$ is an (OCL-based, see [16]) condition over data. Both parts are optional, supporting pure event-based or condition-based sentries. Changes on the artifact state are performed by tasks, which represent atomic operations. They can be used to update the data of artifact instances (e.g., based on the payload of an incoming event), or to add/remove (nested) tuples. Crucially, tasks are used to manage artifacts life cycle. *Create-artifact-instance* tasks enable the creation of new artifact instances of a given type. Creation of artifacts is modelled as a two-way service call, where the returned result is used to create a new tuple for the artifact instance, to assign a new identifier to it, and to fill it with the result's payload. Analogously, tasks may remove existing artifact instances. In the following, we use *model* for the intensional level of a specific business process described in GSM, and *instance* to denote a GSM model with specific data for its information model.

The execution of a business process may involve several *instances* of artifact types described by a GSM model. At any instant, the state of an artifact instance (*snapshot*) is stored in its information model, and is fully characterised by: *(i)* values of attributes in the data model, *(ii)* status of its stages (open or closed) and *(iii)* status of its milestones (achieved or invalidated). 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: a *service call*, whose instances are populated by the data from information model and then sent to the environment and a *service call return*, whose instances represent the corresponding answer from the environment and are used to incorporate the obtained result back into the artifact information model. The environment can also send unsolicited (one-way) events, to trigger specific guards or milestones. Additionally, any change of a status attribute, such as opening a stage or achieving a milestone, triggers an internal event, which can be further used to govern the artifact lifecycle.

Example 1. Figure 1 shows a simple order management process modeled in GSM. The process centers around an *order* artifact, whose information model is characterized by a set of status attributes (tracking the status of stages and milestones), and by an extendible set of ordered *items*, each constituted by a code and a quantity. The order lifecycle contains three top-level atomic stages (rounded rectangles), respectively used to manage the manipulation of the order, its payment, and the delivery of a payment receipt. The order management stage contains a task (rectangle) to add items to the

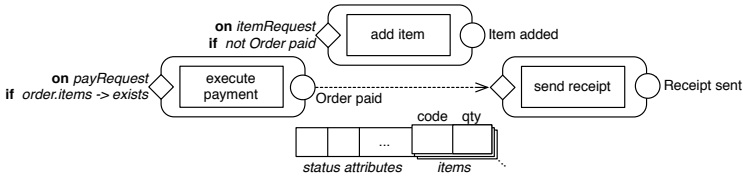

Fig. 1. GSM model of a simple order management process

order. It opens every time an $itemRequest$ event is received, provided that the order has not yet been paid. This is represented using a logical condition associated to a guard (diamond). The stage closes when the task is executed, by achieving an "item added" milestone (circle). A payment can be executed once a $payRequest$ event is issued, provided that the order contains at least one item (verified by the OCL condition $order.items \rightarrow exists$). As soon as the order is paid, and the corresponding milestone achieved, the receipt delivery stage is opened. This direct dependency is represented using a dashed arrow, which is a shortcut for the condition **on** $Order\ paid$, representing the internal event of achieving the "Order paid" milestone.

2.1 Operational Semantics of GSM

GSM is associated to three well-defined, equivalent execution semantics, which discipline the actual enactment of a GSM model [11]. Among these, the *GSM incremental semantics* is based on a form of Event-Condition-Action (ECA) rules, called *Prerequisite-Antecedent-Consequent* (*PAC*) rules, and is centered around the notion of *GSM Business steps* (*B-steps*). An artifact instance remains idle until it receives an incoming event from the environment. It is assumed that such events arrive in a sequence and get processed by artifact instances one at a time. A B-step then describes what happens to an *artifact snapshot* Σ when a single incoming event e is incorporated into it, i.e., how it evolves into a new snapshot Σ' (see Figure 5 in [11]). Σ' is constructed by building a sequence of pre-snapshots Σ_i, where Σ_1 results from incorporating e into Σ by updating its attributes according to the event payload (i.e., its carried data). Each consequent pre-snaphot Σ_i is obtained by applying one of the PAC rules to the previous pre-snapshot Σ_{i-1}. Each of such transitions is called a *micro-step*. During a micro-step some outgoing events directed to the environment may be generated. When no more PAC rules can be applied, the last pre-snapshot Σ' is returned, and the entire set of generated events is sent to the environment.

Each PAC rule is associated to one or more GSM constructs (e.g. stage, milestone) and has three components:

- **Prerequisite:** this component refers to the initial snapshot Σ and determines if a rule is *relevant* to the current B-step processing an incoming event e.
- **Antecedent:** this part refers to the current pre-snapshot Σ_i and determines whether the rule is eligible for execution, or $executable$, at the next micro-step.
- **Consequent:** this part describes the effect of firing a rule, which can be nondeterministically chosen in order to obtain the next pre-snapshot Σ_{i+1}.

Due to nondeterminism in the choice of the next firing rule, different orderings among the PAC rules can exist, leading to non-intuitive outcomes. This is avoided in the GSM

operational semantics by using an approach reminiscent of stratification in logic programming. In particular, the approach *(i)* exploits implicit dependencies between the (structure of) PAC rules to fix an ordering on their execution, and *(ii)* applies the rules according to such ordering [11]. To guarantee B-step executability, avoiding situations in which the execution indefinitely loops without reaching a stable state, the GSM incremental semantics implements a so-called *toggle-once* principle. This guarantees that a sequence of micro-steps, triggered by an incoming event, is always finite, by ensuring that each status attribute can change its value at most once during a B-step. This requirement is implemented by an additional condition in the prerequisite part of each PAC rule, which prevents it from firing twice.

The evolution of a GSM system composed by several artifacts can be described by defining the initial state (initial snapshot of all artifact instances) and the sequence of event instances generated by the environment, each of which triggers a particular B-step, producing a sequence of system snapshots. This perspective intuitively leads to the representation of a GSM model as an infinite-state transition system, depicting all possible sequences of snapshots supported by the model. The initial configuration of the information model represents the initial state of this transition system, and the incremental semantics provides the actual transition relation. The source of infinity relies in the payload of incoming events, used to populate the information model of artifacts with fresh values (taken from an infinite/arbitrary domain). Since such events are not under the control of the GSM model, the system must be prepared to process such events in every possible order, and with every acceptable configuration for the values carried in the payload. The analogy to transition systems opens the possibility of using a formal language, e.g., a (first-order variant of) temporal logic, to verify whether the GSM system satisfies certain desired properties and requirements. For example, one could test generic correctness properties, such as checking whether each milestone can be achieved (and each stage will be opened) in at least one of the possible systems' execution, or that whenever a stage is opened, it will be always possible to eventually achieve one of its milestones. Furthermore, the modeler could also be interested in verifying domain-specific properties, such as checking whether for the GSM model in Figure 1 it is possible to obtain a receipt before the payment is processed.

2.2 Undecidability in GSM

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

Proof. We represent a Turing machine as a GSM artifact, formulating the halting problem as a verification problem over such artifact. 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, without loss of generality, that δ consists of k right-shift transitions

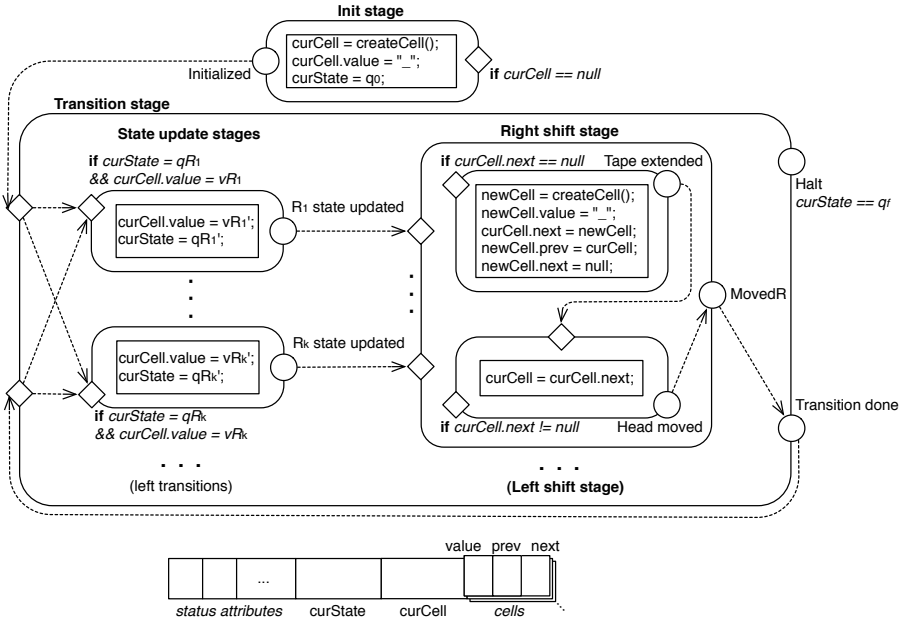

Fig. 2. GSM model of a Turing machine

$R_1, \ldots, R_k$ (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 the 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 and *(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 (double) linked list, which initially contains a single, blank cell, and which is dynamically extended on demand. 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 2, 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*, used to initialize the tape, and *Transition*, encoding 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* opens 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 ensure that this condition is evaluated as soon as the parent stage opens, hence, if the condition is true, the *state update stage*

is immediately executed. When the *state update stage* closes, the achievement of the corresponding milestone triggers one of the guards of the *right shift stage* that handles the head shift. *Right shift stage* 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 performs 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? Since $\mathcal{M}$ is deterministic, the B-steps of the corresponding GSM model give raise to 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. □

3 Translation into Data-Centric Dynamic Systems

Despite having a formally specified operational semantics for GSM models [11], the verification of different properties of such models (e.g. existence of complete execution, safety properties) is still an open problem. A promising framework for the formalization and verification of artifact systems is the one of data-centric dynamic systems (DCDS), recently presented in [4]. Translating a GSM model into a corresponding DCDS enables the application of the decidability results and verification techniques discussed in [4] to the concrete case of GSM. Additionally, such translation will allow to benefit from the results of the ongoing effort towards execution support for DCDS [20]. First we briefly introduce DCDS and then we present a translation that faithfully rewrites a GSM model into a corresponding formal representation in terms of DCDSs.

Formally, a DCDS is a pair $\mathcal{S} = \langle \mathcal{D}, \mathcal{P} \rangle$, where $\mathcal{D}$ is a data layer and $\mathcal{P}$ is a process layer over $\mathcal{D}$. The former maintains all the relevant data in the form of a relational database together with its integrity constraints. In the artifact-centric context, the database is the union of all artifacts information models. The process layer modifies the data maintained by $\mathcal{D}$, and it is defined as a tuple $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \varrho \rangle$ where:

- $\mathcal{F}$ is a finite set of functions representing interfaces to external services, used to import new, fresh data into the system.
- $\mathcal{A}$ is a set of actions of the form $\alpha(p_1, ..., p_n) : \{e_1, ..., e_m\}$, where $p_1, ..., p_n$ are input parameters of an action and e_i are effects of an action. Each effect specification defines how a portion of the next database instance is constructed starting from the current one and has the form $e_i = q_i^+ \wedge Q_i^- \rightsquigarrow E_i$ where:
  - q_i^+ is a union of conjunctive queries (UCQ) over $\mathcal{D}$, used to instantiate the effect with values extracted from the current database.
  - Q_i^- is an arbitrary FO formula that filters away some tuples obtained by q_i^+.

- E_i is a set of effects, specified in terms of facts over $\mathcal{D}$ that will be asserted in the next state; these facts can contain variables of Q (which are then replaced with actual values extracted from the current database) and also service calls, which are resolved by calling the service with actual input parameters and substituting them with the obtained result.[1].

– ϱ is a declarative process specified in terms of a finite set of Condition-Action (CA) rules that determine, at any moment, which actions are executable. Technically, each CA rule has the form $Q \mapsto \alpha$, where α is an action and Q is a FO query over $\mathcal{D}$. Whenever Q has a positive answer over the current database, then α becomes executable, with actual values for its parameters given by the answer to Q.

Example 2. Consider a fragment of an order management process. Once pending, an order can be moved to the ready state by executing a *prepare* action, which incorporates into the system the destination address of the customer associated to the order. A DCDS could encode the executability of the *prepare* action by means of the following CA rule: $order(id, cust) \wedge pending(id) \mapsto$ PREPARE(id). The rule states that whenever an order identified by id and owned by customer $cust$ is pending, it is possible to apply action *prepare* on it. The action can in turn be defined as:

$$\begin{aligned}\text{PREPARE}(id) : \{ \quad order(id, cust) \wedge pending(id) &\rightsquigarrow \{ready(id), dest(id, addr(cust))\} \\ order(x, y) &\rightsquigarrow \{order(x, y)\} \\ order(x, y) \wedge pending(x) \wedge x \neq id &\rightsquigarrow \{pending(x)\} \\ order(x, y) \wedge ready(x) &\rightsquigarrow \{ready(x)\} \quad \}\end{aligned}$$

The first effect states that the order id becomes ready, and its destination is incorporated by calling a service $addr$, which mimics the interaction with the customer. The other effects are used to determine which information is kept unaffected in the next state: all orders remain orders, all ready orders remain ready, and all pending orders remain pending, except the one identified by id.

The execution semantics of a DCDS $\mathcal{S}$ is defined by a possibly infinite-state transition system $\Upsilon_{\mathcal{S}}$, where states are instances of the database schema in $\mathcal{D}$ and each transition corresponds to the application of an executable action in $\mathcal{P}$. Similarly to GSM, where the source of infinity comes from the fact that incoming events carry an arbitrary payload, in DCDSs the source of infinity relies in the service calls, which can inject arbitrary fresh values into the system. Despite the resulting undecidability of arbitrary DCDSs, an interesting class of *state-bounded* DCDSs has been recently identified [4], for which decidability of verification holds for a sophisticated (first-order) temporal logic called $\mu\mathcal{L}_P$. Intuitively, state boundedness requires the existence of an overall bound that limits, at every point in time, the size of the database instance of $\mathcal{S}$ (without posing any restriction on which values can appear in the database). Equivalently, the size of each state contained in $\Upsilon_{\mathcal{S}}$ cannot exceed the pre-established bound. Hence, in the following we will indifferently talk about state-bounded DCDSs or state-bounded transition systems.

Theorem 2 ([4]). *Verification of $\mu\mathcal{L}_P$ properties over state-bounded DCDS is decidable, and can be reduced to finite-state model checking of propositional μ-calculus.*

$\mu\mathcal{L}_P$ is a first-order variant of μ-calculus, a rich branching-time temporal logic that subsumes all well-known temporal logics such as PDL, CTL, LTL and CTL* [14]. $\mu\mathcal{L}_P$

[1] In [4], two semantics for services are introduced: deterministic and nondeterministic. Here we always assume nondeterministic services, which is in line with GSM.

employs first-order formulae to query data maintained by the DCDS data layer, and supports a controlled form of first-order quantification across states (within and across runs).

Example 3. $\mu\mathcal{L}_P$ can express two variants of a correctness requirement for GSM:

- it is always true that, whenever an artifact id is present in the information model, the corresponding artifact will be destroyed (i.e., the id will disappear) *or* reach a state where all its stages are closed;
- it is always true that, whenever an artifact id is present in the information model, the corresponding artifact will persist *until* a state is reached where all its stages are closed.

3.1 Translating GSM into DCDS

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[2].

As introduced in Section 2.1, the execution of a GSM instance is described by a sequence of B-steps. 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 requirements described in Section 2.1 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 introducing 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

[2] For the sake of space, we give a general description of the translation and illustrate the technical development by the example in Figure 4. For a full technical specification of the translation, we refer the interested reader to a technical report [21].

$$R_{exec}(id_R, \overline{x}) \wedge x_k = 0 \wedge exec(k) \wedge R_{block}(id_R, true) \mapsto \quad (1)$$

$$a^k_{exec}(id_R, \overline{a}', \overline{x}) : \{ \quad (2)$$

$$R_{att}(id_R, \overline{a}, \overline{s}, \overline{m}) \wedge R^{S_j}_{chg}(id_R, true) \rightsquigarrow \{R_{att}(id_R, \overline{a}, \overline{s}, \overline{m})[m_j/false]\} \quad (3)$$

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

$$R^M_{exec}(id_R, \overline{x}) \wedge x_k = 0 \rightsquigarrow \{R^M_{exec}(id_R, \overline{x})[x_k/1]\} \quad (5)$$

$$[\mathsf{CopyMessagePools}], [\mathsf{CopyRest}] \quad \} \quad (6)$$

Fig. 3. CA-rule encoding a milestone invalidation upon stage activation

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 execution, 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 4. An example of a translation of a GSM PAC-rule (indexed by k) is presented in Figure 3. 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 the 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 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 pro-

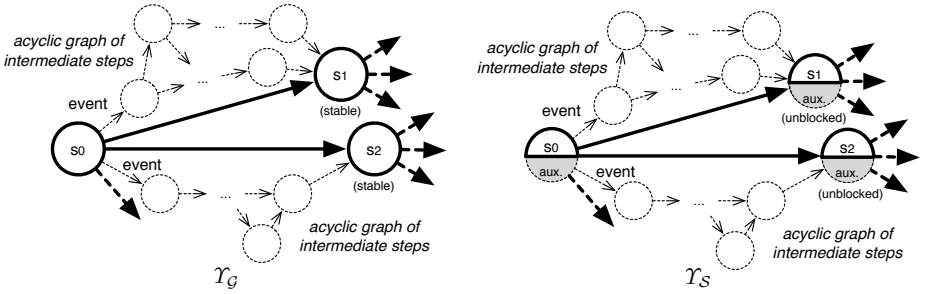

Fig. 4. Construction of the B-step transition system $\Upsilon_{\mathcal{G}}$ and unblocked-state transition system $\Upsilon_{\mathcal{S}}$ for a GSM model $\mathcal{G}$ with initial snapshot s_0 and the corresponding DCDS $\mathcal{S}$

cess, 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 4. 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 (refer to [21] for complete proof):

Theorem 3. *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., $\Upsilon_{\mathcal{G}} \equiv \Upsilon_{\mathcal{S}}|_{\mathcal{G}}$.*

4 State-Bounded GSM Models

We now take advantage of the key decidability result given in Theorem 2, 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 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 system 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 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 3, 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. □

From the combination of Theorems 2 and 3 and Lemma 1, we directly obtain:

Theorem 4. *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, state-boundedness is a "semantic" condition, which is undecidable to check [4]. 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 Fig. 1), are not state-bounded, and thus we provide some modeling strategies to make any GSM model state-bounded.

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 Example 1) 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.*

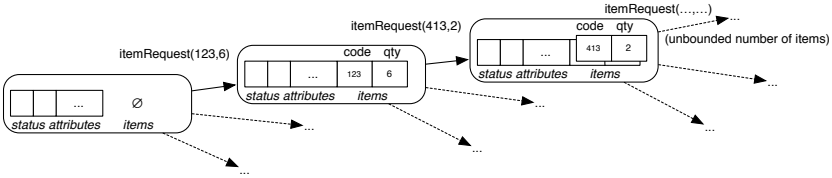

Fig. 5. Unbounded execution of the GSM model in Fig. 1

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 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 4 to get the result. □

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. 1, 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. 5. 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 [21].

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}).

5 Discussion and Related Work

In this work we provided the foundations for the formal verification of the GSM artifact-centric paradigm. So far, only few works have investigated verification of GSM models. The closest approach to ours is [6], where state-boundedness is also used as a key property towards decidability. The main difference between the two approaches is that decidability of state-bounded GSM models is proven for temporal logics of incomparable expressive power. In addition to [6], in this work we also study modeling strategies to make an arbitrary GSM model state-bounded, while they assume that the input model is guaranteed to be state-bounded. Hence, our strategies could be instrumental to [6] as well. In [15] another promising technique for the formal verification of GSM models is presented. However, the current implementation cannot be applied to general GSM models, because of assumptions over the data types and the fact that only one instance per artifact type is supported. Furthermore, a propositional branching-time logic is used for verification, restricting to the status attributes of the artifacts. The results presented in our paper can be used to generalize this approach towards more complex models (such as instance-bounded GSM models) and more expressive logics, given, e.g., the fact that "one-instance artifacts" fall inside the decidable cases we discussed in this paper.

It is worth noting that all the presented decidability results are actually even stronger: they state that verification can be reduced to standard model checking of propositional μ-calculus over finite-state transition systems (thanks to the abstraction techniques studied in [4]). This opens the possibility of actually implementing the discussed techniques, by relying on state-of-the-art model checkers. We also inherit from [4] the complexity boundaries: they state that verification is EXPTIME in the size of the GSM information model which, in the case of instance-bounded GSM models, means in turn EXPTIME in the maximum number of artifact instances that can coexist in the same state.

References

1. van der Aalst, W.M.P., Stahl, C.: Modeling Business Processes - A Petri Net-Oriented Approach. Springer (2011)
2. Armando, A., Ponta, S.E.: Model checking of security-sensitive business processes. In: Degano, P., Guttman, J.D. (eds.) FAST 2009. LNCS, vol. 5983, pp. 66–80. Springer, Heidelberg (2010)
3. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., De Masellis, R., Felli, P., Montali, M.: Description logic knowledge and action bases. Journal of Artificial Intelligence Research, 651–686 (2013)
4. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., Deutsch, A., Montali, M.: Verification of relational data-centric dynamic systems with external services. In: Proc. of PODS, pp. 163–174. ACM Press (2013)
5. Belardinelli, F., Lomuscio, A., Patrizi, F.: An abstraction technique for the verification of artifact-centric systems. In: Proc. of KR. AAAI Press (2012)
6. Belardinelli, F., Lomuscio, A., Patrizi, F.: Verification of gsm-based artifact-centric systems through finite abstraction. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) ICSOC 2012. LNCS, vol. 7636, pp. 17–31. Springer, Heidelberg (2012)
7. Bhattacharya, K., Caswell, N.S., Kumaran, S., Nigam, A., Wu, F.Y.: Artifact-centered operational modeling: Lessons from customer engagements. IBM Systems Journal 46(4) (2007)
8. Calvanese, D., De Giacomo, G., Montali, M.: Foundations of data-aware process analysis: A database theory perspective. In: Proc. of PODS, pp. 1–12. ACM Press (2013)
9. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. The MIT Press (1999)
10. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. 32(3) (2009)
11. Damaggio, E., Hull, R., Vaculin, R.: On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. Information Systems (2012)
12. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic verification of data-centric business processes. In: Proc. of ICDT, pp. 252–267. ACM Press (2009)
13. Dumas, M.: On the convergence of data and process engineering. In: Eder, J., Bielikova, M., Tjoa, A.M. (eds.) ADBIS 2011. LNCS, vol. 6909, pp. 19–26. Springer, Heidelberg (2011)
14. Emerson, E.A.: Model checking and the mu-calculus. In: Descriptive Complexity and Finite Models (1996)
15. Gonzalez, P., Griesmayer, A., Lomuscio, A.: Verifying gsm-based business artifacts. In: Proc. of ICWS, pp. 25–32. IEEE (2012)
16. Group, T.O.M.: Object constraint language, version 2.0. Tech. Rep. formal/06-05-01, The Object Management Group (May 2006), `http://www.omg.org/spec/OCL/2.0/`
17. Morimoto, S.: A survey of formal verification for business process modeling. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part II. LNCS, vol. 5102, pp. 514–522. Springer, Heidelberg (2008)
18. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3) (2003)
19. Puhlmann, F., Weske, M.: Using the *pi*-calculus for formalizing workflow patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 153–168. Springer, Heidelberg (2005)
20. Russo, A., Mecella, M., Montali, M., Patrizi, F.: Towards a reference implementation for data centric dynamic systems. In: Proc. of BPM Workshops (2013)
21. Solomakhin, D., Montali, M., Tessaris, S.: Formalizing guard-stage-milestone meta-models as data-centric dynamic systems. Tech. Rep. KRDB12-4, KRDB Research Centre, Faculty of Computer Science, Free University of Bozen-Bolzano (2012)
22. The Object Management Group: Case Management Model and Notation (CMMN), Beta 1 (January 2013), `http://www.omg.org/spec/CMMN/1.0/Beta1/`