Foundations of Relational Artifacts Verification

Babak Bagheri Hariri[1], Diego Calvanese[1],
Giuseppe De Giacomo[2], Riccardo De Masellis[2], and Paolo Felli[2]

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

Abstract. Artifacts are entities characterized by *data* of interest (constituting the state of the artifact) in a given business application, and a *lifecycle*, which constrains the artifact's possible evolutions. In this paper we study *relational artifacts*, where data are represented by a full fledged relational database, and the lifecycle is described by a temporal/dynamic formula expressed in μ-calculus. We then consider business processes, modeled as a set of condition/action rules, in which the execution of actions (aka tasks, or atomic services) results in new artifact states. We study conformance of such processes wrt the artifact lifecycle as well as verification of temporal/dynamic properties expressed in μ-calculus. Notice that such systems are infinite-state in general, hence undecidable. However, inspired by recent literature on database dependencies developed for data exchange, we present a natural restriction that makes such systems finite-state, and the above problems decidable.

1 Introduction

The artifact-centric approach to design and development of business processes is emerging as an interesting alternative to the traditional methods that focus mainly on processes [19,14,10,1,2,13]. This approach focuses simultaneously on data and processes. Data correspond to key business-relevant entities, which are seen as evolving over time following a so-called *lifecycle*. Processes compose into a workflow atomic tasks or services that are available and of interest. The artifact-centric approach provides a simple and robust structure for business process development, which has been advocated to enhance efficiency, especially in dealing with business transformations [6,7].

The interest in both data and processes as first-class citizens in artifact-centric systems deeply challenges the research community in verification. Indeed, on the one hand, such systems deal with full-fledged processes, which require analysis in terms of sophisticated temporal properties [4]. On the other hand, the presence of data makes the whole system become infinite-state in general, and hence the usual verification techniques based on model checking of finite-state systems cannot be applied [4,21].

In this paper, we study the foundations of artifact-centric systems that use relational databases for their data component. Specifically, we consider several artifacts (fixed in advance) forming a so called *relational artifact system*, each constituted by a *relational database* evolving over time. To characterize such an evolution, we rely on a very rich notion of lifecycle, directly based on stating dynamic properties in terms of *intra-artifact* and *inter-artifact dynamic constraints*. (This generalized form of lifecycle has emerged in the research done within the project ACSI, see `http://www.acsi-project.eu/`.) We express such constraints, and other dynamic properties of interest, in a suitable variant of μ-calculus, one of the most expressive temporal logics used in verification [18,11].

We consider *processes* over artifacts constituted by a set of *actions* (aka atomic tasks, atomic services) and a set of *condition-action rules*, which specify when such actions can be executed. The action specification is possibly the most characterizing part of our framework. Following [9], actions are specified in terms of preconditions and postconditions on artifacts' databases. Such a specification is strongly influenced by the notion of *mappings* in the recent literature on data exchange and data integration [12,17]. In a nutshell, our actions specification considers the current state of the database, and the one obtained by executing an action as two databases related through a set of mappings. In the literature, mappings typically establish correspondences between conjunctive queries, also called tuple-generating dependencies (TGDs) in the database jargon [3]. However here, differently from [9], we do use negation and more generally full first-order queries in defining the preconditions of actions. Technically speaking, this choice requires us to abandon the theory of conjunctive queries and homomorphisms at the base of the results in [9,12,17].

We are interested in two main reasoning tasks. The first one is *conformance of a process to an artifact system*, which consists in checking whether a given process generates the correct lifecycle for the various artifacts and, more generally, whether it satisfies all intra-artifact and inter-artifact constraints. The second reasoning task is *process verification*, that is checking whether a process (over an artifact system) verifies general dynamic properties of interest. Both these reasoning tasks in principle can be based on model checking, though, in our setting, one has to deal with potentially infinite states.

We show that both reasoning tasks are undecidable even for very simple artifact systems and processes. We then introduce a very interesting class of processes for which decidability is granted. We call such processes *weakly acyclic*, since they satisfy a condition analogous to weak acyclicity of a set of mappings in data exchange [12]. Under such a restriction, we are guaranteed that the number of new objects introduced by the execution of actions is finite, and hence, the whole process is finite-state.

2 Relational Artifacts Systems

In this section, we start the description of our framework by introducing relational artifact systems. In the following, we assume the reader to be familiar with standard relational databases, and their connection with first-order logic (FOL). In particular, queries are seen as (possibly open) FOL formulas. Also, we consider as special FOL queries conjunctive queries (CQs), i.e., formulas formed only by conjunctions and existential quantifications, and their unions (UCQs) [3].

A *relational artifact systems* (RAS) is constituted by a set of artifacts, each formed by a relational database evolving over time under restrictions imposed by certain dynamic constraints. We deal with two types of constraints: *intra-artifact dynamic constraints*, which involve each artifact in isolation, and *inter-artifact dynamic constraints*, which take into account relations between artifacts. In this section we introduce such systems.

Relational artifact. A relational artifact is a relational database evolving over time. Hence, it is characterized by the usual notions of *database schema*, giving the structure of the database, and *database instance*, detailing the actual data contained in it, and it is furthermore augmented by a set of *intra-artifact dynamic constrains.* These are temporal constraints expressed in the temporal logic $\mu\mathcal{L}$ introduced later, which allows us to express various constraints over the database: we can assert the usual ones, such as inclusion dependencies, which now become safety temporal constrains, and also what is typically called the *artifact lifecycle*, namely, dynamic constrains on the sequencing of database configurations. More formally, a *relational artifact* is a tuple $A = \langle \boldsymbol{R}, I_0, \Phi \rangle$

- $\boldsymbol{R} = \{R_1, \ldots, R_n\}$ is a database schema, i.e., a set of relation schemas;
- I_0 is a database instance, compliant with the schema $\boldsymbol{R}$, that represents the initial state of the artifact;
- Φ is a $\mu\mathcal{L}$ formula over $\boldsymbol{R}$ constituted by the conjunction of all intra-artifact dynamic constraints of A.

Notice that if we project the dynamic formula Φ over the initial artifact instance I_0, we may get (depending on the structure of Φ) static, i.e., local, constraints on I_0. From now on, we assume to deal with *well formed artifacts*, namely, artifacts whose initial instance satisfies such local constraints.

Relational artifact system. A relational artifact system is composed of several relational artifacts in execution at the same time, each consisting of a database and a set of intra-artifact dynamic constraints. The dynamic interaction between the artifacts is regulated through additional constraints, also expressed in $\mu\mathcal{L}$, which we call *inter-artifact-dynamic constraints.*

In this paper, we make the assumption that artifacts cannot be created or destroyed during the evolution of the system. Under such an assumption we get quite interesting undecidability and decidability results. We are indeed very interested in dropping these limitations in future works, starting from the results presented here. For this reason we start with a finite set of artifacts, and over the whole evolution of the system these will remain the only ones of interest. If an artifact has a terminating lifecycle it becomes dormant, but it will persist in the system.

Formally, an *artifact system* is a pair $\mathcal{A} = \langle \{A_1, \ldots, A_n\}, \Phi_{inter} \rangle$, where $\{A_1, \ldots, A_n\}$ is the finite set of artifacts of the system (each with its own database and intra-artifact dynamic constraints expressed in $\mu\mathcal{L}$), and Φ_{inter}

is a $\mu\mathcal{L}$ formula expressing the conjunction of inter-artifact dynamic constraints. To distinguish relations of various artifacts in $\mathcal{A}$, we use the usual *dot notation* of object-orientation, hence, a relation R_j of artifact A_i of $\mathcal{A}$ is denoted by $A_i.R_j$. When clear from the context, we drop the artifact A_i and we use R_j for the relation. We denote by $\mathcal{I}_0$ the disjoint union of all initial instances of the artifacts in $\mathcal{A}$, i.e., $\mathcal{I}_0 = \bigcup_{i=1,\ldots,n} I_{0,i}$. More generally, $\mathcal{I}$ represents the instance obtained by the (disjoint) union of the current instances of each artifact in $\mathcal{A}$.

Given a database instance I, we denote by $\mathcal{C}_I$ the active domain of I, i.e., the set of individuals (typically constants) appearing in I. Hence, the active domain of $\mathcal{I}_0$ is $\mathcal{C}_{\mathcal{I}_0}$, which is made up by all constants appearing in the initial instances of the various artifacts in $\mathcal{A}$.

Notice that, while artifact systems evolve over time, they do not include a predefined mechanism for progression. Progression is due to the execution of actions, tasks, or services over the system, according to a given process that we will introduce later on. Here it is sufficient to assume that a progression mechanism exists, and its execution results in moving from the initial state, given by the instance $\mathcal{I}_0$, to the next one, and so on.

In this way we build a *transition system* [4] $\mathfrak{A}$, whose states represent possible system instances, and each transition an atomic step in the progression mechanism (whatever it is). In principle, we can model-check such a transition system to verify dynamic properties [4], that is exactly what we are going to do next. However, one has to consider that, in general, $\mathfrak{A}$ is infinite, hence the classical results on model checking [4,11], which are developed for finite transition systems, do not apply. The main goal of this paper is to find interesting conditions under which such a transition system is finite.

Example 1. We model the process of purchasing items within a company. In particular, when a company's employee, who assumes the role of a *requester*, wants to purchase some items, he has to turn to a *buyer*, also internal to the company, who is responsible for purchasing such items from external *suppliers*. In our scenario, we have five actors: two requesters (Bob and Alice), a buyer (Trudy) and two suppliers (SupplierA and SupplierB). The whole purchasing process works as follows: in a first phase, the requester has to fill a so-called *requisition order* with some *line items* chosen from a catalogue. In our simple example the catalogue contains only a monitor, a mouse, and a keyboard. Once the requester has completed this process, he sends the order to the buyer, who extracts the line items from it, and purchases each of them separately. In particular, the buyer groups together into a *procurement order* line items (belonging to a requisition order) that will be purchased from a particular supplier. As a result of this phase, we get different procurement orders, each containing line items that the buyer requests from a single supplier. Then the supplier ships back to the buyer the items included in the procurement order he received, and finally, the items are delivered to the original requester. Of course, we can have many orders processed simultaneously in the system, although we will impose some restrictions.

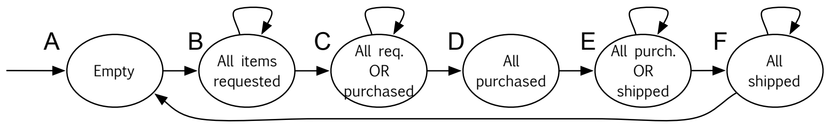

Fig. 1. Informal representation of *dynamic* intra-artifact constraints

In this example, we consider the relational artifact system $\mathcal{A} = \langle\{\mathsf{ReqOrders}, \mathsf{ProcOrders}\}, \Phi_{inter}\rangle$ containing two relational artifacts, holding all relevant data about requisition orders and procurement orders in the system.

$\mathsf{ReqOrders} = \langle \boldsymbol{R}_{RO}, I_{0,RO}, \Phi_{RO}\rangle$, where

– $\boldsymbol{R}_{RO} = \{\, \mathsf{RO}(\mathit{RoCode}, \mathit{ReqName}, \mathit{BuName}),\ \mathsf{ROItem}(\mathit{RoCode}, \mathit{ProdName}, \mathit{Status}),$
$\mathsf{Requester}(\mathit{ReqName}),\ \mathsf{LineItem}(\mathit{ProdName}, \mathit{Price}),$
$\mathsf{Buyer}(\mathit{BuName}),\ \mathsf{Status}(\mathit{StatusName})\ \}$

A requisition order is meant to hold the data associated to every *pending* requisition order: indeed, as soon as the items are delivered to the corresponding requester, each information associated to them is removed from the system. Relation $\mathsf{RO}(\mathit{RoCode}, \mathit{ReqName}, \mathit{BuName})$ holds basic information associated to a single order, i.e., order's code and both requester's and buyer's names. The requested items are kept in the relation $\mathsf{ROItem}(\mathit{RoCode}, \mathit{ProdName}, \mathit{Status})$, whose attribute *Status* keeps track of the status of each line item included in the order (it can be either requested, purchased or shipped). Relations $\mathsf{Requester}(\mathit{ReqName})$, $\mathsf{LineItem}(\mathit{ProdName}, \mathit{Price})$, $\mathsf{Buyer}(\mathit{BuName})$, and $\mathsf{Status}(\mathit{StatusName})$ are included in the schema for technical convenience; in particular, the relation Status is needed in order to easily bind values of the attribute *Status* of each line item in an order.

– $I_{0,RO} = \{\, \mathsf{Requester(Bob)},\ \mathsf{Requester(Alice)},\ \mathsf{Buyer(Trudy)},$
$\mathsf{Status(requested)}, \mathsf{Status(purchased)}, \mathsf{Status(shipped)},$
$\mathsf{LineItem(keyboard, 20)}, \mathsf{LineItem(mouse, 10)}, \mathsf{LineItem(monitor, 200)}\ \}$

According to the previous description of this example scenario, in the initial instance we only have data concerning existing requesters, buyers, suppliers, and the catalogue, featuring three line items. There are no pending orders.

– As for the intra-artifact constraints Φ_{RO}, here we only give an intuition of what will be presented formally later. We want to trace the status of an ordered line item through the attribute $\mathsf{ROItem}.\mathit{Status}$, so we express constraints on the evolutions of all orders in the system by relying on this attribute, as informally depicted in Figure 1. Intuitively, at the beginning, we do not have any order placed by requesters: in the current situation (henceforth called *phase*) the relations RO and ROItem are empty $[A]$. As orders are placed, and new requisition orders are created, we will have a phase in which all currently pending orders have status requested $[B]$, and such condition will hold until, *eventually*, some item in such a status will be purchased by the buyer by creating procurement orders to send to suppliers, hence changing status to purchased. At this point, we will be in a phase such that all items belonging to existing orders are either requested or purchased $[C]$ and finally, at some point, all items will be purchased $[D]$. Having all procurement orders sent to suppliers, some of them will be shipped back, i.e., setting the status of corresponding items to $\mathsf{shipped}$ $[E]$, and at the end, all of them will be shipped back to the buyer $[F]$. Finally, as the items are delivered to the requester, they will be removed from the system and the initial condition will be eventually met again. Notice that we are imposing some restrictions over the evolution of the artifact: for instance, we won't allow for creating new requisition orders (for ordering new line items) as soon as all the existing ones have been purchased $[D]$. Notice also that we will need a way to force the system to eventually exit self-loops. Moreover, in addition to such *dynamic* constraints, we also have some *static* ones, such as inclusion dependencies.

$\mathsf{ProcOrders} = \langle \boldsymbol{R}_{PO}, I_{0,PO}, \Phi_{PO}\rangle$, where

– $\boldsymbol{R}_{PO} = \{\,\mathsf{PO}(PoCode, RoCode, SupName),\ \mathsf{POItem}(PoCode, RoCode, ProdName),$ $\mathsf{Supplier}(SupName),\ \mathsf{LineItem}(ProdName, Price)\}$.

Recall that all line items assigned to the same procurement order must belong to the same requisition order. Hence, similarly to requisition orders, a procurement order's schema includes a relation $\mathsf{PO}(PoCode, RoCode, SupName)$ holding its code, the code of the corresponding requisition order, and the name of the chosen supplier. Relation $\mathsf{POItem}(PoCode, RoCode, ProdName)$ holds instead the set of line items in each procurement order. Attribute $RoCode$ is replicated in this relation for convenience. $\mathsf{Supplier}(SupName)$ keeps the set of existing suppliers whereas $\mathsf{LineItem}(ProdName, Price)$ is the same as the one in the requisition order artifact.

– $I_{0,PO} = \{\,\mathsf{LineItem(keyboard, 20), LineItem(mouse, 10), LineItem(monitor, 200),}$ $\mathsf{Supplier(SupplierA), Supplier(SupplierB)}\}$.

– In this example we don't want to constrain the dynamic evolution of the artifact, so the only intra-artifact constraints we will consider are those needed for consistency. ■

3 Dynamic Constraints Formalism

We turn to the dynamic constraints formalism, used both to specify intra and inter dynamic constraints of artifact systems (including artifact lifecycles), and to specify dynamic properties of processes running over such systems. Several choices are possible: here we focus on a variant of μ-calculus [11], one of the most powerful temporal logics, subsuming both linear time logics, such as LTL and PSL, and branching time logics such as CTL and CTL* [4]. In particular, we introduce a variant of μ-calculus, called $\mu\mathcal{L}$, whose syntax is as follows:

$$\begin{aligned}\Phi ::= {}& Q \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \exists x \in \mathcal{C}_{\mathcal{I}_0}.\Phi \mid \forall x \in \mathcal{C}_{\mathcal{I}_0}.\Phi \mid \\ & \Box\Phi \mid \Diamond\Phi \mid \mu Z.\Phi \mid \nu Z.\Phi \mid Z,\end{aligned}$$

where Q is a possibly open FOL formula over the relations in the artifacts of $\mathcal{A}$, and Z is a second-order predicate variable. The symbols μ and ν can be

$$
\begin{array}{ll}
(\neg\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \Sigma_{\mathfrak{A}} - (\Phi)^{\mathfrak{A}}_{\mathcal{V}} \\
(\Phi_1 \wedge \Phi_2)^{\mathfrak{A}}_{\mathcal{V}} & = (\Phi_1)^{\mathfrak{A}}_{\mathcal{V}} \cap (\Phi_2)^{\mathfrak{A}}_{\mathcal{V}} \\
(\Phi_1 \vee \Phi_2)^{\mathfrak{A}}_{\mathcal{V}} & = (\Phi_1)^{\mathfrak{A}}_{\mathcal{V}} \cup (\Phi_2)^{\mathfrak{A}}_{\mathcal{V}} \\
(\exists x \in \mathcal{C}_{\mathcal{I}_0}.\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \bigcup\{(\Phi)^{\mathfrak{A}}_{\mathcal{V}[x/c]} \mid c \in \mathcal{C}_{\mathcal{I}_0}\} \\
(\forall x \in \mathcal{C}_{\mathcal{I}_0}.\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \bigcap\{(\Phi)^{\mathfrak{A}}_{\mathcal{V}[x/c]} \mid c \in \mathcal{C}_{\mathcal{I}_0}\}
\end{array}
\qquad
\begin{array}{ll}
(Z)^{\mathfrak{A}}_{\mathcal{V}} & = Z\mathcal{V} \subseteq \Sigma_{\mathfrak{A}} \\
(Q)^{\mathfrak{A}}_{\mathcal{V}} & = \{\mathcal{I} \in \Sigma_{\mathfrak{A}} \mid \mathit{ans}(Q\mathcal{V}, \mathcal{I})\} \\
(\Diamond\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \{\mathcal{I} \in \Sigma_{\mathfrak{A}} \mid \exists \mathcal{I}'.\ \mathcal{I} \Rightarrow_{\mathfrak{A}} \mathcal{I}' \text{ and } \mathcal{I}' \in (\Phi)^{\mathfrak{A}}_{\mathcal{V}}\} \\
(\Box\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \{\mathcal{I} \in \Sigma_{\mathfrak{A}} \mid \forall \mathcal{I}'.\ \mathcal{I} \Rightarrow_{\mathfrak{A}} \mathcal{I}' \text{ implies } \mathcal{I}' \in (\Phi)^{\mathfrak{A}}_{\mathcal{V}}\} \\
(\mu Z.\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \bigcap\{\mathcal{E} \subseteq \Sigma_{\mathfrak{A}} \mid (\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z/\mathcal{E}]} \subseteq \mathcal{E}\} \\
(\nu Z.\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \bigcup\{\mathcal{E} \subseteq \Sigma_{\mathfrak{A}} \mid \mathcal{E} \subseteq (\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z/\mathcal{E}]}\}
\end{array}
$$

Fig. 2. Semantics of $\mu\mathcal{L}$ formulas

considered as quantifiers, and we make use of the notions of scope, bound and free occurrences of variables, closed formulas, etc., defined as in FOL. In fact, we consider only closed formulas as specifications of temporal properties to verify.

For formulas of the form $\mu Z.\Phi$ and $\nu Z.\Phi$, we require the *syntactic monotonicity* of Φ wrt Z: every occurrence of the variable Z in Φ must be within the scope of an even number of negation signs. In $\mu\mathcal{L}$, given the requirement of syntactic monotonicity, the least fixpoint $\mu Z.\Phi$ and the greatest fixpoint $\nu Z.\Phi$ always exist.

To define the meaning of a $\mu\mathcal{L}$ formula over an artifact system, we resort to transition systems. Let $\mathfrak{A}$ be a transition system generated by a given progression mechanism over the artifact system $\mathcal{A}$. We denote by $\Sigma_{\mathfrak{A}}$ the states of $\mathfrak{A}$, and by $\mathcal{C}_{\mathfrak{A}}$ all terms (which are in general infinite) occurring in any state of $\mathfrak{A}$. Notice that trivially $\mathcal{C}_{\mathcal{I}_0} \subseteq \mathcal{C}_{\mathfrak{A}}$.

Let $\mathcal{V}$ be a predicate and individual variable valuation on $\mathfrak{A}$, i.e., a mapping from predicate variables to subsets of the states $\Sigma_{\mathfrak{A}}$, and from individual variables to constants in $\mathcal{C}_A$. Then, we assign meaning to $\mu\mathcal{L}$ formulas by associating to $\mathfrak{A}$ and $\mathcal{V}$ an *extension function* $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$, which maps $\mu\mathcal{L}$ formulas to subsets of $\Sigma_{\mathfrak{A}}$. The extension function is defined inductively as shown in Figure 2, where $Q\mathcal{V}$ (resp., $Z\mathcal{V}$) denotes the application of variable valuation $\mathcal{V}$ to query Q (resp., variables Z), and $\mathit{ans}(Q\mathcal{V}, \mathcal{I})$ denotes the result of evaluating the (boolean) query $Q\mathcal{V}$ over the instance $\mathcal{I}$. Moreover, $\mathcal{I} \Rightarrow_{\mathfrak{A}} \mathcal{I}'$ holds iff the progression mechanism allows to progress from $\mathcal{I}$ to $\mathcal{I}'$.

Intuitively, the extension function $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$ assigns to the various $\mu\mathcal{L}$ constructs the following meanings: The boolean connectives have the expected meaning, while (individual) quantification involving transitions from some state to the next is restricted to constants of $\mathcal{C}_{\mathcal{I}_0}$. The extension of $\Diamond\Phi$ consists of the states $\mathcal{I}$ such that for *some* state $\mathcal{I}'$ with $\mathcal{I} \Rightarrow_{\mathfrak{A}} \mathcal{I}'$, we have that Φ holds in $\mathcal{I}'$, while the extension of $\Box\Phi$ consists of the states $\mathcal{I}$ such that for *all* states $\mathcal{I}'$ with $\mathcal{I} \Rightarrow_{\mathfrak{A}} \mathcal{I}'$, we have that Φ holds in $\mathcal{I}'$. The extension of $\mu Z.\Phi$ is the *smallest subset* $\mathcal{E}_\mu$ of $\Sigma_{\mathfrak{A}}$ such that, assigning to Z the extension $\mathcal{E}_\mu$, the resulting extension of Φ is contained in $\mathcal{E}_\mu$. That is, the extension of $\mu Z.\Phi$ is the *least fixpoint* of the operator $(\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z/\mathcal{E}]}$ (here $\mathcal{V}[Z/\mathcal{E}]$ denotes the predicate valuation obtained from $\mathcal{V}$ by forcing the valuation of Z to be $\mathcal{E}$). Similarly, the extension of $\nu Z.\Phi$ is the *greatest subset* $\mathcal{E}_\nu$ of $\Sigma_{\mathfrak{A}}$ such that, assigning to Z the extension $\mathcal{E}_\nu$, the resulting extension of Φ contains $\mathcal{E}_\nu$. That is, the extension of $\nu Z.\Phi$ is the *greatest fixpoint* of the operator $(\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z/\mathcal{E}]}$. When Φ is a closed formula, $(\Phi)^{\mathfrak{A}}_{\mathcal{V}}$ does not depend on $\mathcal{V}$, and we denote it by $\Phi^{\mathfrak{A}}$.

We say that a closed $\mu\mathcal{L}$ formula Φ holds for $\mathfrak{A}$, denoted as $\mathfrak{A} \models \Phi$, iff $\mathcal{I}_0 \in \Phi^{\mathfrak{A}}$. We call *model checking* verifying whether $\mathfrak{A} \models \Phi$ holds.

Example 2 (Continues from Example 1). Now that we have defined our constraints formalism, we are in the position to express the constraints informally discussed in Example 1.

For ReqOrders, we first define formulas corresponding to the phases of the diagram in Figure 1:

$$\begin{aligned}
\psi_A &= \neg\exists x, y, z.\mathsf{ROItem}(x, y, z)\\
\psi_B &= \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow z = \mathsf{requested}) \wedge \exists x, y.\mathsf{ROItem}(x, y, \mathsf{requested})\\
\psi_C &= \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow (z = \mathsf{requested} \vee z = \mathsf{purchased})) \wedge\\
&\quad\ \exists x, y.\mathsf{ROItem}(x, y, \mathsf{requested}) \wedge \exists x, y.\mathsf{ROItem}(x, y, \mathsf{purchased})\\
\psi_D &= \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow z = \mathsf{purchased}) \wedge \exists x, y.\mathsf{ROItem}(x, y, \mathsf{purchased})\\
\psi_E &= \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow (z = \mathsf{purchased} \vee z = \mathsf{shipped})) \wedge\\
&\quad\ \exists x, y.\mathsf{ROItem}(x, y, \mathsf{purchased}) \wedge \exists x, y.\mathsf{ROItem}(x, y, \mathsf{shipped})\\
\psi_F &= \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow z = \mathsf{shipped}) \wedge \exists x, y.\mathsf{ROItem}(x, y, \mathsf{shipped}).
\end{aligned}$$

Then, the dynamic constraints of ReqOrders are captured by the formula

$$\Phi_{RO} = \psi_A \wedge \nu Z.(\textstyle\bigwedge_{i=1,\ldots,11} \Phi_i \wedge \Box Z).$$

It requires that in the initial state of $\mathfrak{A}$ there are not any items included in any pending order, i.e., the relation ROItem is empty, and that all formulas Φ_i listed below hold in every state. Each of Φ_1 to Φ_6 corresponds to a single transition as in Figure 1, expressing the constraint that the artifact remains in its current phase *until* it reaches the following one, also requiring that such a phase is eventually reached in a finite number of steps, and that no other phase is reached until then:

$$\begin{aligned}
\Phi_1 &= \psi_A \rightarrow \mu Z.(\psi_B \vee (\psi_A \wedge \Box Z)) & \Phi_4 &= \psi_D \rightarrow \mu Z.(\psi_E \vee (\psi_D \wedge \Box Z))\\
\Phi_2 &= \psi_B \rightarrow \mu Z.(\psi_C \vee (\psi_B \wedge \Box Z)) & \Phi_5 &= \psi_E \rightarrow \mu Z.(\psi_F \vee (\psi_E \wedge \Box Z))\\
\Phi_3 &= \psi_C \rightarrow \mu Z.(\psi_D \vee (\psi_C \wedge \Box Z)) & \Phi_6 &= \psi_F \rightarrow \mu Z.(\psi_A \vee (\psi_F \wedge \Box Z)).
\end{aligned}$$

The remaining formulas express static constraints, specifically inclusion dependencies and range restrictions:

$$\begin{aligned}
\Phi_7 &= \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow \exists u, v.\mathsf{RO}(x, u, v))\\
\Phi_8 &= \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow \mathsf{Status}(z))\\
\Phi_9 &= \forall x.(\mathsf{Status}(x) \rightarrow (x = \mathsf{requested} \vee x = \mathsf{purchased} \vee x = \mathsf{shipped}))\\
\Phi_{10} &= \forall x, y, z.(\mathsf{ROItem}(x, y, z) \rightarrow \exists w.\mathsf{LineItem}(y, w))\\
\Phi_{11} &= \forall x, y, z.(\mathsf{RO}(x, y, z) \rightarrow (\mathsf{Requester}(y) \wedge \mathsf{Buyer}(z)).
\end{aligned}$$

For ProcOrders, we just need to express some static specifications over instances. Hence, Φ_{PO} is the conjunction of the following formulas, expressing inclusion dependency constraints:

$$\begin{aligned}
&\nu Z.(\forall x, y, z.(\mathsf{PO}(x, y, z) \rightarrow \mathsf{Supplier}(z)) \wedge \Box Z)\\
&\nu Z.(\forall x, y, z.(\mathsf{POItem}(x, y, z) \rightarrow \exists u.\mathsf{PO}(x, y, u)) \wedge \Box Z)\\
&\nu Z.(\forall x, y, z.(\mathsf{POItem}(x, y, z) \rightarrow \exists u.\mathsf{LineItem}(u, z)) \wedge \Box Z).
\end{aligned}$$

Finally, the set of inter-artifact dynamic constraints Φ_{inter} is the conjunction of the following formulas:

$$\begin{array}{l}
\nu Z.(\forall x,y.(\mathsf{ReqOrders.LineItem}(x,y) \leftrightarrow \mathsf{ProcOrders.LineItem}(x,y)) \wedge \Box Z) \\
\nu Z.(\forall x,y,z.(\mathsf{POItem}(x,y,z) \rightarrow \mathsf{ROItem}(y,z,\mathsf{purchased})) \wedge \Box Z) \\
\nu Z.(\forall x,y,z.(\mathsf{PO}(x,y,z) \rightarrow \exists w,k.\mathsf{RO}(y,w,k)) \wedge \Box Z).
\end{array}$$

The first formula requires that the LineItem relations in both artifacts have the same set of tuples, the second one that every item belonging to a procurement order is also included in some requisition order, and the third one that every procurement order corresponds to a requisition order. ■

4 Processes over Artifact Systems

We now concentrate on progression mechanisms for relational artifact systems. In particular, we specify such a mechanism in terms of one or more *processes* that use actions as atomic steps. *Actions* represent atomic tasks or services that act over the artifacts and make them evolve.

Actions. We give a formal specification of actions in terms of preconditions and postconditions, inspired by the notion of mapping in the literature on data exchange [16]. However, we generalize such a notion in order to include negation, arbitrary quantification in preconditions, and the generation of new terms, through the use of *Skolem functions* in postconditions. Notice that, while it is conceivable that most of the actions will act on one artifact only, we do not make such a restriction. Indeed our actions are generally inter-artifact, which lets us easily account for synchronisation between artifacts.

An *action* ρ for $\mathcal{A}$ has the form

$$\rho(p_1,\ldots,p_m) : \{e_1,\ldots,e_m\} \qquad \text{where:}$$

– $\rho(p_1,\ldots,p_m)$ is the *signature* of the action, constituted by a name ρ and a sequence $p_1,\ldots,p_m$ of *input parameters* that need to be substituted by constants for the execution of the action, and
– $\{e_1,\ldots,e_m\}$ is a set of effects, called the *effects' specification.*

We denote by σ a (ground) substitution for the input parameters with terms not involving variables. Given such a substitution σ, we denote by $\rho\sigma$ the action with actual parameters. All effects in the effects' specification are assumed to take place simultaneously. Specifically, an *effect* e_i has the form

$$q_i \rightsquigarrow I'_i \qquad \text{where:}$$

– q_i is a query whose terms are variables $\boldsymbol{x}$, action parameters, and constants from $\mathcal{C}_{\mathcal{I}_0}$. Moreover, q_i has the form $q_i^+ \wedge Q_i^-$, where q_i^+ is a UCQ, and Q_i^-, is an arbitrary FOL formula whose free variables are included in those of q_i^+. Intuitively, q_i^+ selects the tuples to instantiate the effect, and Q_i^- filters away some of them.

– I_i' is a set of facts for the artifacts in $\mathcal{A}$, which includes as terms: terms in $\mathcal{C}_{\mathcal{I}_0}$, input parameters, free variables of q_i, and in addition terms formed by applying an arbitrary Skolem function to one of the previous kinds of terms. Such Skolem terms are used as witnesses of values chosen by the external user/environment when executing the action. Notice that different effects can share a same Skolem function.

Given an instance $\mathcal{I}$ of $\mathcal{A}$, an effect e_i as above, and a substitution σ for the parameters of e_i, the effect e_i extracts from $\mathcal{I}$ the set $ans(q_i\sigma, \mathcal{I})$ of tuples of terms, and for each such tuple θ asserts the set $I_i'\sigma\theta$ of facts obtained from $I_i'\sigma$ by applying the substitution θ for the free variables of q_i. In particular, in the resulting set of facts we may have terms of the form $f(\boldsymbol{t})\sigma\theta$ where $\boldsymbol{t}$ is a set of terms that may be either free variables in $\boldsymbol{x}$, parameters, or terms in $\mathcal{C}_{\mathcal{I}_0}$. We denote by $e_i\sigma(\mathcal{I})$ the overall set of facts, i.e., $e_i\sigma(\mathcal{I}) = \bigcup_{\theta \in ans(q_i\sigma, \mathcal{I})} I_i'\sigma\theta$. The overall *effect* of the action ρ with parameter substitution σ over $\mathcal{I}$ is a new instance $\mathcal{I}' = do(\rho\sigma, \mathcal{I}) := \bigcup_{1 \leq i \leq m} e_i\sigma(\mathcal{I})$ for $\mathcal{A}$.

Some observations are in order: *(i)* In the formalization above, actions are *deterministic*, in the sense that, given an instance $\mathcal{I}$ of $\mathcal{A}$ and a substitution σ for the parameters of an action ρ, there is a *single* instance $\mathcal{I}'$ that is obtained as the result of executing ρ in $\mathcal{I}$. *(ii)* The effects of an action are naturally a form of update of the previous state, and not of belief revision [15]. That is, we never learn new facts on the state in which an action is executed, but only on the state resulting from the action execution. *(iii)* We do not make any persistence (or frame) assumption in our formalization [20]. In principle at every move we substitute the whole old state, i.e., instance, $\mathcal{I}$, with a new one, $\mathcal{I}'$. On the other hand, it should be clear that we can easily write effect specifications that *copy* big chunks of the old state into the new one. For example, $R(\boldsymbol{x}) \rightsquigarrow R(\boldsymbol{x})$ copies the entire set of assertions involving the relation R.

Processes. Essentially processes are (possibly nondeterministic) programs that use artifacts in $\mathcal{A}$ to store their (intermediate and final) computation results, and use actions as atomic instructions. We assume that at every time the current instance $\mathcal{I}$ can be arbitrarily queried through the query answering services, while it can be updated only through actions. Notice that, 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, as often done in formal verification [4,11]. There can be many ways to provide the control flow specification for processes for $\mathcal{A}$. Here we adopt a very simple rule-based mechanism. Notice, however, that our results can be immediately generalized to any process formalism whose processes control flow is finite-state. Notice also that the transition system associated to a process over an artifact might not be finite-state, since its state is formed by both the *control flow* state of the process and the *data* in the artifact system, which are in general unbounded.

Formally, a process Π over a relational artifact system $\mathcal{A}$ is a pair $\langle \boldsymbol{\rho}, \boldsymbol{\pi} \rangle$, where $\boldsymbol{\rho}$ is a finite set of actions and $\boldsymbol{\pi}$ is a finite set of condition-action rules.

A *condition-action rule* π in $\boldsymbol{\pi}$ is an expression of the form

$$Q \mapsto \rho,$$

where ρ is an action in $\boldsymbol{\rho}$ and Q is a FOL formula over artifacts' relations whose free variables are exactly the parameters of ρ, and whose other terms can be either quantified variables or terms in $\mathcal{C}_{\mathcal{I}_0}$. Such a rule has the following semantics: for each tuple σ for which condition Q holds, the action ρ with actual parameters σ *can* be executed. If ρ has no parameters then Q will be a boolean formula. Observe that processes don't force the execution of actions but constrain them: the user of the process will be able to choose any of the actions that the rules forming the process allow.

The *execution* of a process $\Pi = \langle \boldsymbol{\rho}, \boldsymbol{\pi} \rangle$ over a relational artifact system $\mathcal{A}$ is defined as follows: we start from $\mathcal{I}_0$, and for each rule $Q \mapsto \rho$ in $\boldsymbol{\pi}$, we evaluate Q, and for each tuple σ returned, we execute $\rho\sigma$, obtaining a new instance $\mathcal{I}' = do(\rho\sigma, \mathcal{I}_0)$, and so on. In this way we build a *transition system* $\Upsilon(\Pi, \mathcal{A})$ whose states represent possible system instances, and where each transition represents the execution of an instantiated action that is allowed according to the process. A transition $\mathcal{I} \Rightarrow_{\Upsilon(\Pi,\mathcal{A})} \mathcal{I}'$ holds iff there exists a rule $Q \mapsto \rho$ in Π such that there exists a $\sigma \in ans(Q, \mathcal{I})$ and $\mathcal{I}' = do(\rho\sigma, \mathcal{I})$. That is, there exist a rule in Π that can fire on $\mathcal{I}$ and produce an instantiated action $\rho\sigma$, which applied on $\mathcal{I}$, results in $\mathcal{I}'$.

The transition system $\Upsilon(\Pi, \mathcal{A})$ captures the behavior of the process Π over the whole system $\mathcal{A}$. We are interested in formally verifying properties of processes over artifact-based systems, in particular we are interested in *conformance* and *verification*, defined as follows:

Conformance. Given a process Π and an artifact system $\mathcal{A}$, the process is said to be acceptable if it fulfills all intra-artifact and inter-artifact dynamic constraints. In this case, we say that Π *conforms to* $\mathcal{A}$. In order to formally check *conformance*, we can resort to model checking and verify that:

$$\Upsilon(\Pi, \mathcal{A}) \models \Phi_{inter} \wedge \bigwedge_{i=1,\ldots,n} \Phi_i.$$

Verification. Apart from intra-artifacts and inter-artifact dynamic constraints, we are interested in other dynamic properties of the process over the artifact system. We say that a process Π over an artifact system $\mathcal{A}$ *verifies* a dynamic property Φ expressed in $\mu\mathcal{L}$ if $\Upsilon(\Pi, \mathcal{A}) \models \Phi$.

It becomes evident that model checking of the transition system $\Upsilon(\Pi, \mathcal{A})$ generated by a process over an artifact system is the critical form of reasoning needed in our framework. We are going to study such a reasoning task next.

Example 3 (Continues from Example 2). We consider a process $\Pi = \langle \boldsymbol{\rho}, \boldsymbol{\pi} \rangle$ constituted by the following actions $\boldsymbol{\rho}$ and conditions-action rules $\boldsymbol{\pi}$. When specifying an action, we will use $[\ldots]$ to delimit each of the two parts q_i^+ and Q_i^- of the formula $q_i^+ \wedge Q_i^-$ in the left-hand side of an effect specification. Note that in such a formula the part corresponding to Q_i^- might be missing.

Actions. The set $\boldsymbol{\rho}$ of actions is the following. Action $\mathsf{RequestItem}(r,i,b)$ is used by the requester r to request a new line item i to buyer b. Such an action results in adding i to the requisition order of r. Notice that the *RoCode* denoting the requisition order is computed as a function of r and b only: performing this action multiple times for the same requester and buyer will result into adding line items to the same requisition order.

$$\begin{aligned}\mathsf{RequestItem}(r,i,b):\ \{\ &[\exists w.(\mathsf{Requester}(r)\wedge \mathsf{LineItem}(i,w)\wedge \mathsf{Buyer}(b))] \rightsquigarrow\\ &\quad\{\mathsf{RO}(f(r,b),r,b),\ \mathsf{ROItem}(f(r,b),i,\mathsf{requested})\},\\ &\mathit{CopyAll}\ \}\end{aligned}$$

Action $\mathsf{Purchase}(r,i,b,s)$ is used by buyer b for purchasing an item i belonging to requisition order r from supplier s, thus creating (or updating) procurement orders (i.e., the relation $\mathsf{ProcOrders.PO}$) and updating the status of the corresponding items kept by the relation $\mathsf{ReqOrders.ROItem}$. Again, notice that *PoCode* is not a function of the item i passed as parameter. By writing $\mathit{CopyAll}\setminus\mathsf{ROItem}$ we denote the copy of all relations except ROItem.

$$\begin{aligned}\mathsf{Purchase}(r,i,b,s):\ \{\ &[\exists w.\mathsf{RO}(r,w,b)\wedge \mathsf{ROItem}(r,i,\mathsf{requested})\wedge \mathsf{Supplier}(s)] \rightsquigarrow\\ &\quad\{\mathsf{PO}(g(r,b,s),r,s),\ \mathsf{POItem}(g(r,b,s),r,i),\\ &\quad\ \mathsf{ROItem}(r,i,\mathsf{purchased})\},\\ &[\mathsf{ROItem}(x,y,z)]\wedge[\neg\mathsf{ROItem}(r,i,\mathsf{requested})] \rightsquigarrow \{\mathsf{ROItem}(x,y,z)\},\\ &\mathit{CopyAll}\setminus\mathsf{ROItem}\ \}\end{aligned}$$

The following actions are used to ship all items included in a given procurement order p, and to deliver items belonging to a requisition order r to the corresponding requester, respectively. Notice that the first avoids copying all facts concerning p whereas the latter does the same with all facts related to r.

$$\begin{aligned}\mathsf{Ship}(p):\ \{\ &[\mathsf{POItem}(p,x,y)\wedge \exists z.\mathsf{ROItem}(x,y,z)] \rightsquigarrow \{\mathsf{ROItem}(x,y,\mathsf{shipped})\},\\ &[\mathsf{POItem}(x,y,z)]\wedge[\neg\mathsf{POItem}(p,y,z)] \rightsquigarrow \{\mathsf{POItem}(x,y,z)\},\\ &[\mathsf{PO}(x,y,z)]\wedge[\neg\mathsf{PO}(p,y,z)] \rightsquigarrow \{\mathsf{PO}(x,y,z)\},\\ &\mathit{CopyAll}\setminus(\mathsf{POItem}\text{ and }\mathsf{PO})\ \}\\ \mathsf{Deliver}(r):\ \{\ &[\mathsf{ROItem}(x,y,z)]\wedge[\neg\mathsf{ROItem}(r,y,z)] \rightsquigarrow \{\mathsf{ROItem}(x,y,z)\},\\ &[\mathsf{RO}(x,y,z)]\wedge[\neg\mathsf{RO}(r,y,z)] \rightsquigarrow \{\mathsf{RO}(x,y,z)\},\\ &\mathit{CopyAll}\setminus(\mathsf{ROItem}\text{ and }\mathsf{RO})\ \}\end{aligned}$$

Condition-action rules. In each condition-action rule of our process, we instantiate the parameters passed to the action, while simply checking that they are meaningful, i.e., that they are in the current instance. Hence:

$$\begin{aligned}\boldsymbol{\pi}=\{\ &\exists x.(\mathsf{Requester}(r)\wedge\mathsf{LineItem}(i,x)\wedge\mathsf{Buyer}(b))\mapsto\mathsf{RequestItem}(r,i,b),\\ &\exists x.(\mathsf{RO}(r,x,b)\wedge\mathsf{ROItem}(r,i,\mathsf{requested})\wedge\mathsf{Supplier}(s))\mapsto\mathsf{Purchase}(r,i,b,s),\\ &\exists x,y.\mathsf{PO}(p,x,y)\mapsto\mathsf{Ship}(p),\qquad \exists x,y.\mathsf{RO}(r,x,y)\mapsto\mathsf{Deliver}(r)\ \}\end{aligned}$$

We close our example by observing that the process we have specified conforms to the lifecycle in Example 2. ■

5 Undecidability of Conformance and Verification

Next, we consider conformance and verification over relational artifact systems. We show that they are both undecidable in general. The undecidability result does not come as a surprise, since the transition system of a process over an artifact system can easily be infinite-state. Moreover, our framework is so general that it does not enforce a regularity of the infinite state space that would allow one to apply known results on model checking on infinite state systems. However, we show that the undecidability holds even in a very simple case.

We consider a relational artifact system of the form $\mathcal{A}_u = \langle \{A\}, \mathsf{true}\rangle$ with $A = \langle \boldsymbol{R}, I_0, \mathsf{true}\rangle$. That is $\mathcal{A}_u$ is formed by a single artifact A with no intra-artifact or inter-artifact dynamic constraints. In addition, we consider processes with only one action ρ_u and only one condition-action rule $\mathsf{true} \mapsto \rho_u$ that has a true condition and hence allows the execution of the action ρ_u at every moment. The action ρ_u is without parameters, its effects have the form $q_i^+ \rightsquigarrow I_i'$, where q_i^+ is a CQ (hence without any form of negation and of universal quantification), and it includes *CopyAll* effects. We call these kinds of relational artifact systems and processes *simple*. The next lemma shows that it is undecidable to verify in such cases the $\mu\mathcal{L}$ formula $\mu Z.(q \vee \Diamond Z)$, expressing that there exists a sequence of action executions leading to an instance where a boolean CQ q holds.

Lemma 1. *Verifying whether the $\mu\mathcal{L}$ formula $\mu Z.(q \vee \Diamond Z)$ holds for a simple process over a simple artifact is undecidable.*

Proof (sketch). We observe that we can use the set of effects of ρ_u to encode a set of tuple-generating dependencies (TGDs) [3]. Hence we can reduce to the above verification problem the problem of answering boolean CQs in a relational database under a set of TGDs, which is undecidable [5]. (In fact, special care is needed because of the use of Skolem terms instead of labeled nulls.) $\square$

Theorem 1. *Conformance checking and verification are both undecidable for processes over relational artifacts systems.*

Proof (sketch). Lemma 1 gives us undecidability of verification, already for simple relational artifact systems and processes. To get undecidability of conformance it is sufficient to consider the simple process $\varPi_u$ over relational artifact systems of the form $\mathcal{A}_{cu} = \langle \{A_c\}, \mathsf{true}\rangle$, with $A_c = \langle \boldsymbol{R}, I_0, \mu Z.(q \vee \Diamond Z)\rangle$. Note that $\mathcal{A}_{cu}$ is a variant of simple artifact systems $\mathcal{A}_u$ in which the artifact has as intra-artifact dynamic constraint exactly $\mu Z.(q \vee \Diamond Z)$. The claim follows again from Lemma 1, considering that, by definition, checking conformance of the simple process $\varPi_u$ wrt $\mathcal{A}_{cu}$ is equivalent to checking whether $\varUpsilon(\varPi_u, \mathcal{A}_u) \models \mu Z.(q \vee \Diamond Z)$. $\square$

6 Decidability of Weakly Acyclic Processes

Next we tackle decidability, and, inspired by the recent literature on data exchange [16], we isolate a notable case of processes over relational artifact systems for which both conformance and verification are decidable. Our results rely on the possibility of building a special process that we call "positive approximate". For such a process there exists a tight correspondence between the application of an action and a step in the chase of a set of TGDs [3,16]

Given a process $\Pi = \langle \boldsymbol{\rho}, \boldsymbol{\pi} \rangle$, the *positive approximate* of Π is the process $\Pi^+ = \langle \boldsymbol{\rho}^+, \boldsymbol{\pi}^+ \rangle$ obtained from Π as follows. For each action ρ in $\boldsymbol{\rho}$, there is an action ρ^+ in $\boldsymbol{\rho}^+$, obtained from ρ by

– removing all input parameters from the signature, and
– substituting each effect $q_i^+ \wedge Q_i^- \rightsquigarrow I_i'$ with the one that uses only the *positive* part of the head of the effect specification, i.e., with $q_i^+ \rightsquigarrow I_i'$.

Note that the variables in q_i^+ that used to be parameters in ρ, become free variables in ρ^+. Then, for each condition-action rule $Q \mapsto \rho$ in $\boldsymbol{\pi}$, there is a rule $\mathsf{true} \mapsto \rho^+$ in $\boldsymbol{\pi}^+$. Hence, Π^+ allows for executing every action at every step.

Now, relying again on the parallelism between chase in data exchange and action execution in artifact systems, we take advantage of the notion of weak acyclicity in data exchange [16] to devise an interesting class of processes that are guaranteed to generate a finite-state transition system, when run over a relational artifact system. This in turn guarantees decidability of conformance and verification.

Let Π be a process over an artifact system $\mathcal{A}$, and $\Pi^+ = \langle \boldsymbol{\rho}^+, \boldsymbol{\pi}^+ \rangle$ its positive approximate. We call *dependency graph* of Π^+ the following (edge labeled) directed graph:

Nodes: for every artifact $A = \langle \boldsymbol{R}, I_0, \Phi \rangle$ of $\mathcal{A}$, every relation symbol $R_i \in \boldsymbol{R}$, and every attribute *att* or R_i, there is a node (R_i, att) representing a position;
Edges: for every action ρ^+ of $\boldsymbol{\rho}^+$, every effect $q_i^+(\boldsymbol{t}) \rightsquigarrow I_i'(\boldsymbol{t}', f_1(\boldsymbol{t}_1), \ldots, f_n(\boldsymbol{t}_n))$ of ρ^+ (where for convenience we have made explicit the terms occurring in q_i^+ and I_i', and where consequently $\boldsymbol{t}'$, $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n \subseteq \boldsymbol{t}$ are either constants or variables), every variable $x \in \boldsymbol{t}$, and every occurrence of x in q_i^+ in position p, there are the following edges:
  – for every occurrence of x in I_i' in position p', there is an edge $p \rightarrow p'$;
  – for every Skolem term $f_k(t_k)$ such that $x \in \boldsymbol{t}_k$ occurs in I_i' in position p'', there is a *special edge* (i.e., one labeled by $*$) $p \xrightarrow{*} p''$.

We say that Π is *weakly acyclic* if the dependency graph of Π^+ has no cycle going through a special edge.

Intuitively, ordinary edges keep track of the fact that a value may propagate from position p to position p' in a possible trace. Moreover, special edges keep track of the fact that a value in position p can be taken as parameter of a Skolem function, thus contributing to the creation of a (not necessarily new) value in any position p''. If a cycle goes through a special edge, then a new value appearing in a certain position may determine the creation of another one, in the same position, later during the execution of actions. Since this may happen again and again, no bound can be put on the number of newly generated Skolem terms, and thus on the number of new values appearing in the instance. Note that the definition allows for cycles as long as they do not include special edges.

Lemma 2. *Let Π be a* weakly acyclic *process over a relational artifact system $\mathcal{A}$ with initial instance $\mathcal{I}_0$, and let Π^+ be the positive approximate of Π. Then there exists a polinomial in the size of $\mathcal{I}_0$ that bounds the size of every instance generated by Π^+.*

Proof (sketch). The proof follows the line of that in [16] on chase termination for weakly acyclic TGDs. The difference here is that we use Skolem terms and don't have the inflationary behavior of TGDs in applying action effects. However, the key notion of rank used in [16] can still be used to bound the number of terms generated through the Skolem functions. □

Notice that, as a direct result of this lemma, the transition system generated by the positive approximate over $\mathcal{A}$ has a number of states that is finite, and in fact at most exponential in the size of the initial instance $\mathcal{I}_0$ of $\mathcal{A}$. Now we show that a similar result holds for the original process Π. The key to this is the following observation that easily follows from the definition of ρ^+ for an action ρ.

Lemma 3. *For every action ρ over $\mathcal{A}$, instances $\mathcal{I}_1$, $\mathcal{I}_2$ of $\mathcal{A}$, and ground substitution σ for the parameters of ρ, if $\mathcal{I}_1 \subseteq \mathcal{I}_2$ then $do(\rho\sigma, \mathcal{I}_1) \subseteq do(\rho^+, \mathcal{I}_2)$.*

We can extend the result above to any sequence of actions, by induction on the length of the sequence. Hence, we get that the instance obtained from the initial instance by executing a sequence of actions of the original process Π is contained in the instance obtained by executing the same sequence of actions of Π^+. From this observation, considering the bound in Lemma 2, we get the desired result for the original process.

Lemma 4. *Let Π be a* weakly acyclic *process over a relational artifact system $\mathcal{A}$ with initial instance $\mathcal{I}_0$. Then there exists a polinomial in the size of $\mathcal{I}_0$ that bounds the size of every instance generated by Π.*

From this, we obtain our main result.

Theorem 2. *Conformance and verification of $\mu\mathcal{L}$ formulas are decidable for weakly acyclic processes over relational artifact systems.*

Proof (sketch). From Lemma 4, it follows that the transition system generated by a weakly acyclic process over a relational artifact system $\mathcal{A}$ has a number of states that is at most exponential in the size of the initial instance $\mathcal{I}_0$ of $\mathcal{A}$. The claim then follows from known results on verification of μ-calculus formulas over finite transition systems (see e.g., [11]). □

From the exponential bound on the number of states of the generated transition system mentioned in the proof above, we get not only decidability of verification and conformance, but also an ExpTime upper bound for its computational complexity (assuming a bound on the nesting of fixpoints).

7 Conclusions

In this paper we have looked at foundations of artifact-centric systems, and we have shown that weakly acyclic processes over relational artifacts are very interesting both from a formal point of view, since reasoning on them is decidable, and from a practical point of view, since weak-acyclicity appears to be a quite acceptable restriction.

Further research can take several directions. First, one can easily focus on different temporal logics for specifying dynamic constraints, such as LTL or CTL. Observe that the results presented here would apply, being *mu*-calculus more expressive than both LTL and CTL, but certainly they can be refined. Second, we may introduce special equality generating constraints to allow to equate different terms, e.g., a Skolem term and a constant. We are particularly interested in how to extend our decidability result to this case. Also we have assumed that no artifacts are added or destroyed during the execution of a process. We are very interested in relaxing this restriction. Notice that to do so we would need to introduce Skolem terms to denote artifacts, and then extend the notion of weakly acyclic process to block the infinite accumulation of new artifacts. Finally, we are interested in moving from a relational setting to a semantic one, based on ontologies for data access [8], believing that similar results apply.

Acknowledgments. This work has been supported by the EU FP7-ICT Project ACSI (grant no. 257593).

References

1. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A framework for lightweight interacting workflow processes. Int. J. of Cooperative Information Systems 10(4), 443–481 (2001)
2. Abiteboul, S., Bourhis, P., Galland, A., Marinoiu, B.: The AXML artifact model. In: Proc. of TIME 2009, pp. 11–17 (2009)
3. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley Publ. Co., Reading (1995)
4. Baier, C., Katoen, J.P., Guldstrand Larsen, K.: Principles of Model Checking. The MIT Press, Cambridge (2008)
5. Beeri, C., Vardi, M.Y.: The implication problem for data dependencies. In: Even, S., Kariv, O. (eds.) ICALP 1981. LNCS, vol. 115, pp. 73–85. Springer, Heidelberg (1981)
6. Bhattacharya, K., Gerede, C., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 288–304. Springer, Heidelberg (2007)
7. Bhattacharya, K., Guttman, R., Lyman, K., Heath, F.F., Kumaran, S., Nandi, P., Wu, F.Y., Athma, P., Freiberg, C., Johannsen, L., Staudt, A.: A model-driven approach to industrializing discovery processes in pharmaceutical research. IBM Systems Journal 44(1), 145–162 (2005)

8. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The Mastro system for ontology-based data access. Semantic Web J (2011)
9. Cangialosi, P., De Giacomo, G., De Masellis, R., Rosati, R.: Conjunctive artifact-centric services. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 318–333. Springer, Heidelberg (2010)
10. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Bull. on Data Eng. 32(3), 3–9 (2009)
11. Emerson, E.A.: Automated temporal reasoning about reactive systems. In: Moller, F., Birtwistle, G. (eds.) Logics for Concurrency. LNCS, vol. 1043, pp. 41–101. Springer, Heidelberg (1996)
12. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theor. Comp. Sci. 336(1), 89–124 (2005)
13. Fritz, C., Hull, R., Su, J.: Automatic construction of simple artifact-based business processes. In: Proc. of ICDT 2009, pp. 225–238 (2009)
14. Hull, R.: Artifact-centric business process models: Brief survey of research results and challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part II. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008)
15. Katsuno, H., Mendelzon, A.: On the difference between updating a knowledge base and revising it. In: Proc. of KR 1991, pp. 387–394 (1991)
16. Kolaitis, P.G.: Schema mappings, data exchange, and metadata management. In: Proc. of PODS 2005, pp. 61–75 (2005)
17. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. of PODS 2002, pp. 233–246 (2002)
18. Luckham, D.C., Park, D.M.R., Paterson, M.: On formalised computer programs. J. of Computer and System Sciences 4(3), 220–249 (1970)
19. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
20. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. The MIT Press, Cambridge (2001)
21. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Heidelberg (2007)