Conjunctive Artifact-Centric Services

Piero Cangialosi, Giuseppe De Giacomo, Riccardo De Masellis, and Riccardo Rosati

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA – Università di Roma
Via Ariosto, 25, 00185 Rome, Italy
lastname@dis.uniroma1.it

Abstract. Artifact-centric services are stateful service descriptions centered around "business artifacts", which contain both a *data schema* holding all the data of interest for the service, and a *lifecycle schema*, which specifies the process that the service enacts. In this paper, the data schemas are full-fledged relational databases, and the lifecycle schemas are specified as sets of condition-action rules, where conditions are evaluated against the current snapshot of the artifact, and where actions are suitable updates to database. The main characteristic of this work is that conditions and actions are based on conjunctive queries. In particular, we exploit recent results in data exchange to specify through tuple-generating-dependencies (tgds) the effects of actions. Using such basis we develop sound and complete verification procedures, which, in spite of the fact that the number of states of an artifact-centric service can be infinite, reduce to the finite case through a suitable use of homomorphism induced by the conjunctive queries.

1 Introduction

In the past years, what can be called the artifact-centric approach to modeling workflows and services has emerged, with the fundamental characteristic of considering both data and processes as first-class citizens in service design and analysis [21,17,11], see also [27,1]. In this approach the key elements of services are *(i)* data manipulated, which correspond to key business-relevant entities, *(ii)* the service lifecycle (i.e., the process that the service follows), and *(iii)* the tasks invoked and executed. Executing a task has effects on the data managed by the service, on the service state, as well as on the information exchanged with the external world. This "artifact-centric" approach provides a simple and robust structure for workflow and services, and has been demonstrated in practice to permit efficiency in business transformation [5,6]. From the formal point of view, artifact-centric services deeply challenge the research community by requiring simultaneous attention to both data and processes. Indeed, on the one hand they deal with full-fledged processes and require analysis in terms of verification of sophisticated temporal properties [10]. On the other hand, the presence of data [2] makes the usual analysis based on model checking of finite-state systems impossible in general, since when data evolution is taken into account the whole system becomes infinite state. In this paper, we provide a formal model for a family of artifact-centric services, based on the notion of conjunctive queries, used both to define preconditions and effects of tasks. In this setting we take advantage of the recent literature on data exchange and data integration

P.P. Maglio et al. (Eds.): ICSOC 2010, LNCS 6470, pp. 318–333, 2010.

[15,18], which has deeply investigated mapping between databases based on correspondences between conjunctive queries, the so call tuple-generating dependencies (tgds) in the database jargon [2]. In a nutshell, the core idea of our work is to consider the current state of the data and their state after the performance of a task as two databases related through a set of tgds. More precisely, our model follows the spirit of [5,16], but with important generalizations. The artifact "data schema" is a full-fledged relational database, which is used to hold relevant information about the artifact as it passes through the workflow. The "lifecycle schema", which is used to specify the possible ways that the artifact can pass through the workflow, is specified as a set of condition-action rules, where the condition is evaluated against the current snapshot of the artifact (i.e., the current state of the database), and where the actions are "tasks" invocations, which query the current snapshot and generate the next snapshot possibly introducing existential values representing inputs from the outside world. Similar to the context of semantic web services [24], the behaviors of the tasks used here are characterized using pre- and post-conditions. The key point, however, is that both pre-conditions and post-conditions are expressed as conjunctive queries. On top of such a system we introduce a powerful verification logic based on a variant of μ-calculus [19,22,13,7] to express temporal properties. Our verification logic is also based on conjunctive queries, in that it requires the atomic formulas to be conjunctive queries and disallows forms of negation of such queries. No limitations whatsoever are instead put on the fixpoint formulas that are the key element of the μ-calculus. The main result of the paper is showing that the resulting formalism, while quite expressive, and inherently infinite state, is decidable under a reasonable restriction, called *weak-acyclicity* [15], on the form of the tgds expressing the effects of actions. In particular we develop a sound, complete and terminating reasoning procedure for the verification formalism. The crux of the result is that conjunctive queries are unable to distinguish homomorphic equivalent databases, and under suitable but quite general circumstances, the number of homomorphically different states can be bounded to be finite. Thus we can reduce verification to model checking of a finite state abstraction (based on homomorphic equivalence) of the system.

2 Framework

The framework that we propose, called *conjunctive artifact-centric services*, merges data and processes following the artifact-centric approach. Namely an artifact is composed by the following three components:

- **Artifact Data Schema**, which captures the data schema of the information manipulated in the artifact. States of the processes correspond to instances to such a schema. Technically, the artifact data schema is a relational schema, and an instance is a relational database.
- **Artifact Tasks**, which is the set of atomic actions that can be used to manipulate data in the artifacts, i.e., to compute new states given the current one. We assume that the user can freely query (through certain answers, see later) the current data instance, and for simplicity we disregard a specific treatment of the output to the user. Technically, such tasks are specified in terms of dependencies between conjunctive queries.

– **Artifact lifecycle**, which specifies the actual process of the artifacts in terms of tasks that can be executed at each state of the process. Technically, the artifact lifecycle is specified in terms of condition-action rules, where the conditions are again based on conjunctive queries.

It should result immediately clear that such a setting produces in general infinite state processes, and that verification of such processes is in general undecidable. We will leverage on the notion of conjunctive query and the associated notion of homomorphism between instances to gain decidability of verification in spite of the infinite states.

2.1 Artifact Data Schema

Let us enter the formalism by showing how data are represented. As customary in relational databases, we consider an **artifact data schema** as a tuple $\mathcal{S} = \langle \boldsymbol{R}, \boldsymbol{c} \rangle$ where:

– $\boldsymbol{R} = R_1, \ldots, R_n$ is a finite set of *relational (predicate) symbols* each one with an associated arity;
– $\boldsymbol{c} = c_1, c_2, \ldots$ is a finite or countably infinite set of *constants*.

Given an artifact data schema $\mathcal{S}$, an **artifact data instance** over a schema $\mathcal{S}$ is a standard first-order interpretation with a *fixed interpretation doman*. More precisely a data instance is a couple $I = \langle \Delta, \cdot^I \rangle$ where:

– Δ is a countably infinite *domain* fixed a-priori for every data instance. For convenience we partition Δ into two countable infinite disjoint sets $const(\Delta)$ and $ln(\Delta)$, and we use the first set to interpret *constants*, while the second is needed to correctly interpret existentials, we call the latter *labeled nulls* (see later);
– $\cdot^I$ is an interpretation function that associates:
  • to each constant symbol c a constant $c^I \in const(\Delta)$ such that for each $c_1, c_2 \in const(\Delta)$ if $c_1 \neq c_2$ then $c_1^I \neq c_2^I$, namely we make the *Unique Name Assumption*, furthermore we require that every interpretation interprets contants in the same way, that i, given any two interpretation I and I', we have that $c_n^I = c_n^{I'}$ for each constant c_n;
  • to each m-ary relation symbol R_i a (finite) m-ary relation ${R_i}^I \subseteq \Delta^m$.

Intuitively, an artifact data instance is alike a relational database instance, since the $\cdot^{\mathcal{I}}$ function lists all tuples belonging to each relation.

We call a **fact** an expression $R_i(d_1, \ldots, d_m)$. We say that a fact belongs to and interpretation I iff $\boldsymbol{d} = \langle d_1, \ldots, d_m \rangle \in R_i^I$. We can characterize the interpretation function $\cdot^I$ simply by listing of its facts (notice that such a set is finite). Following the database literature [2], we call to the **active domain** $\bar{\Delta}_I$ of an instance I the set of domain elements appearing in facts of I.

To query data instances we use a special class of first-order formulas, widely used in database theory, which corresponds to relational algebra select-project-join queries: *conjunctive queries*. A **conjunctive query** is a formula cq of the form:

$$\exists \boldsymbol{y}.body(\boldsymbol{y}, \boldsymbol{x})$$

where $body$ is a conjunction of atomic formulas involving constants (but no labeled nulls), existentially quantified variables $\boldsymbol{y}$ and free variables $\boldsymbol{x}$.

Intuitively a conjunctive queries returns as answer the domain elements (both constants and nulls) that substitute to the free variables make the formula true in the data instance. More formally, let $I = \langle \Delta, \cdot^I \rangle$ an artifact instance, the *answer to a conjunctive query* $cq(\boldsymbol{x})$ with free variables $\boldsymbol{x}$, denoted by $cq(\boldsymbol{x})^I$ is defined as:

$$cq(\boldsymbol{x})^I = \{\eta \mid \langle I, \eta \rangle \models cq(\boldsymbol{x})\}$$

with $\eta : \boldsymbol{x} \to \Delta$ an assignment for the free variables. In fact, as usual in the database literature [2], we see assignments η simply as tuples of domain elements to be substituted to the free variables.

The characterizing property of conjunctive queries from the semantical point of view is that they are invariant under homomorphic equivalence [2]. That is if two data instances I and I' are homomorphic, then each boolean (without free variables) conjunctive query cq produces exactly the same (boolean) answer: $cq(\boldsymbol{x})^I = cq(\boldsymbol{x})^{I'}$.

Homomorphism [8] indeed plays a key role in our setting, so we remind its definition here. Given two instances $I_1 = \langle \Delta, \cdot^{I_1} \rangle$ and $I_2 = \langle \Delta, \cdot^{I_2} \rangle$ over the same schema $\mathcal{S}$, a **homomorphism** from I_1 to I_2, denoted by $h : I_1 \to I_2$, is a function from Δ to Δ such that:

1. for every constant c, we have that $h(c) = c$ and
2. for every $\langle d_1, \ldots, d_m \rangle \in R_i^{I_1}$, we have that $\langle h(d_1), \ldots, h(d_m) \rangle \in R_i^{I_2}$.

Two instances I_1 and I_2 are **homomorphically equivalent**, written $I_1 \overset{h}{=} I_2$, if there exist two homomorphisms $h_1 : I_1 \to I_2$ and $h_2 : I_2 \to I_1$.

A homomorphism $h : I_1 \to I_2$ preserves the interpretation of constants but not of labeled nulls of I_1, which are mapped either to constants or nulls in I_2, that is the homomorphism can "determine" some nulls values assigning them to constants. In other words, homomorphism interprets nulls of I_1 as existential values.

The existential interpretation of labeled nulls given by homomorphisms suggest a different way of answering to a conjunctive query, that essentially sees an interpretation as a *theory* where all nulls are treated as existential values. To make this notion precise, given an interpretation I we define the (*infinite*) set W_I of all interpretations $I' = \langle \Delta, \cdot^{I'} \rangle$ over $\mathcal{S}$ such that there exists an homomorphism $h : I \to I'$. Then we define the **Certain Answers** of a conjunctive query cq as:

$$cert^I(cq) = \bigcap_{w \in W_I} cq^{I_w}$$

That is the certain answers to a query are all those tuples of (active domain) elements (in fact constants) in I that are produced by the query in every interpretation I' such that there exists an homomorphism $h : I \to I'$. Formally, it can be shown that, the certain answers correspond to the tuples of constants such that, substituted to the free variables of the query, would make the resulting query logically implied by the theory constituted by a single conjunctive query formed by the logical AND of all facts in I, considering all labeled nulls as existentially quantified. From a more pragmatical point

of view, when using certain answers we consider the current instance as representative of several possible instances, and while we assume to have incomplete information on which is exactly the current instance, we still produce all answers that would be produced in all possible instances.

In our framework, we assume that the user can pose arbitrary conjunctive queries to the current instance, but require them to be evaluated returning the certain answers. In this way we become independent of the particular null values occurring in the data instance, since they are not returned as answers, though they can still be used as witness of existential quantified variables.

2.2 Artifact Tasks

A task is specified as a set of *effects* that it can produce, and when it is performed over the current (artifact) data instance, the result is a completely new data instance made up of a subset of the action's *effects*. The formalization of an effect is borrowed from the database and data exchange literature and in particular from the notion of *tuple generating dependencies* (tgds) [2,15]. A **(conjunctive) effect specification** ξ over a schema $\mathcal{S}$ is a formula of the form:

$$\exists \boldsymbol{y}.\phi(\boldsymbol{x},\boldsymbol{y},\boldsymbol{c}) \rightarrow \exists \boldsymbol{w}.\psi(\boldsymbol{x},\boldsymbol{w},\boldsymbol{d})$$

where ϕ and ψ are conjunctions of atoms over $\mathcal{S}$; $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{w}$ denote the variables and $\boldsymbol{c}, \boldsymbol{d}$ the constants occurring in ϕ and ψ. We call the left-hand side of ξ the *premise*, and the right-hand side the *conclusion*. Notice that both the premise and the conclusion are *conjunctive queries*. Formally, let $I = \langle \Delta, \cdot^I \rangle$ be an artifact instance over the schema $\mathcal{S}$, and $\xi = \exists \boldsymbol{y}\ \phi(\boldsymbol{x},\boldsymbol{y},\boldsymbol{c}) \rightarrow \exists \boldsymbol{w}\ \psi(\boldsymbol{x},\boldsymbol{w},\boldsymbol{d})$ an effect specification. The result of **enacting effect specification** ξ on I, is a set of facts $\xi(I)$ defined as follows:

> Let $\boldsymbol{\eta} = (\exists \boldsymbol{y}\ \phi(\boldsymbol{x},\boldsymbol{y},\boldsymbol{c}))^I$, be the answer to the query $\exists \boldsymbol{y}\ \phi(\boldsymbol{x},\boldsymbol{y},\boldsymbol{c})$ in I, then for each $\eta_i \in \boldsymbol{\eta}$ we proceed as follows. For each atomic formula $R_i(\boldsymbol{x},\boldsymbol{w},\boldsymbol{d})$ occurring in ψ, we include in $\xi(I)$ a new fact $R_i^{I'}(\boldsymbol{x},\boldsymbol{w},\boldsymbol{d})|_{\eta_i}^{\psi}$, obtained by substituting every variable in $\boldsymbol{x}$ with the corresponding element given by the assignment η_i, and every variable in $\boldsymbol{w}$ with a fresh (not appearing elsewhere) labeled null ln.

Intuitively, the premise, acting like a query, selects values form the current instance, while the conclusion builds the resulting instance by inserting them in possibly different positions of the schema, and by potentially introducing fresh elements, namely, the labeled nulls and fixed constants.

A **task** T for a schema $\mathcal{S}$ is specified as a set $\boldsymbol{\xi} = \{\xi_1, \ldots, \xi_n\}$ of conjunctive effect specifications. The result of **executing task** T on I, denoted by $I \xrightarrow{T} I^T$, is a new instance $I^T = \langle \Delta, \cdot^{I^T} \rangle$ on the same schema $\mathcal{S}$, obtained as the union of the enactments of each effect specification. Namely $I^T = \langle \Delta, \cdot^{I^T} \rangle$ where is the interpretation function $\cdot^{I^T}$ characterized by the facts $\bigcup_{\xi \in \boldsymbol{\xi}} \xi(I)$.

Let's make some key observations on such tasks. First, we observe that the role of the existential qualification on the two sides of an effect specification is very different.

The existential qualification on the left-hand side is the usual one used in conjunctive queries, which projects out variables used only to make joins. The existential qualification on the right-hand side, instead, is used as a witness of values that should be chosen by the user when executing the effect. In other words, the choice function used for assign witness to the existential on the right should be in the hand of the user. Here since we do not have such a choice at hand, we introduce a fresh null, to which we assign an existential meaning through homomorphism. Essentially we imply that there exists a choice made by the user of the value assigned to those variables.

The second observation is that we do not make any persistence (or frame [23]) assumption in our formalization. In principle at every move we substitute the whole old data instance with a new one. On the other hand, it should be clear that we can easily write effect specifications that *copy* big chunks of the old instance into the new one. For example, $R_i(\boldsymbol{x}) \rightarrow R_i(\boldsymbol{x})$ copies the entire extension of a relation R_i.

2.3 Artifact Lifecycle

The artifact lifecyle is defined in terms of condition/action rules, that specify, for every instance, which tasks can be executed. A **(condition/action) rule** for a schema $\mathcal{S}$ is a couple $\varrho = \langle \pi, T \rangle$ where π is a precondition, and T is a task. The precondition is a *closed* formula over $\mathcal{S}$ of the following form:

$$\pi ::= cq \mid \neg\pi \mid \pi_1 \wedge \pi_2$$

where cq is a boolean conjunctive query. Preconditions are arbitrary boolean combinations of boolean conjunctive queries interpreted under the certain answer semantics, namely:

$$\begin{array}{ll} I \rhd cq & \text{iff } cert^I(cq) = true \\ I \rhd \neg\pi & \text{iff } I \not\rhd \pi \\ I \rhd \pi_1 \wedge \pi_2 & \text{iff } I \rhd \pi_1 \text{ and } I \rhd \pi_2 \end{array}$$

In order to execute a task T, on an instance I, precondition π must certainly hold in I, written as $I \rhd \pi$, and, if this is the case, a new instance I^T is generated, according to T's effects.

Observe that, while we disallow negation in task effects so as to exploit the theory of conjunctive queries, which do not include negation, in the condition/action rules we allow for it, but to do so we actually require conditions to be based on certain answers of conjunctive queries, in this way we force a sort of "negation-as-failure" for negation [9].

Example 1. We illustrate here an example of specification. The scenario concerns an institution, e.g. a bank, that provides services to its customers, such as loans or money transfers. Every service has a distinct cost, that has to be paid in advance by customers that asked for it. A customer may inquire for the provision of a service, that first has to be approved by a supervisor, then paid, and finally provisioned by the bank. Moreover there are special "premier customers" that do not need the service's approval.

The **artifact schema** $\mathcal{S}$ consists of the following relation symbols: $Customer(\underline{cust\_ssn}, name)$ is the relation containing customers information; $Service(\underline{serv\_code}, cost)$ contains information about the

different types of services that the bank offers to its customers; $Service\_Claimed(\underline{serv\_code}, cust\_ssn)$ keeps track of information of services requested by clients; $Request\_Exam(\underline{serv\_code}, spv\_name, outcome)$ is the relation containing the names of supervisors in charge of evaluating customers' claims; $Payment(\underline{serv\_code}, cust\_ssn, amount)$ contains information about service payments; $Service\_Provided(\underline{serv\_code}, cust\_ssn)$ holds the services which have been provided; $Premier\_Member(\underline{cust\_ssn})$ contains the customers that reach the "premier" status; $Account(\underline{acc\_id}, cust\_ssn, maximum\_withdrawal, credit\_card)$ is the relation that holds information about bank accounts.

Tasks model the possible modifications that can be performed over the artifact schema. As syntactic sugar, we include some input parameters (the symbols between brackets after the task name). In order to execute them such parameters must be instantiated with constants.

- **Claim_service**(**cust_ssn**, **serv_code**):
  - $\xi_1 = \exists x, y.\, Customer(cust\_ssn, x) \wedge Service(serv\_code, y) \rightarrow Service\_Claimed(serv\_code, cust\_ssn)$
  - $\xi_2, \ldots, \xi_9 = copy\_frame$

  models the choice of the customer $cust\_ssn$ to apply for the provision of a new service of type $serv\_code$. Since the resulting instance is a completely new one consisting in tuples added by the task only, we need to explicitely "copy" all facts that we do not require to be dropped after the task execution. That is exactly the role of effects $\xi_2, \ldots, \xi_9$ that, for all relations $R_1, \ldots, R_m$, are defined as $\xi_i = R_i(x_1, \ldots, x_n) \rightarrow R_i(x_1, \ldots, x_n)$ with $i \in \{1, \ldots m\}$. Intuitively, the result of firing task $Claim\_service(cust\_ssn,\ serv\_code)$ on an instance I results in a new instance I' that either contains I but also include the new tuple $Service\_Claimed(cust\_ssn, serv\_code)$ provided that the premise are satisfied by I, or $I' = I$ if not.
- **Make_payment**(**cust_ssn**, **serv_code**, **amount**):
  - $\xi_1 = Service\_Claimed(cust\_ssn, serv\_code) \rightarrow Payment(serv\_code, cust\_ssn, amount)$
  - $\xi_2, \ldots, \xi_9 = copy\_frame$

  models the payment operation performed by a customer for a service that has been previously requested, i.e., the resulting instance may include the tuple $Payment(cust\_ssn, serv\_code, amount)$.
- **Grant_approval**(**serv_code**):
  - $\xi_1 = Service\_Claimed(serv\_code, x) \rightarrow \exists z.\, Request\_Exam(serv\_code, z, \text{“}approved\text{”})$
  - $\xi_2, \ldots, \xi_9 = copy\_frame$

  represents the approval of a service that has been requested, by including the fact $Request\_Exam(serv\_code, ln, \text{“}approved\text{”})$ where ln is a fresh labeled null that models a possible supervisor.
- **Provide_services**():
  - $\xi_1 = Service\_Claimed(x, y) \wedge Request\_Exam(x, v, \text{“}approved\text{”}) \rightarrow Service\_Provided(x, y)$
  - $\xi_2, \ldots, \xi_9 = copy\_frame$

models the delivery of all services that have had explicitly approved by a supervisor and that was already paid. The task

- **Quick_provide_service**():
  - $\xi_1 = Service\_Claimed(x, y) \rightarrow Service\_Provided(x, y)$
  - $\xi_2, \ldots, \xi_9 = copy\_frame$

  delivers all the services for which it was paid the correct amount and that have been requested from a premier customer. Lastly
- **Award_premier_status**():
  - $\xi_1 = \exists y, u, w, t.\, Customer(x, y) \wedge Service\_Provided(y, x) \wedge Account(u, x, w, t) \rightarrow Premier\_Member(x)$
  - $\xi_2, \ldots, \xi_9 = copy\_frame$

  awards the premier status to all customers holding a bank account that applied for the provision of a service that had already been accepted.

Finally, we assume that condition-action rules that specify the **artifact lifecycle** allow for executing every task in every state, except for the following rules:

$\varrho_1 = \langle \exists x, y, u, v, w.\, Payment(x, y, w) \wedge Service(x, u) \wedge Request\_Exam(x, v, \text{“}approved\text{”}),$ **Provide_services()**$\rangle$
$\varrho_2 = \langle \exists x, y, w.\, Payment(x, y, w) \wedge Service(x, w) \wedge Premier\_Member(y),$ **Quick_provide_service()**$\rangle$
$\varrho_3 = \langle \exists x, y, u, w, t.\, Service\_Provided(x, y) \wedge Account(u, y, w, t),$ **Award_premier_status()**$\rangle$ □

2.4 Artifact Executions

Let us consider an **artifact** as a tuple $A = \langle \mathcal{S}, \mathcal{T}, \mathcal{C} \rangle$, where: *(i)* $\mathcal{S}$ is an *artifact data schema*; *(ii)* $\mathcal{T}$ is a set of *tasks*; and *(iii)* $\mathcal{C}$ is a set of *condition/action rules*.

An **artifact transition system** for A starting from a initial data instance I_0 is a tuple $\mathfrak{A}_A = \langle \Sigma, \sigma_0, L, Tr \rangle$ where *(i)* Σ is the (possibly infinite) set of states; *(ii)* σ_0 is the initial state; *(iii)* $L : \Sigma \rightarrow \mathcal{I}$ is a labeling function that associates to each state in Σ a data instance of S, with the constraint that $L(\sigma_0) = I_0$; *(iv)* $Tr \subseteq \mathcal{I} \times \mathcal{T} \times \mathcal{I}$ is the transition relation such that $\langle \sigma, T, \sigma' \rangle \in Tr$, denoted $\sigma \stackrel{T}{\Longrightarrow} \sigma'$ if there exists a rule $\varrho = \langle \pi, T \rangle$ such that $L(\sigma) \rhd \pi$, and $L(\sigma') = I'$ where I' is the result of applying task T to data instance $I = L(\sigma)$, i.e we must have that $I \stackrel{T}{\longrightarrow} I'$.

Notice that if an artifact A may generate an infinite number of data instances in its evolution, then every transition system associated to it must have and infinite number of states in order for the labeling function L to be correctly defined. Though transition systems may have more states than data instances, which implies that more states may be labelled with the same data instance. Among the various artifact transition systems for A starting from a initial data instance I_0 there is one of particular significance, the so called **execution tree** of an artifact A starting from I_0, in which each state of the transition system correspond to the full history that has generated it. Such an execution tree is a transition system $\mathfrak{T}_A = \langle \Sigma, \sigma_0, L, Tr \rangle$ whose set of states is defined as follows: *(i)* the root is σ_0; *(iii)* given a state σ for each task $T \in \mathcal{T}$ such that there exists a rule $\varrho = \langle \pi, T \rangle$ such that $L(\sigma) \rhd \pi$, add a state σ'_T, and define $L(\sigma'_T) = I'$ where I' is data

instance resulting by applying T to $L(\sigma)$. We can interpret σ'_T as the T-successor of the node σ.

We observe that the number of states of the execution tree is indeed *infinite*, and also that given any state σ, by looking at the path from the root σ_0 to σ, we can reconstruct the full history that has lead to σ, including the sequence of tasks invoked and the resulting data instance at each step.

All transition systems for and artifact A starting from a given data instance I_0, even if different, denote the same behavior, namely the behavior of the artifact A starting from I_0 and executing the various tasks. To formally capture such and equivalence between transition system, we make use of the notion of *bisimulation* [20]. In fact in formally detailing such a notion, we consider right from the start that the user can only query data instances through conjunctive queries, evaluated to return certain answers.

Given two artifact transition systems $\mathfrak{A}_1 = \langle \Sigma_1, \sigma_{0,1}, L_1, Tr_1 \rangle$ and $\mathfrak{A}_2 = \langle \Sigma_2, \sigma_{0,2}, L_2, Tr_2 \rangle$ a **bisimulation** is a relation $B \subseteq \Sigma_1 \times \Sigma_2$ such that:

$\langle \sigma_1, \sigma_2 \rangle \in B$ implies that:

1. for every conjunctive query cq we have that $cert^{L_1(\sigma_1)}(cq) = cert^{L_2(\sigma_2)}(cq)$;
2. if $\sigma_1 \stackrel{a}{\Longrightarrow} \sigma'_1$ then there exists σ'_2 such that $\sigma_2 \stackrel{a}{\Longrightarrow} \sigma'_2$ and $\langle \sigma'_1, \sigma'_2 \rangle \in B$;
3. if $\sigma_2 \stackrel{a}{\Longrightarrow} \sigma'_2$ then there exists σ'_1 such that $\sigma_1 \stackrel{a}{\Longrightarrow} \sigma'_1$ and $\langle \sigma'_1, \sigma'_2 \rangle \in B$.

We say that two states σ_1 and σ_2 are *bisimilar*, denoted as $\sigma_1 \sim \sigma_2$ if there exists a bisimulation B such that $\langle \sigma_1, \sigma_2 \rangle \in B$. Two transition systems $\mathfrak{A}_1 = \langle \Sigma_1, \sigma_{0,1}, L_1, Tr_1 \rangle$ and $\mathfrak{A}_2 = \langle \Sigma_2, \sigma_{0,2}, L_2, Tr_2 \rangle$ are **bisimilar** if $\sigma_{0,1} \sim \sigma_{0,2}$. We are now able to introduce the verification formalism.

3 Verification Formalism

We turn to verification of conjunctive artifact-centric services. To specify dynamic properties we will use μ-calculus [14] which is one of the most powerful temporal logics for which model checking has been investigated, and indeed is able to express both linear time logics, as LTL, and branching time logics such as CTL or CTL* [10]. In particular, we need to introduce a variant of μ-calculus, called $\mu\mathcal{L}$ that conforms with the basic assumption of our formalism: the use of conjunctive queries and certain answers to talk about data instances. This intuitive requirement can be made formal as follows: our μ-calculus variant must by invariant with respect to the notion of bisimulation introduced above.

Given an artifact $A = \langle \mathcal{S}, \mathcal{T}, \mathcal{C} \rangle$, the **verification formulas** of $\mu\mathcal{L}$ for A have the following form:

$$\Phi ::= cq \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid [T]\Phi \mid \langle T \rangle \Phi \mid \mu Z.\Phi \mid \nu Z.\Phi \mid Z$$

where cq is a boolean conjunctive query (interpreted through certain answers) over the artifact schema, Z is a predicate variable symbol.

The symbols μ and ν can be considered as quantifiers, and we make use of notions of scope, bound and free occurrences of variables, closed formulas, etc. The definitions

of these notions are the same as in first-order logic, treating μ and ν as quantifiers. In fact, we are interested only in closed formulas as specification 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 μ-calculus, given the requirement of syntactic monotonicity, the least fixpoint $\mu Z.\Phi$ and the greatest fixpoint $\nu Z.\Phi$ always exist. In order to define the meaning of such formulas we resort to transition systems. Let $\mathfrak{A}_A = \langle \Sigma, \sigma_0, L, Tr \rangle$ be a transition system for A with initial data instance I_0, and let $\mathcal{V}$ be predicate valuation on $\mathfrak{A}$, i.e., a mapping from the predicate variables to subsets of the states in $\mathfrak{A}$. Then, we assign meaning to μ-calculus formulas by associating to $\mathfrak{A}$ and $\mathcal{V}$ an *extension function* $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$, which maps μ-calculus formulas to subsets of $\mathcal{I}$. The extension function $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$ is defined inductively as follows:

$$
\begin{array}{ll}
(cq)^{\mathfrak{A}}_{\mathcal{V}} & = \{\sigma \in \Sigma \mid cert^{L(\sigma)}(cq)\} \\
(Z)^{\mathfrak{A}}_{\mathcal{V}} & = \mathcal{V}(Z) \subseteq \Sigma \\
(\neg\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \Sigma - (\Phi)^{\mathfrak{A}}_{\mathcal{V}} \\
(\Phi_1 \wedge \Phi_2)^{\mathfrak{A}}_{\mathcal{V}} & = (\Phi_1)^{\mathfrak{A}}_{\mathcal{V}} \cap (\Phi_2)^{\mathfrak{A}}_{\mathcal{V}} \\
(\langle T \rangle \Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \{\sigma \in \Sigma \mid \exists \sigma'.\ \sigma \stackrel{T}{\Longrightarrow} \sigma' \text{ and } \sigma' \in (\Phi)^{\mathfrak{A}}_{\mathcal{V}}\} \\
([T]\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \{\sigma \in \Sigma \mid \forall \sigma'.\ \sigma \stackrel{T}{\Longrightarrow} \sigma' \text{ implies } \sigma' \in (\Phi)^{\mathfrak{A}}_{\mathcal{V}}\} \\
(\mu Z.\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \bigcap\{\mathcal{E} \subseteq \Sigma \mid \ (\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z \leftarrow \mathcal{E}]} \subseteq \mathcal{E}\ \} \\
(\nu Z.\Phi)^{\mathfrak{A}}_{\mathcal{V}} & = \bigcup\{\mathcal{E} \subseteq \Sigma \mid \ \mathcal{E} \subseteq \ (\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z \leftarrow \mathcal{E}]}\}
\end{array}
$$

Intuitively, the extension function $(\cdot)^{\mathfrak{A}}_{\mathcal{V}}$ assigns to the various constructs of μ-calculus the following meanings:

- The boolean connectives have the expected meaning.
- The extension of $\langle T \rangle \Phi$ includes the states σ such that starting from σ, there is an execution of task T that leads to a successive state σ' included in the extension of Φ.
- The extension of $[T]\Phi$ includes the states σ such that starting from σ, each execution of task T leads to some successive state σ' included in the extension of Φ.
- The extension of $\mu Z.\Phi$ is the *smallest subset* $\mathcal{E}_\mu$ of Σ such that, assigning to Z the extension $\mathcal{E}_\mu$, the resulting extension of Φ is contained in $\mathcal{E}_\mu$. That is, the extension of $\mu X.\Phi$ is the *least fixpoint* of the operator $\lambda\mathcal{E}.(\Phi)^{\mathfrak{A}}_{\mathcal{V}[Z \leftarrow \mathcal{E}]}$ (here $\mathcal{V}[Z \leftarrow \mathcal{E}]$ denotes the predicate valuation obtained from $\mathcal{V}$ by forcing the valuation of Z to be $\mathcal{E}$).
- Similarly, the extension of $\nu X.\Phi$ is the *greatest subset* $\mathcal{E}_\nu$ of Σ such that, assigning to X the extension $\mathcal{E}_\nu$, the resulting extension of Φ contains $\mathcal{E}_\nu$. That is, the extension of $\nu X.\Phi$ is the *greatest fixpoint* of the operator $\lambda\mathcal{E}.(\Phi)^{\mathfrak{A}}_{\mathcal{V}[X \leftarrow \mathcal{E}]}$.

The reasoning problem we are interested in is **model checking**: verify whether a $\mu\mathcal{L}$ **closed formula** Φ **holds in an artifact** A **with initial data instance** I_0. Formally such problem is defined as checking whether $\sigma_0 \in I \in (\Phi)^{\mathfrak{T}^{I_0}_A}_{\mathcal{V}}$ (where $\mathcal{V}$ is any valuation, since Φ is closed), that is, whether Φ is true in the root of the A execution tree.

On the other hand we know that there are severals transition system that are bisimilar to the execution tree $\mathfrak{T}^{I_0}_A$. The following theorem state that the formula evaluation in $\mu\mathcal{L}$ is indeed invariant wrt bisimilarity, so we can equivalently check any such transition systems.

Theorem 1. *Let $\mathfrak{A}_1$ and $\mathfrak{A}_2$ be two bisimilar artifact transition systems. Then, for every pair of states σ_1 and σ_2 such that $\sigma_1 \sim \sigma_2$ (including the initial ones), for all formulas Φ of $\mu\mathcal{L}$, we have that $\sigma_{0,1} \in (\Phi)_{\mathcal{V}}^{\mathfrak{A}_1}$ iff $\sigma_{0,2} \in (\Phi)_{\mathcal{V}}^{\mathfrak{A}_1}$.*

Proof. The proof is analogous to the standard proof of bisimulation invariance of mu-calculus, see e.g., [7], though taking into account our specific definition of bisimulation, which makes use of conjunctive queries and certain answers as their evaluation.

In particular if, for some reason we can get a transition system that is bisimilar to the execution tree, and is *finite*, then we can apply the following theorem.

Theorem 2. *Checking a $\mu\mathcal{L}$ formula Φ over a finite transition system $\mathfrak{A}_A = \langle \Sigma, \sigma_0, L, Tr \rangle$ can be done in time*

$$O((|\mathfrak{A}| \cdot |\Phi|)^k)$$

where $|\mathfrak{A}| = |\Sigma| + |Tr|$, i.e., the number of states plus the number of transitions of $\mathfrak{A}$, $|\Phi|$ is the size of formula Φ (in fact, considering conjunctive queries as atomic), and k is the number of nested fixpoints, i.e., fixpoints whose variables are one within the scope of the other.

Proof. We can use the standard μ-calculus model checking algorithms [13], with the proviso that for atomic formulas we use the computation of certain answers of conjunctive queries.

Example 2. Continuing the example introduced above, suppose now we have an initial artifact data instance where: $Customer^{I_0} = \{\langle 337505, \text{“}JohnSmith\text{”}\rangle, \langle 125232, \text{“}MaryStewart\text{”}\rangle\}$, and $Service^{I_0} = \{\langle L057, 100\rangle, \langle L113, 150\rangle, \langle C002, 50\rangle\}$, and all the other relations are empty. Consider the following liveness property, which aks if it is actually possible to obtain the provision of a service:

$$\mu Z.\ ((\exists\, x_1, x_2, x_3. Service(x_1, x_2)\ \wedge\ Service\_Provided(x_1, x_3))\ \vee\ \bigvee_{T \in \mathcal{T}} \langle T \rangle Z$$

The formula is actually true: for example, a state in which $Service\_Provided(L057, 337505)$ holds can be reached after the following sequence of tasks: $Claim\_Service(337505, L057)$, $Make\_Payment(337505, L057, 100)$, $Grant\_Approval(L057)$ and finally $Provide\_Services()$. Next consider the safety property asking whether every possible reachable instance will always contain the information that the service $L113$ has been paid and provided:

$$\nu Z.\ (\exists\, x_1, x_2, x_3. Payment(L113, x_1, x_2) \wedge\ Service\_Provided(L113, x_3)\ \wedge\ \bigwedge_{T \in \mathcal{T}} [T](Z))$$

that is trivially false, since in the initial state I there is no payment for any service. More sophisticated properties such as strong form of fairness for example are also easily expressible in $\mu\mathcal{L}$, though for space limitation we don't report them here $\square$

4 Results

Notice that we still do not have a concrete technique for the verification problem, since model checking results in Theorem 2 only apply to finite structures. In fact, as a consequence of the undecidability of the implication problem for tgds (see e.g. [2]), it is obvious that, without any restrictions on effect specfications, the model checking in our setting is undecidable. Addressing condition of decidability is the purpose of this section. We start by introducing the notion of Skolem transition system and showing its relationship with the concept of execution tree of an artifact.

4.1 Skolem Transition System

For every effect specification $\xi = \exists \boldsymbol{y}\ \phi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c}) \rightarrow \exists \boldsymbol{w}\ \psi(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{d})$ and for every $w \in \boldsymbol{w}$ we define a Skolem term $f^{\xi}_{w}(\boldsymbol{x})$. Such Skolem term is interpreted as a fixed injective function $f^{\xi}_{w} : \Delta \rightarrow ln(\Delta)$. In this way, enacting an effect ξ on a the data instance I results in the set of facts $R^{I'}_{i}(\boldsymbol{x}, f^{\xi}_{w_1}(\boldsymbol{x}), \ldots, f^{\xi}_{w_n}(\boldsymbol{x}), \boldsymbol{d})|^{\psi}_{\eta_i}$, for every for every atom $R^{I'}_{i}(\boldsymbol{x}, w_1, \ldots, w_n), \boldsymbol{d})$ that occurs in ψ and every answer to the left-hand-side query $\eta_i \in (\exists \boldsymbol{y}.\phi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c})^{I})$. The **Skolem execution of a task** T in I is the data instance J formed by the union of all Skolem enactment of the effects in T. Notice that being the interpretation of Skolem terms an fixed function, the Skolem execution of a task is fully determined and functional.

Given an artifact $A = (\mathcal{S}, \mathcal{T}, \mathcal{C})$ and an initial artifact data instance I_0, we define the **Skolem transition system** $\mathfrak{S}_A = \langle \Sigma_s, \sigma_{0,s}, L_s, Tr_s \rangle$ inductively as follows:

- $\sigma_{0,s} \in \Sigma$ and such that $L_s(\sigma_{0,s}) = I_0$;
- for all instances $\sigma \in \Sigma_s$ and for each task $T \in \mathcal{T}$ such that there exists a rule $\varrho = \langle \pi, T \rangle$ such that $L_s(\sigma) \rhd \pi$, let J be data instance resulting from the Skolem execution of task T in $L_s(\sigma)$ then:
  - if there exists an instance $\sigma' \in \Sigma_s$ such that $L_s(\sigma') \overset{h}{=} J$ then add the transition $\sigma \overset{T}{\Longrightarrow} \sigma'$ to Tr_s;
  - if such a state does not exists, then add the a new state σ_J to Σ with $L_s(\sigma_J) = J$ to $\mathcal{I}$ and add the transition edge $\sigma \overset{T}{\Longrightarrow} \sigma_J$ to Tr_s.

Theorem 3. *Let $A = (\mathcal{S}, \mathcal{T}, \mathcal{C})$ be an artifact and I_0 be a data instance over schema $\mathcal{S}$. Then, the execution tree $\mathfrak{T}_{A,I_o} = \langle \Sigma_t, \sigma_{0,t}, L_t, Tr_t \rangle$ is bisimilar to the Skolem transition system $\mathfrak{S}_{A,I_o} = \langle \Sigma_s, \sigma_{0,s}, L_s, Tr_s \rangle$.*

Proof. Let us consider the bisimulation relation $B_{ts} = \{\langle \sigma_t, \sigma_s \rangle \mid \sigma_t \in \Sigma_t \wedge \sigma_s \in \Sigma_s \wedge L_t(\sigma_s) \overset{h}{=} L_t(\sigma_t)\}$. This is the relation formed by the pris of states of the two transition system such that their labeling data instances are homomorphically equivalent. We show that B_{ts} is bisimulation (according to our definition). Indeed consider $\langle \sigma_s, \sigma_t \rangle \in B_{ts}$. Then:

1. For each cq, since $L_t(\sigma_s) \overset{h}{=} L_t(\sigma_t)$ we have that $cert^{L_t(\sigma_s)}(cq) = cert^{L_t(\sigma_s)}(cq)$ from the definition of certain answers and homomorphical equivalence.
2. If $\sigma_t \overset{a}{\Longrightarrow} \sigma'_t$ then there is a rule $\varrho = \langle \pi, T \rangle$ and $L_t(\sigma_t) \rhd \pi$. Since $L_t(\sigma_s) \overset{h}{=} L_t(\sigma_t)$ then *(i)* $L_s(\sigma_s) \rhd \pi$ as well, so $\sigma_s \overset{a}{\Longrightarrow} \sigma'_s$ moreover it is easy to see that $L_t(\sigma'_t) \overset{h}{=} L_s(\sigma'_s)$ by considering definition of executing a task and a Skolem executing task.

3. Symmetric to the previous case.

Finally observe that since $L_t(\sigma_{0,t}) = L_s(\sigma_{0,s}) = I_0$ we trivially get that $\langle \sigma_{0,s}, \sigma_{O,t} \rangle \in B_{ts}$.

This theorem basically allow us to make use of a Skolem transition system rather than an execution tree for our verification tasks, taking advantage of Theorem 1. using *equivalence classes* of homomorphically equivalent instances for the purpose of verification. Notice, however, that this theorem it is not sufficient to achieve a decidability result, since the number of state in the Skolem transition system is bounded only by the number of homomorphically non-equivalent data instances, which is infinite in general. Next we concentrate on conditions that guarantee its finiteness.

4.2 Decidability

Given an artifact $A = \langle \mathcal{S}, \mathcal{T}, \mathcal{C} \rangle$ and the set $\mathcal{I}$ of possible interpretations over $\mathcal{S}$, we consider two different functions: the first one, $\mathsf{f} : \mathcal{T} \times \mathcal{I} \to \mathcal{I}$, is the usual result of Skolem executing a task on I; while the second one, $\mathsf{g} : \mathcal{T} \times \mathcal{I} \to \mathcal{I}$, is the *inflationary* variant of the first one: $\mathsf{g}(T, I) = \mathsf{f}(T, I) \cup I$, that is g generates the result of Skolem executing the task T on I and then copies all "old" facts of I. Notice that no contradiction can arise since effects of tasks, being based on conjunctive queries, are only positive. For f and g we have the following results:

Lemma 1. *Functions* g *and* f *are* monotonic *wrt set containment. Namely, for every task T and instance I, if $I \subseteq J$, we have both $\mathsf{f}(T, I) \subseteq \mathsf{f}(T, J)$ and $\mathsf{g}(T, I) \subseteq \mathsf{g}(T, J)$.*

Lemma 2. *Function* g *is* monotonically increasing*, namely for every task T and instance I, $I \subseteq \mathsf{g}(T, I)$ holds.*

The same result does not hold for function f, because some facts may not be propagated.

Lemma 3. *For every task T and instance I, we have that $\mathsf{f}(T, I) \subseteq \mathsf{g}(T, I)$.*

Let us inductively define the set of instances $\mathcal{L}_{I_0}$ obtained, starting from I_0, by repeatedly applying $\mathsf{g}(\cdot, \cdot)$ in all possible ways. This is the least set such that

- $I_0 \subseteq \mathcal{L}_{I_0}$;
- if $I' \subseteq \mathcal{L}_{I_0}$ then, for every $T \in \mathcal{T}$, $\mathsf{g}(T, I') \subseteq \mathcal{L}_{I_0}$.

Notice also that, as an immediate consequence of its inductive definition, we get $\mathsf{g}(T, \mathcal{L}_{I_0}) = \mathcal{L}_{I_0}$, since $\mathcal{L}_{I_0}$ is a *fixpoint*, indeed, the *least fixpoint* [26].

Lemma 4. *Let I_0 be an instance and $\mathcal{L}_{I_0}$ as above, then for every sequence of instances $I_0, \ldots, I_n$ such that $I_{i+1} = \mathsf{g}(T_i, I_i)$, we have that $I_i \subseteq \mathcal{L}_{I_0}$, for $i = 0, \ldots, n$.*

Proof. By induction of length n of a task sequence.

Lemma 5. *Let $A = \langle \mathcal{S}, \mathcal{T}, \mathcal{C} \rangle$ be an artifact, I_0 and initial data instance, and $\mathcal{L}_{I_0}$ as above. Then for every sequence of instances $I_0, \ldots, I_n$, such that $I_{i+1} = \mathsf{f}(T_i, I_i)$, we have that $I_i \subseteq \mathcal{L}_{I_0}$, for $i = 0, \ldots, n$.*

Proof. By Lemma 3 and 4.

Roughly speaking, the above lemmas, guarantee that every possible instance that can be produced from I_0 by applying in every possible way both f and g functions is bounded by the least fixpoint $\mathcal{L}_{I_0}$. Notice however that $\mathcal{L}_{I_0}$ is infinite in general, in order to get decidability we will still need a finite bound on $\mathcal{L}_{I_0}$.

To get such condition we exploit results from [15] on *weakly-acyclic* tgds. Weak-acyclicity is a syntactic notion that involves the so-called *dependency graph* of the set of tgds TG. Informally, a set TG of tgds is weakly-acyclic if there are no cycles in the dependency graph of TG involving "existential" relation positions. The key property of *weakly-acyclic* tgds is that chasing a data instance with them (i.e., applying them in all possible way) generates a stet of facts (a database) that is finite. We refer to [15] for more details. We show that, under the assumption that the tgds of the artifact are weakly-acyclic, the set $\mathcal{L}_{I_0}$ introduced above is *finite*.

Lemma 6. *Let $A = \langle \mathcal{S}, \mathcal{T}, \mathcal{C} \rangle$ be an artifact. If all effect specifications in every task $T \in \mathcal{T}$ are weakly-acyclic, then the fixpoint $\mathcal{L}_{I_0}$ has finite cardinality.*

Proof. If the set of all effect specification is weakly-acyclic, from [15] we know that the *dependency graph* has no cycle going through a special edge [15]. Since every special edge represents the application of a Skolem function, it follows that for every Skolem execution task sequence, it is not possible to nest the same Skolem function. Indeed, suppose that at a certain point, an effect specification $\xi = \exists \boldsymbol{y}.\phi(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c}) \rightarrow \exists \boldsymbol{w}.\psi(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{d})$ adds a fact $R_i^I(\boldsymbol{x}, f_{w_1}^{\xi}(\boldsymbol{x}), \ldots, f_{w_i}^{\xi}(f_{w_j}^{\xi'}(\ldots(f_{w_i}^{\xi}(\boldsymbol{x})), \ldots, f_{w_n}^{\xi}(\boldsymbol{x}), \boldsymbol{d})|_{\eta_i}^{\psi}$, this means that: *(i)* there is at least a special edge from a position p_1 in a relation R_j that occurs in ϕ, to a position p_2 in R_i that occurs in ψ, due to the presence of the outermost Skolem function $f_{w_i}^{\xi}$, *(ii)* there is a sequence (eventually empty) of special edges that propagate values in position p_2 to position p_m, due to the presence of Skolem function between the outermost and the innermost one, and *(iii)* there is a sequence (of length at least one) of non-special edges that propagate values from p_m back to p_1, because the innermost Skolem function $f_{w_i}^{\xi}$ is nested in itself (the outermost). But this contradicts the hypothesis of weakly-acyclic set of effect specification. Since the domain and the image of a Skolem function is finite, and no nesting of the same Skolem function is possible, there is a bound on the number of different values that can exist in every position of the schema. As a consequence, the number of possible instances that can be obtained from a (finite) initial instance I_0 by applying g in every possible way is finite. Given that g is monotonic, the theorem is proved.

Based on the above theorem, we are able to derive our main result.

Theorem 4. *Let $A = \langle \mathcal{S}, \mathcal{T}, \mathcal{C} \rangle$ be an artifact such that all effect specifications in $\mathcal{T}$ are weakly-acyclic, and let I_0 be a data instance for A . Then, for every formula Φ of $\mu\mathcal{L}$, verifying that Φ holds in A with initial data instance I_0 is decidable.*

Proof. By Theorem 3 and Theorem 1, we can perform model checking of Φ on the Skolem transition system for A and I_0. Now, by Lemma 5, we have that all data instances that can be assigned to the states of the Skolem transition system for A and I_0

must be subsets of $\mathcal{L}_{I_0}$. And by Lemma 6, we get that $\mathcal{L}_{I_0}$ has a finite cardinality. This implies that Skolem transition system is finite and Theorem 2 can be applied.

5 Conclusions

In this paper we have introduced conjunctive artifact-based services, a class of services which pose balanced attention to both data (here a full-fledged relational database) and processes (acting on the database), and, through a suitable use of conjunctive queries in specifying tasks pre- and post-conditions, guarantees decidability.

It is worth noting that decidability results for formalisms that fully take into account both data and processes are rare. Here we mention three of them that are quite relevant for artifact-centric approaches. The most closely related one is [12]), which shares the general setting with our approach but differs in the conditions required to obtain decidability. These are not based on conjunctive queries, but on some decidability results of certain formulas of a first-order variant of linear time temporal logic [25]. Another relevant decidability result is that of SPOCUS relational transducers [3], where decidability is obtained through results on inflationary Datalog. Finally, the work on service composition according to the COLOMBO model [4] is also related. There, decidability is obtained through symbolic abstraction on data and the requirement that process are input bounded (i.e., take only a bounded number of new values (similar to our nulls) taken from input). The result presented here is not subsumed by (nor subsumes) any of the above results. But actually opens a new lode for research in the area, based on the connection with the theory of dependencies in databases that has been so fruitful in data exchange and data integration in recent years [15,18].

Acknowledgments. The authors would like to thank Diego Calvanese and Yves Lesperance for interesting discussions on the paper. This work has been supported by the EU Project FP7-ICT ACSI (257593).

References

1. Abiteboul, S., Bourhis, P., Galland, A., Marinoiu, B.: The axml artifact model. In: TIME, pp. 11–17 (2009)
2. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
3. Abiteboul, S., Vianu, V., Fordham, B.S., Yesha, Y.: Relational transducers for electronic commerce. J. Comput. Syst. Sci. 61(2), 236–269 (2000)
4. Berardi, D., Calvanese, D., De Giacomo, G., Hull, R., Mecella, M.: Automatic Composition of Transition-based Semantic Web Services with Messaging. In: Proc. of VLDB 2005 (2005)
5. Bhattacharya, K., Gerede, C.E., 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)
6. Bhattacharya, K., Guttman, R., Lyman, K., Heath III, 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)

7. Bradfield, J., Stirling, C.: Modal mu-calculi. In: Handbook of Modal Logic, vol. 3, pp. 721–756. Elsevier, Amsterdam (2007)
8. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: STOC, pp. 77–90 (1977)
9. Clark, K.L.: Negation as failure. In: Logic and Data Bases, pp. 293–322 (1977)
10. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. The MIT Press, Cambridge (1999)
11. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. 32(3), 3–9 (2009)
12. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic Verification of Data-Centric Business Processes. In: Proc. of ICDT (2009)
13. Emerson, E.A.: Model checking and the mu-calculus. In: Descriptive Complexity and Finite Models, pp. 185–214 (1996)
14. Emerson, E.A.: Model checking and the mu-calculus. In: Descriptive Complexity and Finite Models, pp. 185–214 (1996)
15. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theor. Comput. Sci. 336(1), 89–124 (2005)
16. Fritz, C., Hull, R., Su, J.: Automatic construction of simple artifact-based business processes. In: ICDT, pp. 225–238 (2009)
17. Hull, R.: Artifact-centric business process models: Brief survey of research results and challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008. LNCS, vol. 5331, pp. 1152–1163. Springer, Heidelberg (2008)
18. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: Proc. of PODS 2002, pp. 233–246 (2002)
19. Luckham, D.C., Park, D.M.R., Paterson, M.: On formalised computer programs. J. Comput. Syst. Sci. 4(3), 220–249 (1970)
20. Milner, R.: An algebraic definition of simulation between programs. In: Proc. of IJCAI, pp. 481–489 (1971)
21. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Syst. J. 42(3), 428–445 (2003)
22. Park, D.: Finiteness is mu-ineffable. Theor. Comput. Sci. 3(2), 173–181 (1976)
23. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press, Cambridge (September 2001)
24. Sohrabi, S., Prokoshyna, N., McIlraith, S.A.: Web service composition via generic procedures and customizing user preferences. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 597–611. Springer, Heidelberg (2006)
25. Spielmann, M.: Verification of relational transducers for electronic commerce. J. Comput. Syst. Sci. 66(1), 40–65 (2003)
26. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific J. of Mathematics 5(2), 285–309 (1955)
27. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A framework for lightweight interacting workflow processes. Int. J. Cooperative Inf. Syst. 10(4), 443–481 (2001)

Cangialosi, P., De Giacomo, G., De Masellis, R., and Rosati, R. Conjunctive artifact-centric services. In *Proc. of the 8th Int. Joint Conf. on Service Oriented Computing (ICSOC 2010)*, vol. 6470 of *Lecture Notes in Computer Science*, pp. 318–333. Springer (2010). doi:10.1007/978-3-642-17358-5_22.


```
@inproceedings{CDDR10,
Author = {Piero Cangialosi and De Giacomo, Giuseppe and De Masellis, Riccardo and Riccardo Rosati},
Booktitle = ICSOC-10,
Pages = {318--333},
Publisher = SPRINGER,
Series = LNCS,
Title = {Conjunctive Artifact-Centric Services},
Volume = 6470,
Year = 2010,
doi = 10.1007/978-3-642-17358-5_22}
```