Verification of Deployed Artifact Systems via Data Abstraction

Francesco Belardinelli and Alessio Lomuscio and Fabio Patrizi

Department of Computing
Imperial College London, UK

Abstract. Artifact systems are a novel paradigm for specifying and implementing business processes described in terms of interacting modules called *artifacts*. Artifacts consist of *data* and *lifecycle* models, accounting for the relational structure of the artifact state and its possible evolutions over time. We consider the problem of verifying artifact systems against specifications expressed in quantified temporal logic. This problem is in general undecidable. However, when artifact systems are deployed, their states can contain only a bounded number of elements. We exploit this fact to develop an abstraction technique that enables us to verify deployed artifact systems by model checking their bounded abstraction.

1 Introduction

Artifact systems (AS) are a recent paradigm in business process modelling and development [9]. Artifacts are a response to criticisms [4] in the workflow and services literature regarding the fact that services are typically represented and reasoned about by mostly neglecting the data involved in the system. In artifact systems the underlying databases on which services operate are treated with similar emphasis to the processes operating on them. While this responds to the need of more accurate and natural modelling, it results in significant difficulties from a theoretical point of view.

A considerable problem arises when trying to verify formally services specified using artifacts. While verification through model checking and its applications (including orchestration and choreography) are relatively well-understood in the plain process-based modelling, the presence of data makes these problems much harder. Put succinctly, infinite domains in the underlying databases lead to infinite state-spaces for the model checking problem. Infinite domains also call for specification languages that support quantification, e.g., first-order temporal logic. This setting is known to generate an undecidable model checking problem [12], thereby reducing, at least theoretically, the possibility of verifying ASs in their most general setting.

The starting point for the work presented in this paper is that any AS, when deployed, can operationally have only size-bounded (though infinitely many) underlying database states, due to the finiteness of the machine running the system. It follows that any verification problem for ASs deployed on concrete machines can be abstracted by considering bounded abstractions of the theoretically infinite domains. This bound can either be obtained from the size of the memory of the machine running the system, or

can be iteratively refined to generate finer abstractions of the concrete system. In the paper we show that under these assumptions the verification problem for ASs is decidable and its complexity, while high, is not vastly dissimilar from that of other applications.

The rest of the paper is organised as follows. In Section 2 we introduce database schemas and a formalisation of AS suitable for verification. In Section 3 we introduce the verification problem in its general form and point to its undecidability. Section 4 illustrates the formalisation through a use-case thereby validating the formal machinery. Section 5 introduces the notion of deployed artifacts and bounded AS, and presents abstraction results leading to the decidability of the verification problem. We conclude by applying the results to the scenario discussed and pointing to further work.

Related Work.

Although different approaches to infinite-state model checking have been proposed [15, 7, 6], we are not aware of results addressing properties that involve the relational structure of the data in each state.

Our approach is largely inspired by [11] and [10], where decidability is achieved by adding syntactic restrictions on both the system description and the specification to verify. These impose a form of *guarded quantification* on variables and limit the access to the values stored as the system evolves. Our work differs from these in that we do not allow quantifiers to occur out of the scope of modal operators in the property to verify; we do not impose any restriction on the system specification; we verify an infinite fragment of the original system; and we consider a branching-time specification language. We do not perceive exploring a fragment of the original system as a limitation, as the explored fragment is in fact *the* deployed system. Moreover, our technique enables a form of (incomplete) verification based on generating successively more refined abstractions of the original system.

Our abstraction technique is based on replacing the actual values of the concrete system with a finite set of *symbolic* values. This approach was previously adopted in [3] in the context of service composition.

Our work is also related to [1], which addresses the orthogonal problem of checking whether an AS introduces only a finite number of new values. The conditions put forward in this work guarantee the underlying model to be finite-state, which is a tighter restriction than in our case. While we focus on properties specified in a first-order extension of CTL, [1] considers a quantified version of the μ-calculus. Although not formally addressed here, we expect our technique to be able to deal with μ-calculus properties, too, thus making our framework a generalisation of [1].

2 Artifacts and Artifact Systems

Broadly speaking, *artifacts* are abstract models of the atomic entities that, by mutually interacting, give rise to a *business process* [9]. They are structures consisting of two parts: a *data model* and a *lifecycle model*. The former captures a fragment of the structure of the information relevant to the process. The latter is a specification of the possible ways such information evolves in response to external or internal events, and

how new events for other artifacts are generated. For our purposes, artifacts can be simply thought of as possibly nested records (i.e., they may contain sets of records as attribute values), equipped with actions that enable changes on their attributes. A characterizing feature of artifacts is the presence of an id and some status attributes. The id field identifies a particular artifact instance, the status fields encode the advancement of the artifact with respect to its lifecycle. Adopting an approach similar to [1], in this paper we formalise artifact systems by means of a database and a set of actions, which account for the artifact data models and the artifact lifecycles, respectively.

Definition 1 (Database schema). *A* (relational) database schema *is a set* $\mathcal{D} = \{P_1/a_1, \ldots, P_n/a_n\}$ *of* relation (or predicate) symbols P_i*, each associated with its arity* $a_i \in \mathbb{N}$.

Definition 2 (Database interpretation). *Given a database schema* $\mathcal{D}$*, a* $\mathcal{D}$-interpretation (or $\mathcal{D}$-instance) over an interpretation domain U *is a mapping* D *associating each relation symbol* P_i *with a* finite a_i*-ary relation over* U*, i.e.,* $D(P_i) \subseteq U^{a_i}$.

The set of all $\mathcal{D}$-interpretations over a given domain U is denoted by $\mathcal{I}_{\mathcal{D}}(U)$. The *active domain* of D, $adom(D)$, is the set of all U-elements occurring in some tuple of some predicate interpretation $D(P_i)$. By definition the active domain $adom(D)$ is finite.

Definition 3 (First-order formulas over $\mathcal{D}$, U, and V). *Given a database schema* $\mathcal{D} = \{P_1/a_1, \ldots, P_n/a_n\}$*, and two sets* U *and* V *of constant and variable symbols, respectively, the language* $\mathcal{L}_{\mathcal{D},U,V}$ *of first-order formulas* φ *over* $\mathcal{D}$*,* U *and* V *is inductively defined as follows:*

$$\varphi ::= t = t' \mid P_i(\boldsymbol{t}) \mid (\varphi) \mid \neg\varphi \mid \forall x\varphi \mid \varphi \rightarrow \varphi,$$

where $\boldsymbol{t}$ *is an* a_i*-tuple of* terms*, and* t, t' *are* terms*, i.e., elements in* $U \cup V$.

In the rest of the paper we assume V fixed, thus omitting the corresponding subscript. When no ambiguity arise, we also omit U. We use the standard abbreviations $\exists$, $\top$, $\bot$, $\wedge$, $\vee$, and $\neq$. Free and bound variables are defined as standard. For a formula φ we denote the set of its variables as $vars(\varphi)$, the set of its free variables as $free(\varphi)$, and the set of constants occurring in φ as $const(\varphi)$. We write $\varphi(\boldsymbol{x})$ with $\boldsymbol{x} = \langle x_1, \ldots, x_\ell \rangle$ to list explicitly in arbitrary order all the free variables in φ. By slight abuse of notation, we treat $\boldsymbol{x}$ as a set, thus we write $\boldsymbol{x} = free(\varphi)$. A *sentence* is a formula with no free variables. Given a FO-formula $\varphi(\boldsymbol{x})$, and a list of terms $\boldsymbol{t}$ s.t. $|\boldsymbol{x}| = |\boldsymbol{t}| = \ell$, $\varphi(\boldsymbol{t})$ represents the formula obtained from φ by replacing every occurrence of the i-th element in $\boldsymbol{x}$ with the i-th element of $\boldsymbol{t}$, for $i = 1, \ldots, \ell$. Obviously, if $\boldsymbol{t}$ contains only constants, $\varphi(\boldsymbol{t})$ is a sentence. A U*-assignment* is a total function $\sigma : V \mapsto U$. For technical convenience, we implicitly assume that every U-assignment σ is extended to the whole U and is the identity on it, i.e., $\forall u \in U, \sigma(u) = u$. Given an assignment σ, we denote by σ_u^x the U-assignment s.t. $\sigma_u^x(x) \doteq u$ and $\sigma_u^x(x') \doteq \sigma(x')$, for every $x' \in V$ s.t. $x' \neq x$.

Definition 4 (Active-Domain Semantics of FO-formulas). *Given a database schema* $\mathcal{D}$*, an interpretation domain* U*, a* $\mathcal{D}$*-interpretation* D *over* U*, a* U*-assignment* σ*, and a FO-formula* $\varphi \in \mathcal{L}_{\mathcal{D},U}$ *over* $\mathcal{D}$ *and* U *(with* V *being fixed), we inductively define whether* D *satisfies* φ *under* σ*, written* $(D, \sigma) \models \varphi$*, as follows:*

$(D, \sigma) \models t = t'$ *iff* $\sigma(t) = \sigma(t')$*;*
$(D, \sigma) \models P_i(t_1, \ldots, t_\ell)$ *iff* $\langle \sigma(t_1), \ldots, \sigma(t_\ell) \rangle \in D(P_i)$*;*
$(D, \sigma) \models (\varphi)$ *iff* $(D, \sigma) \models \varphi$*;*
$(D, \sigma) \models \neg\varphi$ *iff* $(D, \sigma) \not\models \varphi$*;*
$(D, \sigma) \models \varphi \rightarrow \psi$ *iff* $(D, \sigma) \not\models \varphi$ *or* $(D, \sigma) \models \psi$*;*
$(D, \sigma) \models \forall x \varphi$ *iff for every* $u \in adom(D)$, $(D, \sigma^x_u) \models \varphi$.

A formula φ *is* true *in* D*, written* $D \models \varphi$*, iff* $(D, \sigma) \models \varphi$ *for all* U*- assignments* σ.

It can easily be seen that the satisfaction of a formula does not depend on the values that σ assigns to non-free variables. Observe that we are adopting an *active-domain* semantics, i.e., all quantified variables range over the active domain of D. Also, notice that constants are *uninterpreted*, i.e., two constants are equal iff they are the same constant. We can now formally introduce the notion of *artifact system*.

Definition 5 (Artifact System). *An* artifact system *is a tuple* $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi \rangle$*, where:*

- $\mathcal{D} = \{P_1/a_1, \ldots, P_n/a_n\}$ *is a database schema;*
- U *is a (possibly infinite) interpretation domain;*
- D_0 *is an* initial database instance*;*
- Φ *is a finite set of parametric* artifact actions *of the form* $\alpha(\boldsymbol{x}) = \langle \pi(\boldsymbol{y}), \psi(\boldsymbol{z}) \rangle$*, where:*
  - $\boldsymbol{x} = \boldsymbol{y} \cup \boldsymbol{z}$*;*
  - $\alpha(\boldsymbol{x})$ *is the* action signature *and* $\boldsymbol{x}$ *the set of its* (formal) parameters*;*
  - $\pi(\boldsymbol{y})$ *is the* action precondition*, i.e., a FO-formula over* $\mathcal{D}$ *and* U*;*
  - $\psi(\boldsymbol{z})$ *is the* action postcondition*, i.e., a FO-formula over* $\mathcal{D} \cup \mathcal{D}'$ *and* U*, with* $\mathcal{D}' \doteq \{P'_1/a_1, \ldots, P'_n/a_n\}$*;*

For an action $\alpha(\boldsymbol{x})$ we let $const(\alpha(\boldsymbol{x})) = const(\pi(\boldsymbol{x})) \cup const(\psi(\boldsymbol{x}))$. Moreover, if $|\boldsymbol{x}| = \ell$, an *execution* of $\alpha(\boldsymbol{x})$ with *actual parameters* $\boldsymbol{u} \in U^\ell$, is the *ground* action $\alpha(\boldsymbol{u}) = \langle \pi(\boldsymbol{v}), \psi(\boldsymbol{w}) \rangle$, where $\boldsymbol{v}$ (resp., $\boldsymbol{w}$) is obtained by replacing each $\boldsymbol{y}$ ($\boldsymbol{z}$) component y_i (z_i) with the value occurring in $\boldsymbol{u}$ at the same position as y_i (z_i) in $\boldsymbol{x}$ (observe that such replacements make both $\pi(\boldsymbol{v})$ and $\psi(\boldsymbol{w})$ sentences).

The semantics of an artifact system is given in terms of its possible executions, captured by a Kripke structure, whose states are instances of the database schema and whose transitions correspond to the execution of some action.

Definition 6 (Model of an artifact system). *Given an artifact system* $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi \rangle$*, the* model of $\mathcal{S}$ *is the Kripke structure* $\mathcal{K} = \langle \Sigma, D_0, \tau \rangle$*, where:*

- $\Sigma \subseteq \mathcal{I}_\mathcal{D}(U)$ *is the* set of states*;*
- $D_0 \in \Sigma$ *is the* initial state*;*
- $\tau \subseteq \Sigma \times \Sigma$ *is the* transition relation *such that* $\tau(D, D')$ *iff for some action* $\alpha(\boldsymbol{x}) \in \Phi$*, there exists an execution* $\alpha(\boldsymbol{u}) = \langle \pi(\boldsymbol{v}), \psi(\boldsymbol{w}) \rangle$ *such that:*
  - $adom(D') \subseteq adom(D) \cup \{u_1, \ldots, u_\ell\} \cup const(\psi)$*;*
  - $D \models \pi(\boldsymbol{v})$*; in this case we say that the action is* enabled*;*
  - $D \oplus D' \models \psi(\boldsymbol{w})$*, where* $D \oplus D'$ *is the* $(\mathcal{D} \cup \mathcal{D}')$*-interpretation over* U *s.t. for every* $i \in \{1, \ldots, n\}$*,* $D \oplus D'(P_i) = D(P_i)$*, and* $D \oplus D'(P'_i) = D'(P_i)$.

As usual, preconditions represent the requirements that a state needs to fulfil in order for an action to be enabled, and postconditions define the possible successors of the state where the action is executed. The former are simply $\mathcal{L}_{\mathcal{D},U}$ FO-sentences to be evaluated against the current state, while the latter are FO-sentences using *unprimed* and *primed* relational symbols from $\mathcal{D}$, that refer to relations in the current and the successor state. Intuitively, given two states (i.e., $\mathcal{D}$-interpretations) D and D', the operator $\oplus$ constructs a new "joint" interpretation, namely $D \oplus D'$, interpreting unprimed relational symbols in D, and primed in D'. Notice that the active domain of $D \oplus D'$ may include values from $adom(D)$, $const(\psi)$, and $\boldsymbol{u}$ only. Thus D' may contain additional values with respect to D', i.e., those from $const(\varphi)$ and $\boldsymbol{u}$, and is necessarily finite. For this reason, given D and $\alpha(\boldsymbol{u})$ the whole set of D-successors is computable.

Since the actual parameters come from an infinite domain, the set of $\mathcal{K}$-states Σ is in general infinite. This may happen even in presence of a fixed bound on the active domain of each state, which, although bounded, may correspond to any of the infinitely many finite subsets of U not exceeding the bound.

3 The Order-to-Cash Business Process

In this section we formalise a business process inspired by an IBM customer example [13] as an AS. Specifically, the *Order-to-Cash* scenario describes the process a product undergoes from order to delivery. It involves a *manufacturer*, some *customers*, and some *suppliers*. The process starts when a customer prepares and submits a *customer purchase order* (CPO), i.e., a list of products the customer needs.

Upon receiving a CPO, the manufacturer first prepares a *work order* (WO), i.e., a list of the components needed to assemble the requested products. Then she selects a possible supplier for each component, prepares one *material purchase order* (MPO) per selected supplier, and submits each MPO to the corresponding supplier.

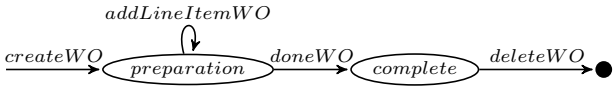
A supplier can either accept or reject a received MPO. In the former case he delivers the requested components to the manufacturer. In the latter case he notifies the manufacturer of his rejection. If an MPO is rejected, the manufacturer can delete it and prepare and submit new MPOs for the rejected components. When all the components required by a product have been delivered to the manufacturer, she assembles the product and, provided the order has been paid for, delivers it to the customer. Any order (directly on indirectly) related to a CPO can be deleted only after the CPO is deleted.

It is natural to identify 3 classes of artifacts in this process, each corresponding to some of the orders manipulated by the participants (CPO, WO and MPO). An intuitive representation of the artifact lifecycles, capturing only the dependence of actions from the artifact statuses, is shown in Fig. 1. We stress that this is an incomplete representation of the business process, as the interaction between actions and the artifact data content is not represented.

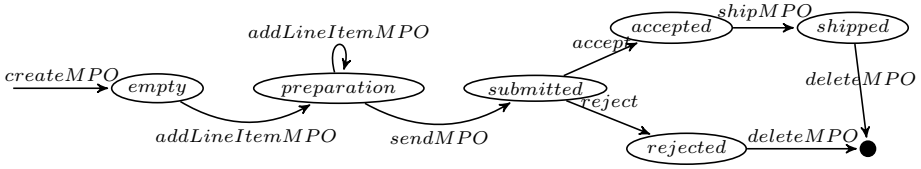
Next, we provide a formal model of the process as an artifact system, where the artifact data models are represented as a relational database schema, and the corresponding lifecycles are formally characterised by an appropriate set of actions.

As to the data model, we reserve a distinguished relation for each artifact class, as well as some auxiliary relations necessary to model the line items present in MPOs

(a) Customer Purchase Order lifecyle



(b) Work Order Lifecyle



(c) Material Purchase Order lifecyle

Fig. 1. Lifecycles of the artifacts involved in the order-to-cash scenario.

and WOs. In addition, we introduce *static* relations to store customer, supplier, product, and material information. The resulting database schema $\mathcal{D}$ is shown in Table 1. $R(n_1, \ldots, n_k)$ defines the relation symbol R of arity k (R/k), where n_i is the name of the i-th component, or *attribute*, of its tuples. The relations $Customers$, $Suppliers$, $Products$, and $Materials$, as well as CPO, WO, and MPO, are self-explanatory. Observe the presence of the attribute $status$ in the relations corresponding to artifacts. As to WO_LI and MPO_LI, they contain the line items occurring in WOs and MPOs, respectively. For instance, the fact that two line items containing materials with codes 5 and 6, and quantities 20 and 15, respectively, occur in the WO with $id = 10$, is captured by the presence of tuples $\langle 10, 5, 20\rangle$ and $\langle 10, 6, 15\rangle$ in WO_LI.

Table 1. Database schema $\mathcal{D}$ of the artifact system for the cash-to-order scenario.

$$
\begin{gathered}
Customers(id, name),\\
Suppliers(id, name),\\
Products(code, descr),\\
Materials(code, descr),\\
CPO(id, customer\_id, product\_code, status),\\
WO(id, cpo\_id, status),\\
WO\_LI(wo\_id, mat\_code, qty),\\
MPO(id, wo\_id, supplier, status),\\
MPO\_LI(mpo\_id, mat\_code, qty).
\end{gathered}
$$

As interpretation domain, we consider the infinite set U of alphanumeric strings. In the initial database instance D_0 the only non-empty relations are $Customers$, $Suppliers$, $Products$, and $Materials$, which contain background information, such as the possible customers, or a catalogue of available products.

System actions capture *legal* operations on the underlying database and, thus, on artifacts. In Table 2 we report some of their specifications. Variables (from V) and constants (from U) are distinguished by fonts v and c, respectively. We adopt the convention that an action affects only those relations whose name occurs in ψ.

Table 2. Specification of the actions affecting the artifact WO in the order-to-cash scenario.

- $createWO(id, cpo) = \langle \pi(id, cpo), \psi(id, cpo) \rangle$, where:
  - $\pi(id, cpo) \equiv \exists code, cid, st\ CPO(cpo, code, cid, st) \wedge$
    $\forall id', c, s\ (WO(id', c, s) \rightarrow id \neq id')$
  - $\psi(id, cpo) \equiv WO'(id, cpo, \mathsf{preparation}) \wedge$
    $\forall id', c, s\ (id \neq id' \rightarrow (WO(id', c, s) \leftrightarrow WO'(id', c, s)))$
- $addLineItemWO(wo, mat, qty) = \langle \pi(wo, mat), \psi(wo, mat, qty) \rangle$, where:
  - $\pi(wo, mat) \equiv \exists cpo\, WO(wo, cpo, \mathsf{preparation}) \wedge \exists desc\, Materials(mat, desc) \wedge$
    $\neg \exists q\, WO\_LI(wo, mat, q)$
  - $\psi(wo, mat, qty) \equiv WO\_LI'(wo, mat, qty) \wedge$
    $\forall w, m, q \Big((WO\_LI(w, m, q) \rightarrow WO\_LI'(w, m, q)) \wedge$
    $(WO\_LI'(w, m, q) \rightarrow (WO\_LI(w, m, q) \vee (w = wo \wedge m = mat \wedge q = qty))) \Big)$
- $doneWO(wo) = \langle \pi(wo), \psi(wo) \rangle$, where:
  - $\pi(wo) \equiv \exists cpo\, WO(wo, cpo, \mathsf{preparation})$
  - $\psi(wo) \equiv \forall w, c, s \Big((w \neq wo \rightarrow (WO(w, c, s) \leftrightarrow WO'(w, c, s))) \wedge$
    $(WO(wo, c, s) \rightarrow (WO'(wo, c, \mathsf{complete}) \wedge (s \neq \mathsf{complete} \rightarrow \neg WO'(wo, c, s))))) \Big)$

Consider the action $createWO$, whose purpose is the creation of a WO-artifact instance. Its precondition requires that cpo is the identifier of some existing CPO, and that id in the new WO is unique with respect to those present when the action is executed. Its postcondition states that, upon execution, the WO relation contains exactly one additional tuple, with identifier attribute set to id, and attribute $status$ set to preparation.

The action $addLineItem$ adds a line item, i.e., a component, to an existing WO. It takes in input the identifier of the WO-artifact (wo), that of the material to add (mat), and the needed quantity (qty). The precondition requires that such parameters correspond to some existing WO-artifact and material, and that the WO-artifact is in state preparation. Moreover, it is required that the material being added is not already present in the WO. The postcondition states that the new line item is added to WO_LI.

As an example of action triggering an artifact's status transition, consider $doneWO$. It is executable only if the WO-artifact is in status preparation and its effect is to set the status attribute to complete.

Notice that although actions are typically conceived to manipulate artifacts of a specific class, e.g., $createWO$ manipulates WO-artifacts, their preconditions and postconditions may depend on artifact instances of different classes, e.g. $createWO$'s precondition depends on CPO-artifacts. We stress that action executability depends not only on the status attribute of an artifact, but on the data content of the whole database, i.e., of all other artifacts. Similarly, action executions affect not only status attributes.

As actions are executed, the database content, hence the state of each artifact, changes. Obviously, after executing an action, other actions become executable, their executions change the database state, thus make other actions executable, and so on.

4 Verification of Artifact Systems

We focus on the problem of verifying an artifact system against a temporal specification of interest. Since the states of an artifact are characterised by their data content, the atomic components of the specifications need to capture relational properties pertaining to the states. This, together with the fact that the domain of data may be infinite, makes the problem substantially more challenging than standard model checking [8].

We first introduce syntax and semantics of our specification language.

Definition 7 (Sentence-atomic FO-CTL formulas (over a system $\mathcal{S}$)). *Given an artifact system $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi \rangle$, the language $\mathcal{L}_\mathcal{S}$ of* sentence-atomic *FO-CTL formulas over $\mathcal{S}$ is inductively defined as follows:*

$$\varphi ::= \phi \mid (\varphi) \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid AX\varphi \mid A\varphi\mathcal{U}\varphi \mid E\varphi\mathcal{U}\varphi,$$

where ϕ is an FO-sentence from $\mathcal{L}_{\mathcal{D},U}$.

The notions of free and bound variables extend in the obvious way to $\mathcal{L}_\mathcal{S}$, as well as functions $vars$, $free$, and $const$. Observe that formulas in $\mathcal{L}_\mathcal{S}$ are in fact sentences, as all of their atomic components are FO-sentences. We use the standard abbreviations $EX\varphi \equiv \neg AX\neg\varphi$, $AF\varphi \equiv A\top\mathcal{U}\varphi$, $AG\varphi \equiv \neg E\top\mathcal{U}\neg\varphi$, $EF\varphi \equiv E\top\mathcal{U}\varphi$, and $EG\varphi \equiv \neg A\top\mathcal{U}\neg\varphi$.

In order to define the semantics of $\mathcal{L}_\mathcal{S}$, we first define *runs* on a Kripke structure.

Definition 8 ($\mathcal{K}$-runs). *Given a Kripke structure $\mathcal{K} = \langle \Sigma, D_0, \tau \rangle$ of an artifact system $\mathcal{S}$, a* $\mathcal{K}$-run *r from a $\mathcal{K}$-state $D \in \Sigma$ is an infinite sequence of $\mathcal{K}$-states $r = D^0 \rightarrow D^1 \rightarrow \cdots$ such that $D^0 = D$ and $\tau(D^i, D^{i+1})$, for $i \geq 0$. For every run r and $i \geq 0$, we define $r(i) \doteq D^i$.*

The semantics of $\mathcal{L}_\mathcal{S}$ formulas is provided in terms of the model $\mathcal{K}$ of $\mathcal{S}$.

Definition 9 (Semantics of $\mathcal{L}_\mathcal{S}$). *Consider a system $\mathcal{S}$ and its model $\mathcal{K}$. Given a formula $\varphi \in \mathcal{L}_\mathcal{S}$ and a $\mathcal{K}$-state $D \in \Sigma$, the satisfaction relation $\models$ is inductively defined as follows:*

- $(\mathcal{K}, D) \models \varphi$ *iff* $D \models \varphi$, *if φ is an FO-sentence;*
- $(\mathcal{K}, D) \models (\varphi)$ *iff* $(\mathcal{K}, D) \models \varphi$;
- $(\mathcal{K}, D) \models \neg\varphi$ *iff* $(\mathcal{K}, D) \not\models \varphi$;
- $(\mathcal{K}, D) \models \varphi \rightarrow \psi$ *iff* $(\mathcal{K}, D) \not\models \varphi$ *or* $(\mathcal{K}, D) \models \psi$;
- $(\mathcal{K}, D) \models AX\varphi$ *iff for all $\mathcal{K}$-runs r s.t.* $r(0) = D$, $(\mathcal{K}, r(1)) \models \varphi$;
- $(\mathcal{K}, D) \models A\varphi\mathcal{U}\psi$ *iff for all $\mathcal{K}$-runs r s.t.* $r(0) = D$, $\exists k \geq 0$ *s.t.* $(\mathcal{K}, r(k)) \models \psi$ *and* $\forall j$ *s.t.* $0 \leq j < k$, $(\mathcal{K}, r(j)) \models \varphi$;
- $(\mathcal{K}, D) \models E\varphi\mathcal{U}\psi$ *iff for some $\mathcal{K}$-run r,* $r(0) = D$, $\exists k \geq 0$ *s.t.* $(\mathcal{K}, r(k)) \models \psi$, *and* $\forall j$ *s.t.* $0 \leq j < k$, $(\mathcal{K}, r(j)) \models \varphi$.

A formula φ is true *in $\mathcal{K}$, written $\mathcal{K} \models \varphi$, if $(\mathcal{K}, D_0) \models \varphi$. We say that $\mathcal{S}$ satisfies φ, written $\mathcal{S} \models \varphi$, if $\mathcal{K} \models \varphi$.*

In the following we describe some properties of the artifact system introduced in Section 3 to model the order-to-cash business process. The first one requires that a product can be shipped to a customer only if all the required materials are already shipped to the manufacturer:

$$\varphi_{ship} = AG\, \forall c \left(shippedCPO(c) \rightarrow \forall m \left(related(c, m) \rightarrow shippedMPO(m) \right) \right),$$

where: $shippedCPO(x) \equiv \exists c, p\ CPO(x, c, p, \mathsf{shipped})$ and $shippedMPO(x) \equiv \exists w, sp\ MPO(x, w, sp, \mathsf{shipped})$, respectively, capture the fact that the CPO and the MPO with $id = x$ are in status shipped; and $related(x, y) \equiv \exists c, p, s\, CPO(x, c, p, s) \wedge \exists w, s\, WO(w, x, s) \wedge \exists sp, st\ MPO(y, w, sp, st)$ holds iff the MPO with $id = y$ is related, via some WO, to the CPO with $id = x$.

The next property captures the existence of a run containing a state whose active domain exceeds a given size-threshold t:

$$\varphi_{t+} = EF\, \exists x_1, \ldots, x_{t+1} \bigwedge_{i \neq j} x_i \neq x_j.$$

We can also express the fact that from some state there exists a way to achieve some goal (although this may not necessarily happen). For instance, the next formula states that there exists always a way to empty all non-static relations:

$$\varphi_{empty} = AG\ EF\ (emptyCPO \wedge emptyWO \wedge emptyMPO),$$

where: $emptyCPO \equiv \neg\exists i, c, p, s\, CPO(i, c, p, s)$, $emptyWO \equiv \neg\exists i, c, s\, WO(i, c, s) \wedge \neg\exists w, m, q\, WO\_LI(w, m, q)$, and $emptyMPO$ is similar to $emptyWO$.

Specifications such as those above are useful to describe properties of ASs. Typically, we are interested in checking automatically whether they are satisfied on particular systems. If we consider the AS described previously in the order-to-cash scenario, it is not difficult to see that φ_{ship}, φ_{t+}, and φ_{empty} are satisfied.

Observe that while we may, and in fact can, ascertain the truth of those specifications on this specific example, we cannot be able to do so automatically on any possible system, as the general model checking problem is undecidable. It is therefore natural to investigate decidable subclasses of this problem.

4.1 The General Problem

We are interested in exploring the model checking problem for artifact systems. Formally, this amounts to checking whether an artifact system $\mathcal{S}$ satisfies a specification $\varphi \in \mathcal{L}_{\mathcal{S}}$, i.e., $\mathcal{S} \models \varphi$. It can be shown that the problem is decidable for U finite, and undecidable otherwise.

To see the former, observe that if U contains only finitely many distinct elements, its model contains a finite set of states, whose (relational) data content can be captured by a finite set of propositions (namely, one proposition per fact). By quantifier elimination φ can be transformed into an equivalent propositional (CTL) temporal formula,

whose propositions are ground atoms from $\mathcal{L}_{\mathcal{S}}$. This corresponds to reducing the whole problem to standard model checking, which is known to be decidable [8].

For the latter, we have the following result.

Theorem 1. *The model checking problem for artifact systems is undecidable.*

Proof (Sketch). The theorem can be proven by showing that every Turing machine T whose tape contains an initial input I can be simulated by an artifact system $\mathcal{S}_{T,I}$, and that the problem of checking whether T terminates on that particular input can be reduced to checking whether $\mathcal{S}_{T,I} \models \varphi$, where φ encodes the termination condition. The detailed construction is similar to that of Th. 4.10 in [11].

The theorem essentially states the general impossibility of checking the correctness of an artifact system's design, when the interpretation domain is infinite. As this assumption is typically fulfilled in practice, this is a considerably negative result. In the following, rather than focusing on the design, we explore conditions on the concrete implementation of a system that yield decidability and enable the reduction of the verification task to standard model checking of finite-state systems.

5 Verification of Deployed Artifact Systems

Artifact systems serve as a theoretical model for systems to be implemented and deployed on actual machines [9, 14]. It is therefore of interest, and of particular relevance in practice, to investigate the model checking problem for such concrete implementations. Observe that any running system can use only the finite, bounded memory provided by the machine it is deployed on (e.g., corresponding to all virtual and physical memory of the server). We show in the following that in this concrete setting the model checking problem against FO-CTL specifications is decidable. Precisely, we show that given a size-bound on the number of values a machine can store at each state, it is decidable whether the artifact system executed on a machine with that bound satisfies the specification.

We start this analysis by defining formally the model of an artifact system deployed on a concrete machine.

Definition 10 (b-bounded model of a system $\mathcal{S}$). *Consider a system $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi \rangle$, its model $\mathcal{K} = \langle \Sigma, D_0, \tau \rangle$, and a bound $b \in \mathbb{N}$ such that $b \geq |adom(D_0)|$. The* b-bounded model of $\mathcal{S}$ *is the Kripke structure $\mathcal{K}_b = \langle \Sigma_b, D_0, \tau_b \rangle$, such that:*

- $\Sigma_b \doteq \{D \in \Sigma$ *such that* $|adom(D)| \leq b\}$*;*
- $\tau_b \doteq \{\langle D, D' \rangle \in \tau$ *such that* $D, D' \in \Sigma_b\}$.

Roughly speaking, $\mathcal{K}_b$ is a sub-model of $\mathcal{S}$, obtained from $\mathcal{K}$ by considering only those $\mathcal{K}$-runs whose states do not exceed the size-bound b. Intuitively, bounded models capture the possible executions of $\mathcal{S}$ on a machine able to accommodate at most b elements. Notice that because the artifact system may still reach infinitely many states, verifying $\mathcal{K}_b$ against a specification φ via an exhaustive visit of its state space is not a viable approach. Nonetheless, we next show that a finite-state, *abstract* model $\hat{\mathcal{K}}_{b,\varphi}$ capturing all the features of $\mathcal{K}_b$ relevant to φ can be constructed. Specifically, we demonstrate

that verifying $\hat{\mathcal{K}}_{b,\varphi}$ against φ is equivalent to verifying $\mathcal{K}_b$ against φ. This will show that the verification of *deployed* ASs is actually decidable. In practice, $\hat{\mathcal{K}}_{b,\varphi}$ is defined indirectly, as the b-bounded model of an *abstract* system $\hat{\mathcal{S}}_{b,\varphi}$ obtained from $\mathcal{S}$, b, and φ, as follows.

Definition 11 ((b,φ)-bounded abstract system). *Given a system $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi \rangle$, a sentence-atomic FO-CTL sentence $\varphi \in \mathcal{L}_{\mathcal{S}}$, and a bound $b \geq |adom(D_0)|$, the (b,φ)-*bounded abstract system of $\mathcal{S}$ *is the system $\hat{\mathcal{S}}_{b,\varphi} = \langle \mathcal{D}, \hat{U}, D_0, \Phi \rangle$, where $\hat{U} = C_{\mathcal{S},\varphi} \cup \hat{C}$, and:*

- $C_{\mathcal{S},\varphi} = const(\varphi) \cup \bigcup_{\phi \in \Phi} const(\phi) \cup adom(D_0)$;
- $\hat{C}$ *is any set of* symbols *s.t.:*
  - $\hat{C} \cap C_{\mathcal{S},\varphi} = \emptyset$,
  - $|\hat{C}| = b + v$, *with* $v = \max_{\phi \in \Phi}\{|vars(\phi)|\}$.

As it turns out, $\hat{\mathcal{S}}_{b,\varphi}$ is an artifact system analogous to $\mathcal{S}$, except for the interpretation domain. Specifically, $\hat{U}$ contains all the constants mentioned in $\mathcal{S}$ or in φ, plus $b + v$ additional symbols. Intuitively, these symbols are used to *simulate* the database content at each state, as well as the new values that actions may introduce upon execution. In particular, at least b distinct symbols are required for the former and v for the latter. Observe that by preserving (the identity of) all mentioned constants, the FO-formulas occurring in $\mathcal{S}$ and φ need no syntactic transformation to preserve their semantics.

As anticipated above, since $\hat{U}$ is finite, checking $\hat{\mathcal{K}}_{b,\varphi} \models \varphi$ is decidable. Below, we show that $\hat{\mathcal{S}}_{b,\varphi}$ contains enough information to check the bounded model of the original artifact system $\mathcal{S}$ against the specification φ.

Theorem 2. *Consider a system $\mathcal{S}$ with U infinite, a bound $b \geq |adom(D_0)|$, and a sentence-atomic FO-CTL formula $\varphi \in \mathcal{L}_{\mathcal{S}}$. If $\hat{\mathcal{S}}_{b,\varphi}$ is the (b,φ)-bounded abstract system of $\mathcal{S}$ then $\mathcal{K}_b \models \varphi \Leftrightarrow \hat{\mathcal{K}}_{b,\varphi} \models \varphi$, where $\mathcal{K}_b$ is the b-bounded model of $\mathcal{S}$ and $\hat{\mathcal{K}}_{b,\varphi}$ is the b-bounded model of $\hat{\mathcal{S}}_{b,\varphi}$.*

The theorem shows that instead of checking $\mathcal{K}_b \models \varphi$ (where $\mathcal{K}_b$ is infinite), we can check $\hat{\mathcal{K}}_{b,\varphi} \models \varphi$. Since $\hat{\mathcal{K}}_{b,\varphi}$ is finite, this essentially corresponds to a standard model checking problem, thus any technique for this is also effective for the verification of the b-bounded model of $\mathcal{S}$.

Next, we sketch the main steps of the proof of Theorem 2. We essentially show that $\hat{\mathcal{K}}_{b,\varphi}$ is a sound abstraction of the bounded model of $\mathcal{S}$, that is, it retains enough information to carry out the verification task. Our approach is inspired by the decidability proof of verification of *input-bounded* ASM$^+$s presented in [11]. Interestingly, differently from that, the assumption of size-boundedness allows us to conclude that $\mathcal{K}_b$ and $\hat{\mathcal{K}}_{b,\varphi}$ are *bi-similar*, thus enabling verification of branching-time properties. Firstly, we define when two $\mathcal{D}$-instances are *isomorphic*.

Definition 12 (C-isomorphic $\mathcal{D}$-instances). *Two $\mathcal{D}$-instances D and $\hat{D}$, respectively over U and $\hat{U}$, are said C-isomorphic, for $C \subseteq U, \hat{U}$, written $D \sim_C D$, iff there exists a bijection $i : adom(D) \cup C \mapsto adom(\hat{D}) \cup C$ that is the identity on C, and such that for every $j = 1, \ldots, n$, and for every $\boldsymbol{u} \in adom(D)^{a_i}$, $D \models P_j(\boldsymbol{u}) \Leftrightarrow \hat{D} \models P_j(i(\boldsymbol{u}))$, where $i(\boldsymbol{u}) \doteq \langle i(u_1), \ldots, i(u_{a_j}) \rangle$.*

The relation $\sim_C$ can be shown to be an equivalence relation.

The following rephrases a well-known result.

Proposition 1. *Given two $\mathcal{D}$-instances D and $\hat{D}$, respectively over U and $\hat{U}$, an FO-sentence φ from $\mathcal{L}_{\mathcal{D},U}$, and a set $C \subseteq U, \hat{U}$ such that $const(\varphi) \subseteq C$, if $D \sim_C \hat{D}$, then*

$$D \models \varphi \Leftrightarrow \hat{D} \models \varphi.$$

This is the main ingredient that allows us to prove Theorem 2. It states that if two $\mathcal{D}$-instances are C-isomorphic, they are indistinguishable by any sentence over $\mathcal{D}$ containing only constants from C. We can now define when two Kripke structures are *bi-similar*.

Definition 13 (C-bisimilar Kripke structures). *Given two Kripke structures $\mathcal{K} = \langle \Sigma, D_0, \tau \rangle$ and $\hat{\mathcal{K}} = \langle \hat{\Sigma}, \hat{D}_0, \hat{\tau} \rangle$, with $\Sigma \subseteq \mathcal{I}_{\mathcal{D}}(U)$ and $\hat{\Sigma} \subseteq \mathcal{I}_{\mathcal{D}}(\hat{U})$, and a finite set of constants $C \subseteq U, \hat{U}$, $\mathcal{K}$ and $\hat{\mathcal{K}}$ are said C-*bisimilar*, written $\mathcal{K} \approx_C \hat{\mathcal{K}}$, iff there exists a relation $R \subseteq \Sigma \times \hat{\Sigma}$, called C-*(preserving) bisimulation, s.t. $\langle D_0, \hat{D}_0 \rangle \in R$, and if $\langle D, \hat{D} \rangle \in R$ then:*

- *$D \sim_C \hat{D}$;*
- *for all D' s.t. $\tau(D, D')$ there exists $\hat{D}'$ s.t. $\hat{\tau}(\hat{D}, \hat{D}')$ and $\langle D', \hat{D}' \rangle \in R$;*
- *for all $\hat{D}'$ s.t. $\hat{\tau}(\hat{D}, \hat{D}')$ there exists D' s.t. $\tau(D, D')$ and $\langle D', \hat{D}' \rangle \in R$.*

When $\langle D, \hat{D} \rangle \in R$, we say that D and $\hat{D}$ are C-bisimilar (with respect to $\mathcal{K}$ and $\hat{\mathcal{K}}$), written $D \approx_C \hat{D}$.

Since by definition $D \approx_C \hat{D}$ implies $D \sim_C \hat{D}$, by Proposition 1, for every FO-sentence φ over $\mathcal{D}$ and U such that $const(\varphi) \subseteq C$, $D \models \varphi \Leftrightarrow \hat{D} \models \varphi$. Observe that the atoms of a FO-CTL sentence are FO-sentences and can thus be evaluated at each state of a Kripke structure. Therefore, we have the following result.

Lemma 1. *Given $\mathcal{K}$, $\hat{\mathcal{K}}$, and C as above, for every sentence-atomic FO-CTL formula φ over $\mathcal{D}$ and U such that $const(\varphi) \subseteq C$, if $D \in \Sigma$ and $\hat{D} \in \hat{\Sigma}$ are such that $D \approx_C \hat{D}$, then*

$$(\mathcal{K}, D) \models \varphi \Leftrightarrow (\hat{\mathcal{K}}, \hat{D}) \models \varphi.$$

Proof (Sketch). By induction on the structure of φ.

In other words sentence-atomic FO-CTL formulas containing only constants from C do not distinguish among C-bisimilar Kripke structures. As a consequence, an infinite-state Kripke structure can be verified against a sentence-atomic FO-CTL formula by verifying any other C-bisimilar structure, including a finite one, against the same specification.

The final step of the proof consists in showing that the b-bounded model of $\mathcal{S}$, $\mathcal{K}_b$, which is infinite-state in general, is $C_{\mathcal{S},\varphi}$-bisimilar to $\hat{\mathcal{K}}_{b,\varphi}$, which is, instead, finite by construction (see Def. 11).

Lemma 2. *Consider a system $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi \rangle$ and a sentence-atomic FO-CTL formula $\varphi \in \mathcal{L}_{\mathcal{S}}$. Fix a bound $b \geq |adom(D_0)|$, let $\mathcal{K}_b$ be the b-bounded model of $\mathcal{S}$, $\hat{\mathcal{S}}_{b,\varphi} = \langle \mathcal{D}, \hat{U}, D_0, \Phi \rangle$ the (b, φ)-bounded abstract system of $\mathcal{S}$, and $\hat{\mathcal{K}}_{b,\varphi}$ its b-bounded model. Then, for $C_{\mathcal{S},\varphi}$ as in Def. 11, $\mathcal{K}_b \approx_{C_{\mathcal{S},\varphi}} \hat{\mathcal{K}}_{b,\varphi}$.*

Proof (Sketch). The proof consists in constructing a particular $C_{\mathcal{S},\varphi}$-bisimulation between $\mathcal{K}_b$ and $\hat{\mathcal{K}}_{b,\varphi}$. The result then follows.

Lemmas 1 and 2 allow us to prove Theorem 2; so we achieve decidability of the problem in presence of a known bound.

Obviously, not all specifications satisfied by $\hat{\mathcal{K}}_{b,\varphi}$ are preserved in the original (unbounded execution of) $\mathcal{S}$. For instance, consider the specification $\varphi_{t-} \doteq \neg\varphi_{t+}$, with φ_{t+} as in Section 3, which expresses the fact that all states of every run contain at most t distinct elements. This is clearly satisfied by $\hat{\mathcal{K}}_{t,\varphi_{t-}}$, but not by $\mathcal{S}$. On the other hand, preservation is guaranteed for existential specifications. Precisely, let $\mathcal{L}^E_{\mathcal{S}} \subseteq \mathcal{L}_{\mathcal{S}}$ be the sublanguage of sentence-atomic FO-ECTL formulas φ, inductively defined as:

$$\varphi ::= \phi \mid (\varphi) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid EX\varphi \mid E\varphi\mathcal{U}\varphi,$$

where ϕ is a FO-sentence. Then, we have the following result:

Theorem 3. *Given a system $\mathcal{S}$, a bound $b \geq |adom(D_0)|$, and a sentence-atomic FO-ECTL formula $\varphi \in \mathcal{L}^E_{\mathcal{S}}$, if $\hat{\mathcal{K}}_{b,\varphi} \models \varphi$ then $\mathcal{S} \models \varphi$.*

Thus, a sound (though incomplete) technique to check whether $\mathcal{S} \models \varphi$ for $\varphi \in \mathcal{L}^E_{\mathcal{S}}$ consists in iteratively increasing b and checking whether $\hat{\mathcal{K}}_{b,\varphi} \models \varphi$. If at some point the check is successful, then $\mathcal{S} \models \varphi$. For instance, to check whether the AS $\mathcal{S}$ of Section 3 contains a run with some state exceeding a size-threshold T, one can iteratively check whether $\hat{\mathcal{K}}_{b,\varphi_{T+}} \models \varphi_{T+}$ by increasing b at each iteration, starting with $b = T + 1$. In this particular case, it is easy to see that for any T the check is successful for $b = T+1$, as the system is unbounded. A similar approach can be adopted to find counterexamples of *universal* specifications (i.e., negations of existential ones) such as φ_{ship}. There are obvious correspondences with the technique of Bounded Model Checking [5]. Here, however, the bound is on the size of the states, rather than the length of runs.

If $\mathcal{S}$ is size-bounded itself (and the bound is known), all specifications are preserved from the abstract to the concrete model and viceversa. If this is not the case, however, nothing can be said with respect to specifications that are neither universal nor existential. For instance, by checking that $\hat{\mathcal{K}}_{b,\varphi_{empty}} \models \varphi_{empty}$ we cannot conclude anything in general about $\mathcal{S} \models \varphi_{empty}$. That is, our technique fails in proving that $\mathcal{S} \models \varphi_{empty}$ (although we know that this holds). However, by changing the bound b, we can check whether the property holds on any deployed instance of $\mathcal{S}$.

We now analyse the time-complexity of our technique, by considering the cost of reducing the problem of checking whether $\mathcal{K}_b \models \varphi$ to standard CTL model checking. Given $\mathcal{S}$, b, and φ, this essentially requires: building $\hat{\mathcal{K}}_{b,\varphi}$ (from $\hat{\mathcal{S}}_{b,\varphi}$); transforming φ into a propositional CTL formula φ_p, by recursively replacing each formula of the form $\forall x\varphi(x)$ with $\bigwedge_{\hat{u}\in\hat{U}} \varphi(\hat{u})$; and then applying an algorithm for CTL model checking, to check whether $\hat{\mathcal{K}}_{b,\varphi} \models \varphi_p$. This gives the following result.

Theorem 4. *Given an artifact system $\mathcal{S} = \langle \mathcal{D}, U, D_0, \Phi\rangle$, a sentence-atomic FO-CTL formula $\varphi \in \mathcal{L}_{\mathcal{S}}$, and a bound $b \geq |adom(D_0)|$, checking whether $\mathcal{K}_b \models \varphi$ can be done in time $\mathcal{O}(2^{2|\hat{U}|^a}|\varphi||\hat{U}|^{|\varphi|})$, where $a = \sum_{i=1,\ldots,n} a_i$, with a_i the arity of $P_i \in \mathcal{D}$, and $\hat{U}$ defined as in Def. 11.*

Proof (sketch). $\hat{\mathcal{K}}_{b,\varphi} \models \varphi_p$ *can be checked in time* $\mathcal{O}((s+t)\cdot|\varphi_p|)$*, where* s *and* t *are, respectively, the number of states and transitions of* $\hat{\mathcal{K}}_{b,\varphi}$ *[8]. We have* $s \leq |\mathcal{I}_{\mathcal{D}}(\hat{U})| \leq 2^{|\hat{U}|^a}$*,* $t \leq s^2$*, and* $s+t \leq 2s^2 \leq 2^{2|\hat{U}|^a+1}$*. For* φ_p*, each quantifier elimination makes the current expansion grow by a factor* $|\hat{U}|$*, thus* $|\varphi_p| \leq |\varphi| \cdot |\hat{U}|^{|\varphi|}$*.*

Theorem 4 gives a doubly exponential bound, which comes from the arity of the relations in $\mathcal{D}$. Observe that the bound is singly exponential in $|\hat{U}|$, which is typically greater than a. Moreover, if one needs to check the correctness of $\mathcal{S}$ against φ for different bounds b, then a can be considered constant, and the cost of increasing b becomes only singly exponential. While certainly a high complexity, we observe that comparable bounds are obtained in [1] and [11]. In particular, the latter led to the implementation of a system performing surprisingly well in cases of practical interest. This may suggest that worst-case instances are not frequent in practice, and that a similar behavior might be observed also in implementations of our technique.

6 Conclusions and Future Work

In this paper we have considered the problem of checking a deployed artifact system against a temporal specification expressed in a FO extension of CTL. A notable feature of deployed systems is the existence of an upper bound on the number of elements they can store at each state at execution time. This allows us to reduce the problem to standard model checking by executing the system using only a finite number of abstract symbols instead of an infinite number of concrete ones.

Roughly speaking, our technique can be seen as an inspection of a fragment, containing only bounded states, of the original, concrete system. While this does not allow us to draw conclusions in the general case, it may provide some answers in particular cases. In this respect, we have shown that FO-ECTL properties satisfied by the abstract system are also satisfied by the concrete system, and that if the concrete system is bounded itself, our technique is complete.

We are interested in pursuing this work further. Firstly, we have shown that the bounded model of the abstract system is bi-similar to the bounded model of the concrete system. This suggests that all the obtained results, here presented in the context of FO-CTL, also hold for a FO extension of the μ-calculus analogous to [1]. If confirmed, this would imply that our work can be generalised to this setting.

Secondly, an interesting extension concerns the possibility of quantifying over variables across the scopes of modal operators, thus enabling us to capture temporal relationships among elements at different states. This introduces a major difficulty as it apparently requires to record elements from potentially infinitely many states. We are interested in pursuing abstraction techniques to avoid this problem.

Finally, the framework considered here may be extended to a Multi-Agent framework similarly to [2], thus accounting for the agents that execute the actions and their knowledge about the system.

References

1. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, R. De Masellis, and P. Felli. Foundations of Relational Artifacts Verification. In *Proc. of BPM*, 2011. To appear.
2. F. Belardinelli, A. Lomuscio, and F. Patrizi. A Computationally-Grounded Semantics for Artifact-Centric Systems and Abstraction Results. In *Proc. of IJCAI*, 2011. To appear.
3. D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic Composition of Transition-based Semantic Web Services with Messaging. In *Proc. of VLDB*, 2005.
4. K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su. Towards Formal Analysis of Artifact-Centric Business Process Models. In *Proc. of BPM*, 2007.
5. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded Model Checking. *Advances in Computers*, 58:118–149, 2003.
6. A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract Regular Model Checking. In *Proc. of CAV*, 2004.
7. D. Caucal. On Infinite Transition Graphs having a Decidable Monadic Theory. *Theoretical Computer Science*, 290(1):79–115, 2003.
8. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2000.
9. D. Cohn and R. Hull. Business Artifacts: A Data-Centric Approach to Modeling Business Operations and Processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.
10. A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic Verification of Data-centric Business Processes. In *Proc. of ICDT*, 2009.
11. A. Deutsch, L. Sui, and V. Vianu. Specification and Verification of Data-Driven Web Applications. *J. Comput. Syst. Sci.*, 73(3):442–474, 2007.
12. D. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyaschev. *Many-Dimensional Modal Logics: Theory and Applications*, volume 148 of *Studies in Logic*. Elsevier, 2003.
13. R. Hull, E. Damaggio, R. De Masellis, F. Fournier, M. Gupta, F. T. Heath III, S. Hobson, M. H. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín. Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events. In *Proc. of DEBS*, 2011. To appear.
14. R. Hull, N. C. Narendra, and A. Nigam. Facilitating Workflow Interoperation Using Artifact-Centric Hubs. In *Proc. of ICSOC-ServiceWave*, 2009.
15. I. Walukiewicz. Model Checking CTL Properties of Pushdown Systems. In *Proc. of FSTTCS*, 2000.