Automatic Verification of Data-Centric Business Processes

Alin Deutsch[*]
UC San Diego

Richard Hull[†]
IBM Research

Fabio Patrizi
Sapienza Univ. of Rome

Victor Vianu[*]
UC San Diego

ABSTRACT

We formalize and study business process systems that are centered around "business artifacts", or simply "artifacts". Artifacts are used to represent (real or conceptual) key business entities, including both their data schema and lifecycles. The lifecycle of an artifact type specifies the possible sequencings of services that can be applied to an artifact of this type as it progresses through the business process. The artifact-centric approach was introduced by IBM, and has been used to achieve substantial savings when performing business transformations.

In this paper, artifacts carry attribute records and internal state relations (holding sets of tuples) that services can consult and update. In addition, services can access an underlying database and can introduce new values from an infinite domain, thus modeling external inputs or partially specified processes described by pre-and-post conditions. The lifecycles associate services to the artifacts using declarative, condition-action style rules.

We consider the problem of statically verifying whether all runs of an artifact system satisfy desirable correctness properties expressed in a first-order extension of linear-time temporal logic. We map the boundaries of decidability for the verification problem and provide its complexity. The technical challenge to static verification stems from the presence of data from an infinite domain, yielding an infinite-state system. While much work has been done lately in the verification community on model checking specialized classes of infinite-state systems, the available results do not transfer to our framework, and this remains a difficult problem. We identify an expressive class of artifact systems for which verification is nonetheless decidable. The complexity of verification is PSPACE-complete, which is no worse than classical finite-state model checking.

This investigation builds upon previous work on verification of data-driven Web services and ASM transducers, while addressing significant new technical challenges raised by the artifact model.

1. INTRODUCTION

Businesses and other organizations increasingly rely on business process management, and in particular the management of electronic workflows underlying business processes. While most workflow is still organized around relatively flat process-centric models, over the past several years a *data-centric* approach to workflow has emerged. A watershed paper in this area is [44], which introduces the *artifact-centric* approach to workflow modeling. This approach focuses on data records, known as “business artifacts” or simply “artifacts”, that correspond to (real or conceptual) key business entities, including both their data schema and their lifecycle, which in turn specifies how/when services (a.k.a. tasks) are invoked and sequenced on the artifacts. This approach provides a simple and robust structure for workflow, and has been demonstrated in practice to yield substantial savings when performing business transformations [4].

From the formal perspective, little is understood about artifact-centric (and other data-centric) workflow. Citation [30] outlines a general framework for designing artifact-centric workflow meta-models, e.g., by varying the data model and/or the process specification paradigm for lifecycles that is used. Citations [27, 28] provide preliminary investigations into the analysis of artifact-centric workflows that use state-machine based lifecycle specifications, [6] provides a prliminary investigation into the analysis of artifact-centric workflows with declarative lifecycle specifications, and [26] provides a preliminary investigation into automated synthesis of declarative artifact-centric workflows. The current paper develops static analysis techniques in the context of substantially richer declarative artifact-centric workflows, in which each artifact carries, in addition to the data record containing business-relevant values, a full relational state (i.e., a set of tuples whose contents can change over time) used to hold additional business-relevant information. The focus here is on decision problems for a first-order extension of linear-time temporal logic. In general such decision problems are undecidable, but the paper identifies a large, useful class of workflows for which the complexity of verification is PSPACE-complete, which is no worse than classical finite-state model checking.

Following [7], an artifact-centric workflow model typically focuses on (a) the business artifacts, (b) the (macro-)life cycle of the artifacts, (c) the services (a.k.a. tasks) that operate on the artifacts, and (d) the mechanisms whereby services are associated to the ar-

[*]A. Deutsch and V. Vianu were supported in part by the National Science Foundation under award IIS-0415257.

[†]R. Hull was supported in part by the National Science Foundation under awards IIS-0415195, CNS-0613998 and IIS-0812578. Part of this research performed while Hull was at Bell Labs Research, Alcatel-Lucent Technologies.

ICDT 2009, March 23–25, 2009, Saint Petersburg, Russia.
Copyright 2009 ACM 978-1-60558-423-2/09/0003 ...$5.00

tifacts. In this paper each artifact type will involve a family of attributes along with one relational state (a simple generalization permits multiple relational states). Intuitively, the artifact starts with just a few of the attributes defined (initialized), and the invoked services fill in, or overwrite, the artifact's attributes and relational state as the artifact moves through its lifecycle. As illustrated in [7], the macro-lifecycle of the artifacts can be used to capture the key business-relevant *stages* (or states) in the lifecycle of an artifact. Stage transitions are typically specified using a finite-state machine, often with a handful to tens of states.[1] In practice, the primary operations of a business may involve tens of artifacts for small-to-medium-size businesses, and hundreds of artifacts for large businesses.

The current paper specifies services and their association to artifacts in a declarative manner, using input parameters, output parameters, pre-conditions, and post-conditions. This model is inspired by the model of [6], and more broadly by the field of semantic web services [40] (where post-conditions are called "conditional effects"). The use of post-conditions permits non-determinism in the outcome of a service, as is typically the case, for example, in a business process service in which a human makes a final determination about the value of an attribute while satisfying certain constraints. Also following [6], in this paper the movement of artifacts from one stage to another is specified declaratively, in our case via state update rules (known as condition-action rules in [6]). The model of [6] permits only attributes, and the analysis focuses only on whether these attributes are defined or undefined (so their value is abstracted away). In the current paper we handle both attributes and relational states. The service and property specifications manipulate the data values they hold, and can compare them according to a dense linear order. Further, we include here a static database which can be accessed (but not updated) during the processing of artifacts by the services' state update rules and pre- and post-conditions.

The workflow model used in this paper is illustrated with a running example that models a scenario where a manufacturer fills customer purchase orders, negotiating the price of each item on a case-by-case basis. The example is introduced in Example 2.4 and presented in full in the appendix.

This paper considers the problem of statically verifying whether all runs of an artifact system satisfy desirable correctness properties expressed in a first-order extension of linear-time temporal logic called LTL-FO . This language can express a wide variety of properties pertaining to the consistency of the specification (e.g two Boolean flags are mutually exclusive at every step of the business process), or to the policies implemented by a business process. For instance, in the running example, one wishes to guarantee the following:

> If the customer's status is not *preferred* and the credit rating is worse than *good*, then before accepting an order for a product with final negotiated price lower than the manufacturer's desired price, explicit approval from a human executive must be requested.

We also show how other common analysis tasks for business processes can be reduced to verification of LTL-FO properties.

The main technical challenge to static verification lies in the fact that the artifact systems studied here are infinite-state systems, as the domain of the data is infinite. In the general case, testing correctness of properties is undecidable. We map the boundaries of decidability for the verification problem, and identify a class of restrictions such that (a) artifact systems and properties that lie within this class are decidable, and (b) relaxing any of the restrictions leads to undecidability. For the restricted setting, the decision problem is PSPACE-complete. As will be seen, the running example obeys the restrictions, thus illustrating that they are not prohibitive for practical scenarios.

Further related work. As mentioned above, artifacts and related notions have been discussed in the research literature for several years now. The specific notion of artifact, along with specification of key stages in its life-cycle, was first introduced in [44], and subsequently studied. from both practical and theoretical perspectives, in [4, 5, 17, 7, 6, 27, 28, 30, 38, 34, 36, 47]. Some key roots of the artifact-centric approach are present in adaptive objects [35], adaptive business objects [41], business entities, and "document-driven" workflow [48]. The notion of documents as in document engineering [29] is focused on certain aspects of artifacts, namely the artifact data itself and how it can be used to facilitate communication between sub-organizations in the course of workflow processing. The Vortex workflow framework [32, 24, 31] is also data centric, and provides a declarative framework for specifying if and when workflow services are to be applied to a given artifact. More recently, [2] has studied automatic verification in the context of workflow based on Active XML documents.

Work on formal analysis of artifact-centric business processes in restricted contexts has been reported in [6, 27, 28]. Properties investigated in these studies include reachability [27, 28], general temporal constraints [28], and the existence of complete execution or dead end [6]. Citations [27, 28] are focused on an essentially procedural version of artifact-centric workflow, and [6] is the first to study a declarative version. For the variants considered in each paper, verification is generally undecidable; decidability results were obtained when rather severe restrictions are placed, e.g., restricting all guards on state transitions to be "true" [27], restricting to bounded domains [28, 6], or restricting the language for conditions to refer only to artifacts (and not their attribute values) [28]. None of the above papers permit an arbitrary external database, separate from the artifacts, in their frameworks.

The OWL-S proposal [40, 39] describes the semantics of services with input, output, pre-condition, and post-conditions (known there as *conditional effects*). In that work, the pre-conditions and effects refer to *fluents*, that is predicates whose values can change over time. These are used to model evolving databases, for instance for flight reservations, bank accounts, and warehouse inventories. The declarative artifact-centric approach to workflow modeling used here is closely related to that of semantic web services in general, and OWL-S in particular.

Static analysis for semantic web services is considered in [42], but

[1] We note the difference between the "relational states" associated with artifacts in the current paper, and the "states" of state machines used to specify an artifact lifecycle as in [7, 44]. The relational states here are part of the core data maintained in an artifact, and can hold sets of tuples; in contrast the lifecycle state machine can be in just one state at a time. Note that in the current paper, either an attribute or a relational state of an artifact can be used to record the state-machine state that an artifact is currently in.

in a context restricted to finite domains.

The work [21] studies static verification of data-driven Web services that interact with external users through a Web browser interface and generate Web pages dynamically by queries on an underlying database. The study identifies decidable cases of the problem of verifying if all runs of a Web service satisfy a correctness property specified as a sentence in LTL-FO , the language of first order logic extended with linear-time temporal logic operators, which we adopt also in this paper. Similar extensions have been previously used in various contexts [25, 1, 46, 21, 23]. The model studied in [21] extends prior formalisms for specifying electronic commerce applications with additional features that turn out to be essential for describing Web applications. Its immediate ancestor is the ASM transducer [46, 45], a more remote one is the relational transducer [3]. The artifact system model could conceptually (if not naturally) be encoded into the extended ASM transducer model of [21]. However, this would not yield a proof of the results in this paper, because business process modeling requires two non-trivial extensions. First, runs of artifact systems must be allowed to use infinitely many domain values in order to model arbitrary inputs from external users or partially specified processes described by pre- and post-conditions (unlike transducers, where the domain of each run is restricted to the active domain of the finite database). Second, the underlying domain is ordered, which turns out to be a key feature in writing practically useful pre- and post-conditions. These extensions render the proof of decidability of verification considerably more involved.

In the broader context of verification, data-centric business processes can be viewed as a special case of infinite-state systems. Over the last decade, much work in the verification community has focused on extending classical model checking to infinite-state systems (e.g., see [15] for a survey). However, in much of this work the emphasis is on studying recursive control rather than data, which is either ignored or finitely abstracted. More recent work has been focusing specifically on data as a source of infinity. This includes augmenting recursive procedures with integer parameters [11], rewriting systems with data [12, 10], Petri nets with data associated to tokens [37], automata and logics over infinite alphabets [14, 13, 43, 18, 33, 8, 10], and temporal logics manipulating data [18, 19]. However, the restricted use of data and the particular properties verified have limited applicability to database-driven systems such as data-centric business artifacts.

Paper outline. Our model of artifact systems and the language LTL-FO are introduced in Section 2, together with our running example. Section 3 states the restrictions needed for decidability of verification, and provides the main decidability result. In Section 4, the restrictions are shown to be tight by considering several relaxations that lead to undecidability of verification. Applications of the main verification results to other business process analysis tasks are provided in Section 5. We end with brief conclusions. An appendix contains the full running example.

2. FRAMEWORK

We introduce here our model and basic definitions and notation.

We assume fixed an infinite, countable domain D equipped with a total dense order $\leq$ with no endpoints. As usual, a database schema $\mathcal{D}$ consists of a finite set of relation symbols with specified arities. The arity of relation R is denoted $a(R)$. An instance, or interpretation, over a database schema, is a mapping associating to each relation symbol R of the schema a finite relation over D, of arity $a(R)$. We assume familiarity with First-Order logic (FO) over database schemas. Given a schema $\mathcal{D}$, $\mathcal{L}_\mathcal{D}$ denotes the set of FO formulas over $\mathcal{D} \cup \{=, \leq\}$ ($=$ and $\leq$ are built-in relations over D). In addition to relations, FO formulas may use a finite set of *constants*, consisting of elements of D. As customary in relational calculus, constants are always interpreted as themselves (this differs from constants in classical logic). If $\varphi(\bar{x})$ is an FO formula with free variables $\bar{x}$, and $\bar{u}$ is a tuple over D of the same arity as $\bar{x}$, we denote by $\varphi(\bar{u})$ the sentence obtained by substituting $\bar{u}$ for $\bar{x}$ in $\varphi(\bar{x})$. Note that, since D is infinite, an FO formula $\varphi(\bar{x})$ may be satisfied by infinitely many tuples $\bar{u}$ over D (so may define an infinite relation). Finiteness and effective evaluation can be guaranteed by using the *active domain semantics*, in which the domain is restricted to the set of elements occurring in the given instance (sometimes augmented with a specified finite set of constants in D, by default empty). For an instance I, we denote its active domain by *adom*(I). We assume unrestricted semantics unless otherwise specified.

The artifact model uses a specific notion of class, schema and instance, defined next.

DEFINITION 2.1. *An* artifact class *is a pair* $\mathcal{C} = \langle R, S\rangle$ *where* R *and* S *are two relation symbols. An* instance *of* $\mathcal{C}$ *is a pair* $C = \langle \mathsf{R}, \mathsf{S}\rangle$, *where* (i) R, *called* attribute relation, *is an interpretation of* R *containing exactly one tuple over* D, *and* (ii) S, *called* state relation, *is a finite interpretation of* S *over* D.

We also refer to an *artifact instance of class* $\mathcal{C}$ as *artifact instance*, or simply *artifact* when the class is clear from the context or irrelevant.

DEFINITION 2.2. *An* artifact schema *is a tuple*

$$\mathcal{A} = \langle \mathcal{C}_1, \ldots, \mathcal{C}_n, \mathcal{DB}\rangle$$

where each $\mathcal{C}_i = \langle R_i, S_i\rangle$ *is an artifact class,* $\mathcal{DB}$ *is a relational schema, and* $\mathcal{C}_i$, $\mathcal{C}_j$, *and* $\mathcal{DB}$ *have no relation symbols in common for* $i \neq j$.

By slight abuse, we sometimes identify an artifact schema $\mathcal{A}$ as above with the relational schema

$$\mathcal{DB}_\mathcal{A} = \mathcal{DB} \cup \{R_i, S_i \mid 1 \leq i \leq n\}.$$

An instance of an artifact schema is a tuple of class instances, each corresponding to an artifact class, plus a database instance:

DEFINITION 2.3. *An* instance *of an artifact schema*

$$\mathcal{A} = \langle \mathcal{C}_1, \ldots, \mathcal{C}_n, \mathcal{DB}\rangle$$

is a tuple $A = \langle C_1, \ldots, C_n, DB\rangle$, *where* C_i *is an instance of* $\mathcal{C}_i$ *and* DB *is an instance of* $\mathcal{DB}$ *over* D.

Again by slight abuse, we identify each instance

$$A = \langle C_1, \ldots, C_n, DB\rangle$$

of $\mathcal{A}$ with the relational instance $DB \cup \{\mathsf{R_i}, \mathsf{S_i} | 1 \leq i \leq n\}$ over schema $\mathcal{DB}_{\mathcal{A}}$. Let $\mathcal{A}$ be an artifact schema and $\mathcal{DB}_{\mathcal{A}}$ its relational schema. Given an artifact instance over $\mathcal{A}$, the semantics of formulas in $\mathcal{L}_{\mathcal{A}}$ is the standard semantics on the associated relational instance over $\mathcal{DB}_{\mathcal{A}}$.

EXAMPLE 2.4. We illustrate the expressive power of the artifact model by specifying a scenario where a manufacturer fills customer purchase orders, negotiating the price of each line item on a case-by-case basis. We focus on two artifacts manipulated by the negotiation process, ORDER and QUOTE.

During the workflow, the customer repeatedly adds new line items into (or updates existing ones in) the purchase order modeled by the ORDER artifact. Each line item specifies a product and its quantity. Every tentative line item spawns a negotiation process, in which manufacturer and customer complete rounds of declaring ask and bid prices, until agreement is reached or the negotiation fails. The prices at every round are stored in the QUOTE artifact, which also holds the manufacturer's initially desired price, the lowest bid he is willing to entertain, and the final negotiated price. Once the negotiation on a tentative line item succeeds, its outcome is scrutinized by a human executive working for the manufacturer. Upon the executive's approval, the line item is included into the purchase order. During the negotiation, the manufacturer consults an underlying database, which lists information about available products (e.g. manufacturing cost) and about customers (e.g. credit rating and status).

The corresponding artifact system $\Gamma_{ex} = \langle \mathcal{A}, \Sigma \rangle$ is partially described here and in Example 2.9 (see Appendix for the full specification). For convenience, we allow an artifact class to have several state relations. This can be easily simulated with a single state relation (see also Appendix).

The artifact schema is $\mathcal{A} = \langle ORDER, QUOTE, \mathcal{DB} \rangle$, detailed as follows.

$\mathcal{DB} = \langle \text{PRODUCT}, \text{CUSTOMER} \rangle$ is the database schema, where:

- PRODUCT$(prod_id, manufacturing_cost, min_order_qty)$ lists product manufacturing cost and minimum order quantity, and
- CUSTOMER$(customer_id, status, credit_rating)$ lists customer status and credit rating.

$ORDER = \langle R_O, \mathsf{line\_items}, \mathsf{in\_process}, \mathsf{done} \rangle$
is the artifact class containing the information about a customer's order.

- $R_O(order\#, customer\_id, need\_by, li\_prod, li\_qty)$
is the attribute relation holding the order number, the identifier of the customer who placed the order, the day it is needed by. The role of attributes *li_prod* and *li_qty* is described later.
- line_items$(prod_id, qty)$
is a state relation that acts as a "shopping cart" holding the collection of line items requested so far.
- in_process and done
are nullary state relations (Boolean flags) keeping track of the stage the artifact is in. [2]

The intention is that, in stage in_process, the customer repeatedly updates the shopping cart by filling an individual, tentative line item into attributes *li_prod* and *li_qty*. Subsequently, this line item is inserted into line_items provided the price negotiation succeeds. When the customer completes the purchase order, the ORDER artifact transitions to stage done.

$$QUOTE = \langle R_Q, \mathsf{li\_quotes}, \mathsf{idle}, \mathsf{desired\_price\_calc}, \mathsf{negotiation}, \mathsf{approval\_pending}, \mathsf{archive} \rangle$$

is the artifact class modeling quotes, with:

- $R_Q(\, order\#, desired\_price, lowest\_acceptable\_price, ask, bid, final\_price, approved, li\_prod, li\_qty, manufacturing\_cost)$
is the attribute relation.
- li_quotes$(prod_id, qty, price)$
is a state relation storing the line item with the final negotiated price quotes.
- idle, desired_price_calc, negotiation, approval_pending, and archive
are nullary state relations keeping track of the stage the artifact is in.

When inactive, the QUOTE artifact is in state idle, but moves to desired_price_calc as soon as the customer fills in the product id and quantity of a line item. In this stage, *desired_price* attribute is set (from the manufacturer's point of view), possibly taking into account the *need_by* date attribute in the corresponding ORDER artifact and the manufacturing cost listed in the PRODUCT database. During the ensuing negotiation stage, the ask and bid prices are repeatedly set (in attribute *ask* by the manufacturer, respectively *bid* by the customer) until a final price is established and recorded in attribute *final_price*, or the negotiation fails. Final prices may require approval by a human executive who works for the manufacturer. While approval is awaited, the QUOTE artifact is in stage approval_pending. Approval is granted by setting Boolean attribute *approved*. Approved final prices are then archived in state relation li_quotes (while the QUOTE artifact is in stage archive). □

We now define the syntax of services. It will be useful to associate to each attribute relation R of an artifact schema $\mathcal{A}$ a fixed sequence $\bar{x}_R$ of distinct variables of length $a(R)$.

DEFINITION 2.5. *A service σ over an artifact schema $\mathcal{A}$ is a tuple $\sigma = \langle \pi, \psi, \mathcal{S} \rangle$ where:*

[2] Artifact class ORDER illustrates an extension of Definition 2.1 that allows several state relations. This extension is for convenience only: it is easy to show a reduction from multiple-state artifacts to single-state artifacts that preserves our decidability result.

- π, *called* pre-condition, *is a sentence in* $\mathcal{L}_{\mathcal{A}}$*;*
- ψ, *called* post-condition, *is a formula in* $\mathcal{L}_{\mathcal{A}}$*, with free variables*
$\{\bar{x}_R \mid R \text{ is an attribute relation of a class in } \mathcal{A}\}$*;*
- $\mathcal{S}$ *is a set of* state rules *containing, for each state relation* S *of* $\mathcal{A}$*, one, both or none of the following rules:*
  - $S(\bar{x}) \leftarrow \phi_S^+(\bar{x})$*;*
  - $\neg S(\bar{x}) \leftarrow \phi_S^-(\bar{x})$*;*

  where $\phi_S^+(\bar{x})$ *and* $\phi_S^-(\bar{x})$ *are* $\mathcal{L}_{\mathcal{A}}$*-formulas with free variables* $\bar{x}$ *s.t.* $|\bar{x}| = a(S)$.

DEFINITION 2.6. *An* artifact system *is a pair* $\Gamma = \langle \mathcal{A}, \Sigma \rangle$*, where* $\mathcal{A}$ *is an artifact schema and* Σ *is a non-empty set of services over* $\mathcal{A}$*.*

We next define the semantics of services. We begin with the notion of possible successor of a given artifact instance with respect to a service.

DEFINITION 2.7. *Let* $\sigma = \langle \pi, \psi, \mathcal{S} \rangle$ *be a service over artifact schema* $\mathcal{A}$*. Let* A *and* A' *be instances of* $\mathcal{A}$*. We say that* A' *is a* possible successor *of* A *with respect to* σ *(denoted* $A \xrightarrow{\sigma} A'$*) if the following hold:*

1. $A \models \pi$*;*
2. $A'|\mathcal{DB} = A|\mathcal{DB}$*;*
3. *if* $\bar{u}_R$ *is the content of the attribute relation* R *of* $\mathcal{A}$ *in* A'*, then* A *satisfies the post-condition* ψ *where* $\bar{x}_R$ *is replaced by* $\bar{u}_R$ *for each* R*;*
4. *for each state relation* S *of* $\mathcal{A}$ *and tuple* $\bar{u}$ *over* $adom(A)$ *of arity* $a(S)$*,* $A' \models S(\bar{u})$ *iff*
$$A \models \quad (\phi_S^+(\bar{u}) \wedge \neg\phi_S^-(\bar{u})) \vee (S(\bar{u}) \wedge \phi_S^+(\bar{u}) \wedge \phi_S^-(\bar{u})) \\ \vee (S(\bar{u}) \wedge \neg\phi_S^+(\bar{u}) \wedge \neg\phi_S^-(\bar{u}))$$
where $\phi_S^+(\bar{u})$ *and* $\phi_S^-(\bar{u})$ *are interpreted under active domain semantics, and are taken to be false if the respective rule is not provided.*

Note that, according to (2) in Definition 2.7, services do not update the database contents (thus, the database contents is fixed throughout each run, although it may of course be different across runs). Instead, the data that is updatable throughout a run is carried by the artifacts themselves, as attribute and state relations. This distinction between the static and updatable portions of the data is convenient for technical reasons, as it is used in formulating the restrictions needed for verification (see Section 3). Note that, if so desired, one can make the entire database updatable by turning it into a state. Also observe that the distinction between state and database is only conceptual, and does not preclude implementing all relations within the same DBMS.

We next define the notion of run of an artifact system $\Gamma = \langle \mathcal{A}, \Sigma \rangle$. An *initial instance* of Γ is an artifact instance over $\mathcal{A}$ whose states are empty.

DEFINITION 2.8. *A* run *of an artifact system* $\Gamma = \langle \mathcal{A}, \Sigma \rangle$ *is an infinite sequence* $\rho = \{\rho_i\}_{i \geq 0}$ *of artifact instances over* $\mathcal{A}$ *(also called* configurations*) such that:*

- ρ_0 *is an initial instance of* Γ*;*
- *for each* $i \geq 0$*,* $\rho_i \xrightarrow{\sigma} \rho_{i+1}$ *for some* $\sigma \in \Sigma$*.*

A pre-run *is a finite sequence* $\{\rho_i\}_{0 \leq i \leq n}$ *satisfying the same conditions as above for* $i < n$*. We say that a pre-run is* blocking *if its last configuration has no possible successor.*

EXAMPLE 2.9. Continuing Example 2.4, we show how to model the operations allowed on artifacts by the set Σ of available services. Due to space constraints, we relegate most of the specification of Σ to the appendix, focusing here on the service that models the negotiation process. To illustrate the artifact model's natural ability to specify processes at different levels of abstraction, we describe the negotiation process at two levels. In a first, coarser cut, the process is abstracted as service $abstract\_negotiation = \langle \pi^{an}, \psi^{an}, \mathcal{S}^{an} \rangle$ about which we only know that the final price is reached when the ask and bid prices coincide, and that it is guaranteed to lie between the allowed margins stored in attributes *desired_price* and *lowest_acceptable_price* of artifact QUOTE. The specification of this service is relatively simple and given in the appendix. Alternatively, we show below service $refined\_negotiation = \langle \pi^{rn}, \psi^{rn}, \mathcal{S}^{rn} \rangle$ which refines the negotiation process all the way to the level of individual negotiation rounds, each of which sets the current ask and bid prices.

Conventions. We adopt the following conventions:

(i) We model uninitialized attributes by setting them to the reserved constant ω.

(ii) We model Boolean states by nullary state relations, and drop the parentheses from atoms using them: $\mathsf{S}()$ becomes S. We assume the usual encoding of *true* as the singleton nullary relation, and *false* as the empty nullary relation. In particular, all Boolean states are initially *false* (since all state relations are initially empty).

(iii) For convenience, we use the following syntactic sugar for post-conditions: we write post-conditions as non-Horn rules $h(\bar{x}) := b(\bar{y})$ where the *head* h is a conjunction of atoms over attribute relations in $\mathcal{A}$, with variables $\bar{x}$, and the *body* b is a formula in $\mathcal{L}_{\mathcal{A}}$ with free variables $\bar{y}$, where $\bar{y} \subseteq \bar{x}$. The semantics is that whenever $A \xrightarrow{\sigma} A'$ holds, $A' \models h(\bar{u})$ for some tuple $\bar{u}$, and $A \models b(\bar{u}|\bar{y})$. Moreover, artifact relations not mentioned in h remain unchanged. Clearly, this syntactic sugar can be simulated by the official post-conditions, and conversely.

We now describe service $refined\_negotiation = \langle \pi^{rn}, \psi^{rn}, \mathcal{S}^{rn} \rangle$:

The pre-condition

$$\pi^{rn} \doteq \mathsf{negotiation}$$

ensures that the service applies only as long as the Boolean state flag $\mathsf{negotiation}$ is set in the QUOTE artifact.

Post-condition ψ^{rn} is given as

$$\begin{aligned}
&R_Q\ (o,d,l,a,b,f,app,p,q,m) := \\
&\quad (\exists a', b'\ R_Q(o,d,l,a',b',\omega,app,p,q,m) \wedge a' \neq b' \wedge \\
&\quad l \leq a \leq a' \wedge b' \leq b \wedge f = \omega) \\
&\quad \vee \\
&\quad (R_Q(o,d,l,a,b,\omega,app,p,q,m) \wedge a = b = f).
\end{aligned}$$

According to the first disjunct, the negotiation is well-formed, i.e. the bid never exceeds the *ask* price, and in each round, asking prices a never increase while bids b never decrease. Moreover, as long as *ask* and *bid* price differ, the final price remains undefined (equal to ω). Notice that the values of a and b are otherwise unconstrained, being simply drawn from the infinite domain. This reflects the fact that they are external input from the manufacturer, respectively customer. The second disjunct states that once *ask* price a and *bid* price b coincide, the final price f is automatically set to the common value.

$\mathcal{S}^{rn}$ contains rules that, upon detecting successful negotiation, switch the QUOTE artifact to stage approval_pending if the customer does not enjoy preferred status with excellent credit. If he does, then the approval is short-circuited and the QUOTE goes directly to stage archive. The negotiation is successful when the *ask* and *bid* prices agree.

approval_pending $\leftarrow$
$(\exists o,d,l,a,f,app,p,q,m\ R_Q(o,d,l,a,a,f,app,p,q,m) \wedge$
$\exists c,n\ R_O(o,c,n,p,q) \wedge$
$\neg\text{CUSTOMER}(c, \text{"}preferred\text{"}, \text{"}excellent\text{"}))$

archive $\leftarrow$
$(\exists o,d,l,a,f,app,p,q,m\ R_Q(o,d,l,a,a,f,app,p,q,m) \wedge$
$\exists c,n\ R_O(o,c,n,p,q) \wedge$
$\text{CUSTOMER}(c, \text{"}preferred\text{"}, \text{"}excellent\text{"}))$

¬negotiation $\leftarrow (\exists o,d,l,a,f,app,p,q,m$
$R_Q(o,d,l,a,a,f,app,p,q,m))$

Note that state flag negotiation must be set before the state update rules execute (since pre-condition π^{rn} is satisfied). If neither of the state rule bodies is satisfied, then according to the possible successor semantics, the negotiation flag remains set, enabling another negotiation round. □

One of the points illustrated by Example 2.9 is that the artifact model is particularly well-suited for expressing a wide spectrum of abstraction levels desired in specification. This is shown by the two specifications of the negotiation process, one refining it down to individual rounds, the other abstracting it to an atomic sub-task with a post-condition on its outcome. In practice, the motivation for abstraction ranges from lack of information about an external process provided by an autonomous third party as a black box with pre- and post-execution guarantees, to modeling non-deterministic processes governed by chance or human agents rather than by program. There are also technical reasons, such as the undecidability of verification in the presence of arithmetic (as is the case in many settings, including ours). In all these cases, abstracted sub-processes can be naturally modeled as services, leveraging the non-determinism in their post-conditions.

In order to specify temporal properties of runs, we use an extension of linear-time temporal logic (LTL). Recall that LTL is propositional logic augmented with temporal operators such as **X** (next), **U** (until), **G** (always) and **F** (eventually). Essentially, the extension we use, denoted LTL-FO, is obtained from LTL by replacing propositions by FO statements about individual artifact instances in the run. The different statements may share variables that are universally quantified at the end. Similar extensions have previously been used in various contexts [25, 1, 46, 21, 23].

DEFINITION 2.10. *The language LTL-FO (first-order linear-time temporal logic) is obtained by closing FO under negation, disjunction, and the following formula formation rule: If φ and ψ are formulas, then $\mathbf{X}\varphi$ and $\varphi\mathbf{U}\psi$ are formulas. Free and bound variables are defined in the obvious way. The* universal closure *of an LTL-FO formula $\varphi(\bar{x})$ with free variables $\bar{x}$ is the formula $\forall\bar{x}\varphi(\bar{x})$. An LTL-FO sentence is the universal closure of an LTL-FO formula.*

Let $\mathcal{A}$ be an artifact schema. An LTL-FO sentence over $\mathcal{A}$ is one where each FO component is over $\mathcal{DB}_{\mathcal{A}}$. The semantics of LTL-FO formulas is standard, and we describe it informally. Let $\Gamma = \langle \mathcal{A}, \Sigma \rangle$ be an artifact system, and $\forall\bar{x}\varphi(\bar{x})$ an LTL-FO sentence over $\mathcal{A}$. The artifact system Γ satisfies $\forall\bar{x}\varphi(\bar{x})$ iff every run of Γ satisfies it. Let $\rho = \{\rho_i\}_{i\geq 0}$ be a run of Γ, and let $\rho_{\geq j}$ denote $\{\rho_i\}_{i\geq j}$, for $j \geq 0$. Note that $\rho = \rho_{\geq 0}$. The run ρ satisfies $\forall\bar{x}\varphi(\bar{x})$ iff for each valuation ν of $\bar{x}$ in D, $\rho_{\geq 0}$ satisfies $\varphi(\nu(\bar{x}))$. The latter is defined by structural induction on the formula. Satisfaction of an FO sentence ψ by ρ_i is defined in the obvious way. The semantics of Boolean operators is standard. The meaning of the temporal operators **X, U** is the following (where $\models$ denotes satisfaction and $j \geq 0$):

- $\rho_{\geq j} \models \mathbf{X}\varphi$ iff $\rho_{\geq j+1} \models \varphi$,
- $\rho_{\geq j} \models \varphi\mathbf{U}\psi$ iff $\exists k \geq j$ such that $\rho_{\geq k} \models \psi$ and $\rho_{\geq l} \models \varphi$ for $j \leq l < k$.

Observe that the above temporal operators can simulate all commonly used operators, including **F** (eventually), **G** (always), and **B** (before, which requires its first argument to hold before its second argument fails). Indeed, $\mathbf{F}\varphi \equiv$ *true* $\mathbf{U}\ \varphi$, $\mathbf{G}\varphi \equiv \neg\mathbf{F}(\neg\varphi)$, and $\varphi\mathbf{B}\psi \equiv \neg(\neg\varphi\mathbf{U}\neg\psi)$. We use the above operators as shorthand in LTL-FO formulas whenever convenient.

Note that, as customary in verification, LTL-FO properties of artifact systems concern exclusively their infinite runs. Thus, blocking finite pre-runs are ignored. In particular, if an artifact system has only blocking pre-runs (so no proper run) then it vacuously satisfies all LTL-FO formulas. For this and other reasons, one may wish to know if, for a given artifact system *(i)* all of its pre-runs are blocking, or *(ii)* there exists a blocking pre-run. We consider decidability of these questions at the end of Section 3 (Corollary 3.4) and Section 4 (Corollary 4.3).

EXAMPLE 2.11. We illustrate desirable properties for the artifact system Γ_{ex} in Example 2.9. These properties pertain to the global evolution of Γ_{ex}, as well as to the consistency of its specification. One such consistency property requires the state flags in_process and done in class ORDER to always be mutually exclusive:

$$\mathbf{G}(\neg(\mathsf{in\_process} \wedge \mathsf{done})).$$

A more data-dependent consistency property requires line item quotes archived in state li_quotes of the QUOTE artifact to pertain only to tentative line items previously input by the customer (into attributes *li_prod* and *li_qty* of the ORDER artifact), and which underwent successful negotiation and approval. Successful negotiation occurs when *ask, bid* and *final_price* coincide and the QUOTE artifact is in state archive:

$$\begin{array}{l} \forall pid, qty, prc \\ \quad \mathbf{G}\ ((\exists o,c,n\ R_O(o,c,n,pid,qty) \wedge \\ \qquad \exists d,l,m\ R_Q(o,d,l,prc,prc,prc,\text{"}yes\text{"},pid,qty,m) \wedge \\ \qquad \text{archive}) \\ \qquad\qquad \mathbf{B} \\ \qquad \neg\text{li\_quotes}(pid,qty,prc)) \end{array}$$

Notice the use of the *before* operator **B** (requiring its first argument to hold before its second argument fails).

The following property is more semantic in nature, capturing part of the manufacturer's business model. It requires that if the customer's status is not *"preferred"* and the credit rating is worse than *"good"*, then before archiving a line item with final negotiated price lower than the manufacturer's desired price, explicit approval from a human executive must have been requested. We assume the following ordering on the constants indicating the credit rating: $\text{"}poor\text{"} < \text{"}fair\text{"} < \text{"}good\text{"} < \text{"}excellent\text{"}$.

$$\begin{array}{l} \varphi_3 : \\ \quad \forall o,c,n,p,q,d,l,f,m,s,r \\ \qquad \mathbf{G}\ ((R_O(o,c,n,p,q)\ \wedge \text{in\_process}\ \wedge \text{negotiation} \wedge \\ \qquad\quad R_Q(o,d,l,f,f,f,\omega,p,q,m)\ \wedge\ f < d \wedge \\ \qquad\quad CUSTOMER(c,s,r) \wedge s \neq \text{"}preferred\text{"} \wedge \\ \qquad\quad r < \text{"}good\text{"}) \rightarrow \\ \qquad\quad (\text{approval\_pending} \\ \qquad\qquad\quad \mathbf{B} \\ \qquad\quad \neg(\text{archive} \wedge \text{li\_quotes}(p,q,f)))) \end{array}$$

Note that φ_3 involves both artifacts and the underlying database. If the negotiation process is described by service *refined_negotiation*, then the property happens to be satisfied: indeed, recall from its state rules that this service requests approval whenever the customer's status is not preferred and his credit rating is not excellent. In particular, this applies to customers whose rating is worse than good, according to the above ordering of credit ratings. □

A detailed specification of all services involved in our running example can be found in the appendix.

3. DECIDABLE VERIFICATION

In this section we establish the main decidability result on verification of artifact systems.

It is easily seen that satisfaction of an LTL-FO formula by an artifact system is generally undecidable, using Trakhtenbrot's theorem. To obtain decidability, we introduce a restricted class of artifact systems and LTL-FO properties, called *guarded*. This is the analog to artifact systems of the input-boundedness restriction, first introduced by Spielmann in the context of ASM transducers [46], and subsequently used for Web service verification [21]. The guarded restriction mainly requires a form of bounded quantification in formulas used in state update rules and LTL-FO properties, together with some additional restrictions. The guarded restriction is formulated as follows.

DEFINITION 3.1. *Let $\Gamma = \langle \mathcal{A}, \Sigma \rangle$ be an artifact system. The set of guarded FO formulas over $\mathcal{A}$ is obtained by replacing in the definition of FO the quantification formation rule by the following:*

- *if φ is a formula, α is an atom using an attribute relation of some artifact of $\mathcal{A}$, $\bar{x} \subseteq free(\alpha)$, and $\bar{x} \cap free(\beta) = \emptyset$ for every state atom β in φ, then $\exists \bar{x}(\alpha \wedge \varphi)$ and $\forall \bar{x}(\alpha \rightarrow \varphi)$ are formulas.*

An artifact system is guarded iff all formulas used in the state rules of its services are guarded, and all pre-and-post conditions are $\exists^*$FO formulas[3] in which all state atoms are ground (i.e. contain only constants). An LTL-FO sentence over $\mathcal{A}$ is guarded iff all of its FO components are guarded.

Note that, in addition to the usual bounded quantification conditions, Definition 3.1 places the restriction $\bar{x} \cap free(\beta) = \emptyset$ for every state atom β occurring in the scope of a quantification of x in a guarded formula. This says that state atoms can only contain constants or free variables in guarded formulas. Together with the fact that state atoms must appear ground in pre-and-post conditions, this places strong restrictions that considerably limit the use of state information. Unfortunately, both restrictions are needed for decidability of verification. On the positive side, as illustrated by our running example, guarded artifact systems appear to remain powerful enough to model significant applications.

EXAMPLE 3.2. The artifact system Γ_{ex} in our running example is guarded. This includes the complete specification in Appendix, which shows that the guardedness restriction still offers significant expressive power. For instance, notice that post-condition ψ^{rn} is an $\exists^*$FO formula with no non-ground state atoms (trivially so, as it mentions no state at all). In addition, in all state rules the quantified variables appear guarded by atoms using attribute relations R_O or R_Q. No quantified variables appear in any state atom because no such atoms are mentioned. See the state rules of service *include_line_item* in Appendix for a less trivial example of guarded state rules. There, state atoms do occur in the rule body, but only with non-quantified variables. All properties listed in Example 2.11 are guarded.

For an example of an unguarded state rule, consider a relaxation of the insertion rule in $\mathcal{S}^{rn}$ demanding that executive approval be short-circuited and the archive flag be set for all preferred customers with better than fair rating:

$$\begin{array}{l} \text{archive} \leftarrow \\ \qquad \exists o,d,l,a,f,app,p,q,m,r \\ \qquad\quad R_Q(o,d,l,a,a,f,app,p,q,m) \wedge \\ \qquad\quad \exists c,n\ R_O(o,c,n,p,q) \wedge \\ \qquad\qquad CUSTOMER(c,\text{"}preferred\text{"},r) \wedge r > \text{"}fair\text{"} \end{array}$$

[3] Note that these formulas do not have to obey the restricted quantification formation rule.

The problem here is that quantified variable r does not appear in any attribute atom (indeed it cannot, since neither R_O nor R_Q have any rating attribute). While our undecidability results in Section 4 imply that not every unguarded rule or property can be equivalently rewritten into a guarded one, it is often possible to do so by slightly modifying the specification, such that it preserves the intended business process semantics, at the cost of widening the attribute relation. This happens to apply here. One can simply extend the attribute schema of R_O to include the customer's rating, in addition to the originally included customer id. The rating attribute would be set at the same time as the customer id attribute. The latter is set by service *initiate_order* in Appendix, using a guarded post-condition that would remain guarded after the proposed extension. □

The main result on decidability of verification for artifact systems is the following.

THEOREM 3.3. *It is decidable, given a guarded artifact system Γ and a guarded LTL-FO formula φ, whether every run of Γ satisfies φ. Furthermore, the complexity of the decision problem is* PSPACE*-complete for fixed arity schemas, and* EXPSPACE *otherwise*[4].

The main challenge in establishing the above result is that artifact systems are *infinite-state systems*, due to the presence of unbounded data. To deal with this, the key idea is to develop a concise, symbolic representation of equivalence classes of runs of Γ, called *pseudoruns*, that retain just the information needed to check satisfaction of φ, and can be generated in PSPACE without explicitly constructing any actual run or database. The high-level structure of the proof is similar to the one for decidability of verification for extended ASM transducers [21]. However, the result for the artifact model substantively extends previous ones in two significant ways: (i) runs of artifact systems may use infinitely many domain values (unlike extended ASM transducers where the domain of each run is restricted to the active domain of the finite database), and (ii) the underlying domain is ordered. These extensions require much more care in developing the pseudorun technique, and render the proof of decidability considerably more difficult. This proof is omitted here.

Finally, we consider the issue of blocking pre-runs. As remarked in Section 2, LTL-FO properties of artifact systems concern only their (infinite) runs and ignore blocking pre-runs. In particular, if an artifact system has only blocking pre-runs (so no proper run) then it vacuously satisfies all LTL-FO formulas. It therefore becomes of interest to know whether all pre-runs of an artifact system are blocking. Moreover, blocking may also be of interest for reasons specific to the application (see also discussion in Section 5). We can show the following.

COROLLARY 3.4. *It is decidable, given a guarded artifact system Γ, whether all pre-runs of Γ are blocking. Furthermore, the complexity is* PSPACE *for fixed-arity schemas, and* EXPSPACE *otherwise.*

Proof: The result follows immediately from Theorem 3.3. Indeed, all pre-runs of Γ are blocking iff Γ has no (infinite) runs iff $\Gamma \models$ *false*. The latter is decidable with the stated complexities by Theorem 3.3. □

One may also wish to know if a given artifact system has *some* blocking pre-run. Interestingly, this turns out to be undecidable for guarded artifact systems (see Corollary 4.3).

4. BOUNDARIES OF DECIDABILITY

In this section we consider several variations of our artifact model and relaxations of the guarded conditions and show that they lead to undecidability of verification. This suggests that the restrictions we presented in order to ensure decidability are quite tight. Due to space constraints, the presentation of the alternative models is informal.

Attributes versus states. We first revisit the distinction between the attribute relation R and the state relation S in artifact classes $C = \langle R, S \rangle$. One might legitimately wonder if the separate treatment is relevant to verification. We next show that this is indeed the case. More precisely, consider a modification of the artifact model where the state S is treated in the same way as R, except that R holds a single tuple while S holds an entire relation. In particular, in the definition of a service using artifact class $C = \langle R, S \rangle$:

- the pre-and-post conditions of the service are $\exists^*$FO formulas using R, S and the database (with S-atoms no longer restricted to be ground as previously);
- as before, the initial value of S is empty;
- there are separate post-condition formulas ψ_R and ψ_S for R and S, defining their contents in the output (R consists, as before, of *one* arbitrary tuple satisfying ψ_R, while S consists of the *set* of tuples satisfying ψ_S, with active domain semantics to guarantee finiteness).

We refer to R as the *tuple attribute set* of C and to S as the *relational attribute set* of C. We refer to such artifact systems as *hybrid-attribute*. Note that, in this model, there are no longer separate state relations. Since there are no states, the guarded restriction on hybrid-attribute services now simply amounts to the $\exists^*$FO form of the pre-and-post conditions. The guarded restriction for LTL-FO properties remains unchanged. We can show the following (the proof is by reduction from the Post Correspondence Problem).

THEOREM 4.1. *It is undecidable, given a guarded hybrid-attribute artifact system Γ and guarded LTL-FO formula φ, whether $\Gamma \models \varphi$. Moreover, this holds even for singleton artifact systems whose relational attribute set consist of a single attribute, and for a fixed LTL-FO formula φ with no variables.*

Relaxing the guarded restrictions. We now consider several relaxations of the guarded restrictions. It turns out that even very small such relaxations lead to undecidability of verification. Specifically, we consider the following: (i) allowing non-ground state atoms in pre-and-post conditions, (ii) allowing state projections in

[4] The best lower bound we know for arbitrary arity schemas is CO-NEXPTIME, shown by reduction from validity of $\forall^*\exists^*$FO sentences, known to be CO-NEXPTIME-complete [9].

state update rules (a simple form of un-guarded quantification), (iii) allowing un-guarded quantification in the LTL-FO property, and (iv) extending LTL-FO with path quantifiers.

We can show that each of the relaxations (i)-(iv) leads to undecidability of verification. The proof of (i) is similar to that of Theorem 4.1. The proofs of (ii) and (iii) are by reduction from the implication problem for functional and inclusion dependencies, known to be undecidable [16]. The proof of (iv) is by reduction from validity of $\exists^*\forall^*$FO sentences, also known to be undecidable [9]. The proofs of (ii)-(iv) can be easily adapted from analogous results obtained for extended ASM transducers [21]. We therefore omit the details.

Functional dependencies. It is natural to ask whether the decidability of verification holds under the assumption that the database satisfies certain integrity constraints. Unfortunately, we show that even simple key dependencies lead to undecidability.

THEOREM 4.2. *It is undecidable, given a guarded singleton artifact system Γ, a set of functional dependencies F over $\mathcal{DB}$, and a guarded LTL-FO sentence φ, whether $\rho \models \varphi$ for every run ρ of Γ on a database satisfying F. Moreover, this holds even if $\mathcal{DB}$ consists of one binary and one unary relation, and F consists of a single key constraint on the binary relation.*

The proof is done by reduction from the PCP, similarly to Theorem 4.1 (details are omitted).

Existence of a blocking pre-run. Recall the question raised in Section 2: does an artifact system have *(i) only* blocking pre-runs, or *(ii) some* blocking pre-run? We showed in Section 3 that *(i)* is decidable for guarded artifact systems (Corollary 3.4). Interestingly, *(ii)* turns out to be undecidable.

COROLLARY 4.3. *It is undecidable, given a guarded artifact system Γ, whether Γ has some blocking pre-run.*

The result is shown similarly to Theorems 4.1 and 4.2, by reduction from the PCP. The key idea is to first search for a match to the PCP (without assurance that the key dependency assumed in Theorem 4.2 is satisfied), and in case of success make continuance of the run contingent upon violation of the dependency. This reduces the existence of a solution to the PCP to the existence of a blocking pre-run.

Order versus successor. Recall that decidability of verification holds under the assumption that the domain D is countable and equipped with a dense, total order $\leq$ with no endpoints. If $\leq$ is replaced by a successor relation on D, verification becomes undecidable. The proof is, again, by reduction from the PCP.

We note that it remains open whether verification remains decidable if some of the assumptions on $\leq$ do not hold, for instance if $\leq$ is not dense.

5. FURTHER APPLICATIONS

We next discuss several problems previously raised in the context of artifact systems, to which our results on verification can be beneficially applied.

Business rules. We consider an extension of the artifact formalism in support of service reuse and customization. In practice, services are often provided by autonomous third-parties, who typically strive for wide applicability and impose as unrestrictive pre-conditions as possible. In contrast, the designer who incorporates third-party services into the business process often requires more control over when these services apply, in the form of more restrictive pre-conditions. Such additional control may also be needed to ensure compliance with business regulations formulated by third parties, independently of the specific application. To address such needs, [6] introduces *business rules*, which are conditions that can be super-imposed on the pre-conditions of existing services without changing their implementation.

We adopt the notion here and formalize it as follows. Given an artifact system $\Gamma = \langle \mathcal{A}, \Sigma \rangle$, we associate a set $\mathcal{B} = \{\beta_\sigma \mid \sigma \in \Sigma\}$ of business rules to the services in Σ. A *business rule* is a sentence in $\mathcal{L}_\mathcal{A}$, just like a service pre-condition.

For instance, we revisit our running example and assume that order shipment is modeled by the *ship* service, whose pre-condition only checks that the ORDER artifact is in state done. We also assume the existence of a *collect_payment* service, which applies when the ORDER is in state done. Finally, we assume that the ORDER artifact is extended with a paid boolean state flag which is set by the *collect_payment* service. Now we wish to super-impose the following business rule, which implements the policy that only platinum customers with excellent credit may get their order shipped before payment is received:

$$\begin{array}{l}\beta_{ship}: \\ \quad \exists o, c, n, p, s, r\ R_O(o, c, n, p, q) \wedge \text{CUSTOMER}(c, s, r) \wedge \\ \quad\quad (s = "platinum" \wedge r = "excellent" \vee \textsf{paid})\end{array}$$

Verification under business rules. The verification problem for artifact system Γ and property φ *under business rules* $\mathcal{B}$, denoted $\Gamma \models_\mathcal{B} \varphi$, means checking that every run of Γ' satisfies φ, where Γ' is obtained by adding each business rule as a pre-condition conjunct to its corresponding service in Γ. We say that a business rule is *guarded* if it is guarded when viewed as a service pre-condition. It follows immediately as a corollary of Theorem 3.3 that verification under $\mathcal{B}$ is decidable if Γ, φ and all business rules in $\mathcal{B}$ are guarded.

A related problem concerns *incremental* verification under business rules. Note that, if $\Gamma \models \varphi$, then $\Gamma \models_\mathcal{B} \varphi$. However, $\Gamma \not\models \varphi$ does *not* imply that $\Gamma \not\models_\mathcal{B} \varphi$. Thus, properties such as reachability of a configuration satisfying some desired property are not inherited when business rules are added. It is of interest whether such properties can be verified incrementally; however, we do not address this here.

Redundant business rules. Towards streamlining the specification, a desirable goal is the removal of redundant business rules. This involves checking whether, given an artifact system $\Gamma = \langle \mathcal{A}, \Sigma \rangle$, a new business rule β associated to some service $\sigma \in \Sigma$ has any effect on Γ, i.e. excludes at least one of its runs. The latter problem amounts to verifying that at any point in a run of Γ, the pre-condition π of σ implies β: $\Gamma \models \mathbf{G}(\pi \rightarrow \beta)$. If Γ is guarded, and β is guarded in the sense of guarded FO components of LTL-FO properties, then, again as a corollary of Theorem 3.3, checking if β has an effect on Γ is decidable. Indeed, if π is guarded, then we have $\pi \doteq \exists \bar{x}\ f(\bar{x})$ with f a quantifier-free formula in $\mathcal{L}_{\mathcal{A}}$. Then

$$\begin{aligned}&\Gamma \models \mathbf{G}(\exists \bar{x} f(\bar{x}) \rightarrow \beta) \text{ iff } \Gamma \models \mathbf{G}(\forall \bar{x} \neg f(\bar{x}) \vee \beta)\\ &\text{iff } \Gamma \models \underbrace{\forall \bar{x}\ \mathbf{G}(\neg f(\bar{x}) \vee \beta)}_{\varphi}\end{aligned}$$

where φ is a guarded LTL-FO property if β is. For example, β_{ship} above is guarded.

Redundant attributes. Another design simplification consists of redundant attribute removal, a problem raised in [6]. We formulate this as follows. We would like to test whether there is a way to satisfy a property φ of runs without using one of the attributes, say a, of artifact A. Checking redundancy of a reduces to the following verification problem:

$$\Gamma \not\models \underbrace{\varphi \rightarrow \mathbf{F}(\exists \bar{x} \exists a\ R_A(\bar{x}, a) \wedge a \neq \omega)}_{\varphi'}$$

where we assume wlog that a is last in A's attribute relation R_A. Recall from Section 2 the convention of representing undefined attributes using a constant ω. The argument of the temporal operator **F** (*eventually*) checks that attribute a is defined. If φ is guarded and has no global variables (i.e. its FO components are all sentences), then φ' is a guarded LTL-FO property. Therefore Theorem 3.3 applies, yielding decidability.

Verifying termination properties. Recall that our semantics of artifact systems and LTL-FO properties ignores blocking runs. However, in some applications, one would like to verify properties relating to termination. As discussed in Section 3, it is decidable if *all* pre-runs of an artifact system are blocking (Corollary 3.4). However, it may be desirable to verify more expressive properties involving blocking configurations. To this end, one can modify the semantics to render all runs infinite by repeating forever blocking configurations, whenever reached. It can be shown that our results continue to hold with this semantics. Note that one can state, within a guarded LTL-FO property, that a configuration of a guarded artifact system is blocking (all variables in negations of the $\exists^*$FO pre-conditions become globally quantified universally).

6. CONCLUSIONS

In this paper, we introduce the artifact system model, which formalizes a business process modeling paradigm that has recently attracted the attention of both the industrial and research communities. We study the problem of automatic verification of artifact systems, with the goal of increasing confidence in the correctness of such business processes.

All prior versions of the artifact model are inherently data-aware, being essentially evolved dataflow models. The version we consider extends prior models, taking significant additional steps towards data-awareness. It includes an underlying database which can be consulted by the services, and equips artifacts with updatable state relations. The service and property specifications allow sophisticated manipulation of data values via first-order formulae over the attributes and state of artifacts, the underlying database, and an infinite, ordered underlying domain. Data awareness raises a significant challenge compared to classical finite-state model checking, by turning artifact systems into infinite-state systems, whose verification problem is notoriously difficult.

We trace the boundaries of decidability for verification and we identify the guarded restriction, defining a practically appealing and fairly tight class of artifact systems and properties for which verification is decidable in PSPACE. This complexity is the best one can hope for, given that finite-state model checking is already PSPACE-complete. Our decidability result is significantly more difficult than the previous results of [46, 21] for ASM transducers and Web services, because each run is allowed to use infinitely many values from an underlying ordered domain. This extension is critical to the artifact framework, in order to adequately model arbitrary external input and partially specified processes given by pre- and post-conditions. Finally, we show that the verification techniques can also be leveraged to solve other static analysis tasks previously formulated for the artifact framework.

While the PSPACE complexity of verification is reasonable within the landscape of static analysis, one must legitimately wonder whether verification of such complexity is feasible in practice. Fortunately, previous experience is quite encouraging. Indeed, a PSPACE-complete verification algorithm for data-driven Web services, exhibiting excellent performance on a significant range of applications, has been implemented in the WAVE prototype [20, 22]. The implementation relies on a mix of symbolic model checking and database optimization techniques. We believe that a similar approach is likely to also be effective, after appropriate extensions enabled by our results, in the context of data-centric business process verification. This would be of interest to the database, computer-aided verification, and business process communities.

7. REFERENCES

[1] S. Abiteboul, L. Herr, and J. V. den Bussche. Temporal versus first-order logic to query temporal databases. In *Proc. ACM PODS*, pages 49–57, 1996.

[2] S. Abiteboul, L. Segoufin, and V. Vianu. Static analysis of Active XML systems. In *Proc. Intl. Symp. on Principles of Database Systems (PODS)*, pages 221–230, 2008.

[3] S. Abiteboul, V. Vianu, B. Fordham, and Y. Yesha. Relational transducers for electronic commerce. *JCSS*, 61(2):236–269, 2000. Extended abstract in PODS 98.

[4] K. Bhattacharya, N. S. Caswell, S. Kumaran, A. Nigam, and F. Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal*, 46(4):703–721, 2007.

[5] K. Bhattacharya et al. A model-driven approach to industrializing discovery processes in pharmaceutical research. *IBM Systems Journal*, 44(1):145–162, 2005.

[6] K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *Proc. Int. Conf. on Business Process Management (BPM)*, pages 288–304, 2007.

[7] K. Bhattacharya, R. Hull, and J. Su. A Data-centric Design Methodology for Business Processes. In J. Cardoso and W. van der Aalst, editors, *Handbook of Research on Business*

Process Management. 2009. to appear.

[8] M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Proc. IEEE Conf. on Logic in Computer Science (LICS)*, pages 7–16, 2006.

[9] E. Borger, E. Gradel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 1997.

[10] A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *FCT'07*, volume 4639 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2007.

[11] A. Bouajjani, P. Habermehl, and R. Mayr. Automatic verification of recursive procedures with one integer parameter. *Theoretical Computer Science*, 295:85–106, 2003.

[12] A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. In *TACAS'07*, volume 4424 of *Lecture Notes in Computer Science*, pages 690–705. Springer, 2007.

[13] P. Bouyer. A logical characterization of data languages. *Information Processing Letters*, 84(2):75–85, 2002.

[14] P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 182(2):137–162, 2003.

[15] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification of infinite structures. In *Handbook of Process Algebra*, pages 545–623. Elsevier Science, 2001.

[16] A. K. Chandra and M. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. Comp.*, 14(3):671–677, 1985.

[17] D. Cohn, F. Heath, F. Pinel, J. Vergo, and P. Dhoolia. Siena: From powerpoint to web application in 5 minutes (demo paper). In *Proc. Proc. of Intl. Conf. on Service Oriented Computing (ICSOC)*, Sydney, Australia, 2008.

[18] S. Demri and R. Lazić. LTL with the Freeze Quantifier and Register Automata. In *Proc. IEEE Conf. on Logic in Computer Science (LICS)*, pages 17–26, 2006.

[19] S. Demri, R. Lazić, and A. Sangnier. Model checking freeze LTL over one-counter automata. In *Proceedings of the 11th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'08)*, pages 490–504, 2008.

[20] A. Deutsch, M. Marcus, L. Sui, V. Vianu, and D. Zhou. A verifier for interactive, data-driven web applications. In *Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 539–550, 2005.

[21] A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web applications. *JCSS*, 73(3):442–474, 2007.

[22] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. A system for specification and verification of interactive, data-driven Web applications. In *Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 772–774, 2006.

[23] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *Proc. Intl. Symp. on Principles of Database Systems (PODS)*, pages 90–99, 2006.

[24] G. Dong, R. Hull, B. Kumar, J. Su, and G. Zhou. A framework for optimizing distributed workflow executions. In *Proc. Intl. Workshop on Database Programming Languages (DBPL)*, pages 152–167, 1999.

[25] E. A. Emerson. Temporal and modal logic. In J. V. Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 995–1072. North-Holland Pub. Co./MIT Press, 1990.

[26] C. Fritz, R. Hull, and J. Su. Automatic construction of simple artifact-based workflows. In *Proc. of Intl. Conf. on Database Theory (ICDT)*, 2009.

[27] C. E. Gerede, K. Bhattacharya, and J. Su. Static analysis of business artifact-centric operational models. In *IEEE International Conference on Service-Oriented Computing and Applications*, 2007.

[28] C. E. Gerede and J. Su. Specification and verification of artifact behaviors in business process models. In *Proceedings of 5th International Conference on Service-Oriented Computing (ICSOC)*, Vienna, Austria, September 2007.

[29] R. Glushko and T. McGrath. *Document Engineering: Analyzing and Designing Documents for Business Infomratics and Web Services*. MIT Press, Cmabridge, MA, 2005.

[30] R. Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008*, Monterrey, Mexico, 2008.

[31] R. Hull, F. Llirbat, B. Kumar, G. Zhou, G. Dong, and J. Su. Optimization techniques for data-intensive decision flows. In *Proc. IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 281–292, 2000.

[32] R. Hull, F. Llirbat, E. Simon, J. Su, G. Dong, B. Kumar, and G. Zhou. Declarative workflows that support easy modification and dynamic browsing. In *Proc. Int. Joint Conf. on Work Activities Coordination and Collaboration*, 1999.

[33] M. Jurdzinski and R. Lazić. Alternation-free modal mu-calculus for data trees. In *Proc. IEEE Conf. on Logic in Computer Science (LICS)*, pages 131–140, 2007.

[34] S. Kumaran, R. Liu, and F. Y. Wu. On the duality of information-centric and activity-centric models of business processes. In *Proc. Intl. Conf. on Advanced Information Systems Engineering (CAISE)*, 2008.

[35] S. Kumaran, P. Nandi, T. Heath, K. Bhaskaran, and R. Das. ADoc-oriented programming. In *Symp. on Applications and the Internet (SAINT)*, pages 334–343, 2003.

[36] J. Küster, K. Ryndina, and H. Gall. Generation of BPM for object life cycle compliance. In *Proceedings of 5th International Conference on Business Process Management (BPM)*, 2007.

[37] R. Lazić, T. Newcomb, J. Ouaknine, A. Roscoe, and J. Worrell. Nets with tokens which carry data. In *ICATPN'07*, volume 4546 of *Lecture Notes in Computer Science*, pages 301–320. Springer, 2007.

[38] R. Liu, K. Bhattacharya, and F. Y. Wu. Modeling business contexture and behavior using business artifacts. In *CAiSE*, volume 4495 of *LNCS*, 2007.

[39] D. Martin et al. OWL-S: Semantic markup for web services, W3C Member Submission, November 2003.

[40] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[41] P. Nandi and S. Kumaran. Adaptive business objects – a new component model for business integration. In *Proc. Intl. Conf. on Enterprise Information Systems*, pages 179–188,

2005.

[42] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of web services. In *Intl. World Wide Web Conf. (WWW2002)*, 2002.

[43] F. Neven, T. Schwentick, and V. Vianu. Finite State Machines for Strings Over Infinite Alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004.

[44] A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.

[45] M. Spielmann. Abstract State Machines: Verification problems and complexity. Ph.D. thesis, RWTH Aachen, 2000.

[46] M. Spielmann. Verification of relational transducers for electronic commerce. *JCSS.*, 66(1):40–65, 2003. Extended abstract in PODS 2000.

[47] J. Strosnider, P. Nandi, S. Kumaran, S. Ghosh, and A. Arsanjani. Model-driven synthesis of soa solutions. *IBM Systems Journal*, pages 415–432, 2008.

[48] J. Wang and A. Kumar. A framework for document-driven workflow systems. In *Business Process Management*, pages 285–301, 2005.

APPENDIX

Running Example

We present here our full running example. We model a scenario where a manufacturer fills customer purchase orders, negotiating the price of each line item on a case-by-case basis. We focus on two artifacts manipulated by the negotiation process, ORDER and QUOTE. During the workflow, the customer repeatedly adds new line items into (or updates existing ones in) the purchase order modeled by the ORDER artifact. Each line item specifies a product and its quantity. Line items are first tentatively filled in the ORDER attributes *li_prod* and *li_qty*. Every tentative line item spawns a negotiation process, in which manufacturer and customer complete rounds of declaring ask and bid prices, until agreement is reached or the negotiation fails. The prices at every round are stored in the QUOTE artifact, which also holds the manufacturer's initially desired price, the lowest bid he is willing to entertain, and the final negotiated price. Once the negotiation on a tentative line item succeeds, its outcome is scrutinized by a human executive working for the manufacturer. Upon the executive's approval, the line item is included into the purchase order (by insertion into the ORDER state line_items), and the final price is archived (in the QUOTE state li_quotes). During the negotiation, the manufacturer consults an underlying database, which lists information about available products (e.g. manufacturing cost) and about customers (e.g. credit rating and status).

The corresponding artifact system $\Gamma_{ex} = \langle \mathcal{A}, \Sigma \rangle$ is formally described below. As a font convention, we use *R* to refer to an artifact's attribute relation and S for state relations.

The artifact schema is $\mathcal{A} = \langle ORDER, QUOTE, \mathcal{DB} \rangle$, detailed as follows.

1. $\mathcal{DB} = \langle \text{PRODUCT}, \text{CUSTOMER} \rangle$ is the database schema, where:
   - PRODUCT$(prod_id, manufacturing_cost,$ $min\_order\_qty)$ is the relation containing products and production information, and
   - CUSTOMER$(customer_id, status, credit_rating)$ contains information about customers, such as customer status and credit rating.
2. $ORDER = \langle R_O, \underbrace{\mathsf{line\_items}, \mathsf{in\_process}, \mathsf{done}}_{S_O} \rangle$ is the artifact class containing the information about a customer's order:
   - $R_O(order\#, customer\_id, need\_by, li\_prod, li\_qty)$ is the attribute relation holding the order number, the identifier of the customer who placed the order, the day it is needed by. The role of attributes *li_prod* and *li_qty* is described below.
   - line_items$(prod_id, qty)$ is a state relation that acts as a "shopping cart" holding the collection of line items requested so far.
   - in_process and done are nullary state relations (boolean flags) keeping track of the stage the artifact is in. [5]

   The intention is that, in stage in_process, the customer repeatedly updates the shopping cart by specifying an individual, tentative line item described by attributes *li_prod* and *li_qty*. Subsequently, this line item is inserted into, deleted from, or replaces in line_items an item with the same *prod_id*, provided the price negotiation succeeds. When the customer completes the purchase order, the ORDER artifact transitions to stage done.
3. $$QUOTE = \langle R_Q, \left.\begin{array}{l} \mathsf{li\_quotes}, \mathsf{idle}, \mathsf{desired\_price\_calc}, \\ \mathsf{negotiation}, \mathsf{approval\_pending}, \mathsf{archive} \rangle \end{array}\right\} : S_Q$$
   is the artifact class modeling quotes, with:
   - $R_Q(\ order\#, desired\_price,$ $lowest\_acceptable\_price, ask, bid,$ $final\_price, approved, li\_prod, li\_qty,$ $manufacturing\_cost)$ is the attribute relation.
   - li_quotes$(prod_id, qty, price)$ is a state relation holding the final negotiated price quotes for the line items in the corresponding ORDER artifact.
   - idle, desired_price_calc, negotiation, approval_pending, archive are nullary state relations.

   When inactive, the QUOTE artifact is in state idle, but moves to desired_price_calc as soon as the customer fills in the product id and quantity of a line item. In this stage, *desired_price* attribute is set (from the manufacturer's point of view), possibly taking into account the *need_by* date attribute in the corresponding ORDER artifact and the manufacturing cost listed in the PRODUCT database. During the ensuing negotiation stage, the ask and bid prices are repeatedly set (in attribute *ask* by the manufacturer, respectively *bid* by the customer) until a final price is established and recorded in attribute *final_price*, or the negotiation fails. Final prices may

[5] Artifact class ORDER illustrates an extension of Definition 2.1 that allows several state relations. This extension is for convenience only: it is easy to show that it provides no additional expressive power and preserves our decidability results. Indeed, given artifact system Γ with multiple states per artifact and LTL-FO sentence φ, we can construct in polynomial time artifact system Γ' and sentence φ' such that $\Gamma \models \varphi$ iff $\Gamma' \models \varphi'$. Moreover, if Γ and φ are guarded, then so are Γ' and φ'.

require approval by a human executive to whom the negotiator reports. While approval is awaited, the QUOTE artifact is in stage approval_pending. Approval is granted by setting boolean attribute *approved*. Approved final prices are then archived in state relation li_quotes (while the QUOTE artifact is in stage archive).

The operations allowed on artifacts are modeled by the set Σ of available services, summarized below.

- Service $initiate\_order = \langle \pi^{io}, \psi^{io}, \mathcal{S}^{io} \rangle$ initializes the ORDER artifact, modeling the input of the order number and customer id by the manufacturer, and the need-by date by the customer.
- Service $add\_or\_modify\_line\_item = \langle \pi^{am}, \psi^{am}, \mathcal{S}^{am} \rangle$ models the customer's choice of a tentative line item to be added to the purchase order, or to replace another line item for the same product. The service records this choice in attributes *li_prod* and *li_qty* of the ORDER artifact, and initializes the QUOTE artifact in view of the upcoming negotiation. This involves copying the order number and line item information from ORDER to QUOTE, and filling the QUOTE's *manufacturing_cost* attribute with the corresponding value looked up in the PRODUCT database.
- Service $include\_line\_item = \langle \pi^{il}, \psi^{il}, \mathcal{S}^{il} \rangle$ includes into the purchase order the current tentative line item, by storing it in ORDER state line_items. The corresponding final negotiated price is archived in QUOTE state li_quotes.
- Service $commit\_order = \langle \pi^{co}, \psi^{co}, \mathcal{S}^{co} \rangle$ simply switches the ORDER artifact to the done stage, which disables any further line item modifications. This service models the customer's non-deterministic decision to finalize the purchase order.
- Service $set\_quote\_interval = \langle \pi^{sq}, \psi^{sq}, \mathcal{S}^{sq} \rangle$ sets the *desired_ price* and *lowest_acceptable_price* attributes of the QUOTE artifact to frame the subsequent negotiation. This service abstracts a complex sub-task, possibly taking into account the ORDER's *need_by* attribute, the manufacturer's desired profit margin, the customer's status, and input from a human manager.
- Service $quote\_approval = \langle \pi^{qa}, \psi^{qa}, \mathcal{S}^{qa} \rangle$ models the human supervisor who reviews the quote on behalf of the manufacturer. The process is a black box, about which is only known that it switches the QUOTE artifact to archive stage, and it sets the *approved* attribute to either *"yes"* or *"no"*.

To showcase the artifact model's natural ability to specify processes even partially, we describe the negotiation process at two levels of abstraction.

- In a first, coarser cut, the process is abstracted as service $abstract\_negotiation = \langle \pi^{an}, \psi^{an}, \mathcal{S}^{an} \rangle$ about which we only know that the final price is reached when the ask and bid prices coincide, and that it is guaranteed to lie between the allowed margins stored in attributes *desired_price* and *lowest_acceptable_price* of artifact QUOTE.
- Alternatively, we use service $refined\_negotiation =$ $\langle \pi^{rn}, \psi^{rn}, \mathcal{S}^{rn} \rangle$ to refine the negotiation process all the way to the level of individual negotiation rounds, each of which sets the current ask and bid prices.

Conventions. We adopt the following conventions:

(i) We model uninitialized attributes by setting them to the reserved constant ω.

(ii) We model Boolean states by nullary state relations, and drop the parentheses from atoms using them: S() becomes S. We assume the usual encoding of *true* as the singleton nullary relation, and *false* as the empty nullary relation. In particular, all Boolean states are initially *false* (since all state relations are initially empty).

(iii) For convenience, we use the following syntactic sugar for post-conditions: we write post-conditions as non-Horn rules $h(\bar{x}) := b(\bar{y})$ where the *head* h is a conjunction of atoms over attribute relations in $\mathcal{A}$, with variables $\bar{x}$, and the *body* b is a formula in $\mathcal{L}_{\mathcal{A}}$ with free variables $\bar{y}$, where $\bar{y} \subseteq \bar{x}$. The semantics is that whenever $A \xrightarrow{\sigma} A'$ holds, $A' \models h(\bar{x} \leftarrow \bar{u})$ for some tuple $\bar{u}$, and $A \models b(\bar{y} \leftarrow \bar{u}|\bar{y})$. Moreover, artifact relations not mentioned in h remain unchanged. Clearly, this syntactic sugar can be simulated by the official post-conditions, and conversely.

Service $initiate\_order = \langle \pi^{io}, \psi^{io}, \mathcal{S}^{io} \rangle$ initializes the ORDER artifact, where:

- $\pi^{io} \doteq \neg$in_process,
  i.e. the service applies when the ORDER artifact is not used to process another order;
- The post-condition ψ^{io} given by

  $$R_O(o, c, n, \omega, \omega) := o \neq \omega \wedge n \neq \omega \wedge \exists r, s \text{ CUSTOMER}(c, s, r)$$

  guarantees that the attributes *order#*, *customer_id* and *need_by* are initialized (set distinct from ω). By the semantics of post-conditions, the pick of o, c, n is non-deterministic. The pick of n models the customer's input, while that of o models the assignment of an order number by the manufacturer. Note that no further constraints are imposed on these values, they are simply picked from the infinite domain. In contrast, the customer c must be one of the existing customers listed in the database relation CUSTOMER. The pick of c also models the manufacturer's input.

  The tentative line item's price p and quantity q are left uninitialized (they equal ω) and will be set by the customer during an activity modeled by service $add\_or\_modify\_line\_item$ below.
- The state rules in $\mathcal{S}^{io}$ include the following:
  - in_process $\leftarrow true$,
    an insertion rule that sets the in_process boolean flag.
  - $\neg$done $\leftarrow true$,
    a deletion rule that resets boolean state flag done, ensuring it is mutually exclusive with in_process. (it is the responsibility of all other services operating on ORDER to keep them so
    – see $add\_or\_modify\_line\_item$ below).

  No rule refers to state relation line_items, as no line item exists yet.

Service $add\_or\_modify\_line\_item$ models the customer's choice of a tentative line item to be added to the purchase order, or to replace another line item for the same product. The service affects both the ORDER and the QUOTE artifact, and it is given as $\langle \pi^{am}, \psi^{am}, \mathcal{S}^{am} \rangle$, where:

- $\pi^{am} \doteq \exists o, c, n\ \textsf{in\_process} \wedge R_O(o, c, n, \omega, \omega) \wedge \textsf{idle} \wedge R_Q(\omega, \omega, \omega, \omega, \omega, \omega, \omega, \omega, \omega, \omega)$,
  i.e. the service applies only if the ORDER artifact is in the in_process stage, no other line item is currently being processed, and if the QUOTE artifact is currently unused (in the idle stage, with all attributes uninitialized).
- The post-condition ψ^{am} is

  $$\begin{aligned} R_O(o, c, n, p, q) \wedge R_Q(o, \omega, \omega, \omega, \omega, \omega, \omega, p, q, m) := \\ \exists q'\ R_O(o, c, n, \omega, \omega) \wedge p \neq \omega \wedge q \neq \omega \wedge \\ q \geq q' \wedge \text{PRODUCT}(p, m, q') \end{aligned}$$

  Note that the customer's input of product id p and quantity q is modeled as a non-deterministic pick from the infinite domain. The picked p must appear in the PRODUCT catalog stored in the database. The quantity q is less restricted: we only know that it is defined ($q \neq \omega$) and, reflecting the manufacturer's policy, it exceeds the minimum-order quantity q' listed in the PRODUCT catalog.

  According to the post-condition, the service reacts as follows to the customer's input of the tentative line item. It stores the values p and q into the attributes *li_prod, li_qty* of the ORDER artifact. Note that attributes *order#, customer_id* and *need_by* remain unchanged. The service also initializes the *order#* attribute of the QUOTE artifact to refer to the corresponding order, and also stores in it p, q and the manufacturing cost m for product p, which is looked up in the database catalog PRODUCT. The QUOTE artifact's remaining attributes are left undefined, to be set during negotiation.
- $\mathcal{S}^{am}$ contains no ORDER state rule as the order's state is left unchanged. It contains the following state rules that move the QUOTE artifact to the desired_price_calc stage, which enables the sub-task of quote negotiation: insertion rule

  $$\textsf{desired\_price\_calc} \leftarrow \mathit{true}$$

  and deletion rule

  $$\neg\textsf{idle} \leftarrow \mathit{true}.$$

Service $\mathit{include\_line\_item} = \langle \pi^{il}, \psi^{il}, \mathcal{S}^{il} \rangle$ includes into the purchase order the current tentative line item, by storing it in ORDER state line_items. The corresponding final negotiated price is archived in QUOTE state li_quotes. We have:

- The pre-condition

  $$\begin{aligned} \pi^{il} \doteq\ & \textsf{in\_process} \wedge \exists o, c, n, p, q\ R_O(o, c, n, p, q)\ \wedge \\ & o \neq \omega \wedge c \neq \omega \wedge n \neq \omega \wedge p \neq \omega \wedge q \neq \omega\ \wedge \\ & \exists d, l, f, m\ R_Q(o, d, l, f, f, f, \text{"}\mathit{yes}\text{"}, p, q, m)\ \wedge \\ & \textsf{archive} \end{aligned}$$

  ensures that the service applies only if a current line item p, q exists, the ORDER artifact is in stage in_process, and the QUOTE artifact lists a successful and approved negotiation for this line item and order (notice the common occurrence of o, p, q in both the QUOTE and the ORDER atoms). Successful negotiation occurs when the ask, bid and final price coincide, and the artifact is in state archive. The final price is approved when the *approved* attribute is set to "yes".
- The post-condition ψ^{il} given by

  $$\begin{aligned} R_O(o, c, n, \omega, \omega) \wedge R_Q(\omega, \omega, \omega, \omega, \omega, \omega, \omega, \omega, \omega, \omega) := \\ \exists p', q'\ R_O(o, c, n, p', q') \end{aligned}$$

  guarantees that, regardless of the current value p', q' of the line item, in the successor ORDER artifact the values are reset to undefined (ω), to make room for the next tentative line item. Notice that the ORDER attributes *order#, customer_id* and *need_by* are preserved. The QUOTE attributes are all reset, in preparation for the next negotiation.
- The state rules in $\mathcal{S}^{il}$ include the following:
  - The state insertion rule

    $$\textsf{line\_items}(p, q) \leftarrow \exists o, c, n\ R_O(o, c, n, p, q)$$

    operates on artifact ORDER, inserting the values of attributes *li_prod* and *li_qty* into state relation line_items. The state deletion rule

    $$\begin{aligned} \neg\textsf{line\_items}(p, q) \leftarrow\ & \exists o, c, n, q'\ R_O(o, c, n, p, q')\ \wedge \\ & \textsf{line\_items}(p, q) \end{aligned}$$

    deletes any other entry pertaining to the same product p (if any). Recall that, according to the possible successor definition, if state line_items already contains an entry for product p, the combined effect of the insertion and deletion rule is that of *updating* the quantity of product p to the latest customer-provided (and successfully negotiated) value. If no prior entry for p exists, then the deletion rule has no effect.
  - The final negotiated price for this line item is archived in QUOTE state li_quotes by the following insertion rule: $\textsf{li\_quotes}(p, q, f) \leftarrow$
    $\exists o, d, l, m\ R_Q(o, d, l, f, f, f, \text{"}\mathit{yes}\text{"}, p, q, m).$
    The following insertion and deletion rules move the QUOTE artifact to state idle, signaling its availability for a new negotiation sub-task:

    $\textsf{idle} \leftarrow \mathit{true}$ and
    $\neg\textsf{archive} \leftarrow \mathit{true}.$

Service $\mathit{commit\_order} = \langle \pi^{co}, \psi^{co}, \mathcal{S}^{co} \rangle$ simply switches the ORDER artifact to the done stage, which disables any further line item modifications. This service models the customer's non-deterministic decision to finalize the purchase order.

- $\pi^{co} \doteq \textsf{in\_process}\ \wedge$
  $\exists o, c, n, p, q\ R_O(o, c, n, p, q) \wedge p = \omega \wedge q = \omega$,
  i.e. the full order can be committed only if no tentative line item is still being processed (which would make $p \neq \omega, q \neq \omega$).
- ψ^{co} is given by

  $$R_O(o, c, n, p, q) := R_O(o, c, n, p, q)$$

  i.e. the artifact's attributes do not change.
- $\mathcal{S}^{co}$ contains only the rules
  $\textsf{in\_process} \leftarrow \mathit{false}$ and
  $\textsf{done} \leftarrow \mathit{true}.$

The following services model the negotiation process.

Service $\mathit{set\_quote\_interval}$ sets the *desired_ price* and *lowest_acceptable_price* attributes of the QUOTE artifact to frame the subsequent negotiation. This service abstracts a complex subtask, possibly taking into account the ORDER's *need_by* attribute, the manufacturer's desired profit margin, the customer's status, and input from a human manager.

$set\_quote\_interval = \langle \pi^{sq}, \psi^{sq}, \mathcal{S}^{sq} \rangle$, where:

- $\pi^{sq} \doteq$ desired_price_calc.
- Post-condition ψ^{sq} given by

  $$\begin{aligned} R_Q(o,d,l,d,\omega,\omega,\omega,p,q,m) :=& \\ & d \neq \omega \wedge l \neq \omega \wedge d \geq l \geq m \wedge \\ & R_Q(o,\omega,\omega,\omega,\omega,\omega,\omega,p,q,m). \end{aligned}$$

  models only what is known about the quote generation procedure viewed as a black box: namely that the desired price is higher than the lowest acceptable one, which in turn exceeds the manufacturing cost. It also sets the initial asking price to the desired price in preparation for the negotiation stage.
- The rules in $\mathcal{S}^{sq}$ simply switch the artifact to the negotiation stage, and are omitted.

To showcase the artifact model's natural ability to specify processes even partially, we describe the negotiation process at two levels of abstraction.

In a first, coarser cut, the process is abstracted as service $abstract\_negotiation = \langle \pi^{an}, \psi^{an}, \mathcal{S}^{an} \rangle$ about which we only know that the final price is reached when the ask and bid prices coincide, and that it is guaranteed to lie between the allowed margins stored in attributes *desired_price* and *lowest_acceptable_price* of artifact QUOTE.

- $\pi^{an} =$ negotiation,
  since the process can only start when the QUOTE artifact is ready, which is signaled by setting this state flag.
- Post-condition ψ^{an}, given as

  $$\begin{aligned} R_Q(o,d,l,f,f,f,app,p,q,m) :=& \\ & \exists a', b', f' \; R_Q(o,d,l,a',b',f',app,p,q,m) \wedge \\ & l \leq f \leq d \end{aligned}$$

  guarantees that the final price f agrees with the final ask and bid prices regardless of their initial values a', b', and that f lies between the desired price d and the lowest acceptable price l.
- We omit the rules in $\mathcal{S}^{an}$, which move the artifact to state approval_pending.

Alternatively, we can refine the negotiation process all the way to the level of individual negotiation rounds, each of which sets the current ask and bid prices. A post-condition ensures that the negotiation is well-formed, i.e. the bid never exceeds the ask price, and that across rounds, asking prices never increase, while bids never decrease. The negotiation is successful when ask and bid prices agree, at which time the QUOTE artifact moves to the approval_pending state, the *final_price* attribute is set, and and no further rounds are conducted.

Service $refined\_negotiation = \langle \pi^{rn}, \psi^{rn}, \mathcal{S}^{rn} \rangle$ is described as follows:

- $\pi^{rn} =$ negotiation
  ensures that the service applies only as long as the boolean state flag negotiation is set in the QUOTE artifact.
- Post-condition ψ^{rn} is given as

  $$\begin{aligned} R_Q\ &(o,d,l,a,b,f,app,p,q,m) := \\ &(\exists a', b' \; R_Q(o,d,l,a',b',\omega,app,p,q,m) \wedge \\ &\; a' \neq b' \wedge l \leq a \leq a' \wedge b' \leq b \wedge f = \omega) \\ &\; \vee \\ &\; (R_Q(o,d,l,a,b,\omega,app,p,q,m) \wedge a = b = f). \end{aligned}$$

  According to the first disjunct, the negotiation is well-formed, i.e. the bid never exceeds the ask price, and in each round, asking prices a never increase while bids b never decrease. Moreover, as long as ask and bid price differ, the final price remains undefined (equal to ω). Notice that the values of a and b are otherwise unconstrained, being simply drawn from the infinite domain. This models their external input by the manufacturer, respectively customer. The second disjunct states that once ask price a and bid price b coincide, the final price f is automatically set to the common value.
- $\mathcal{S}^{rn}$ contains rules that, upon detecting successful negotiation, switch the QUOTE artifact to stage approval_pending if the customer does not enjoy preferred status with excellent credit. If he does, then the approval is short-circuited and the QUOTE goes to stage archive. The negotiation is successful when ask and bid prices agree.

  $$\begin{aligned} &\text{approval\_pending} \leftarrow \\ &(\exists o,d,l,a,f,app,p,q,m \; R_Q(o,d,l,a,a,f,app,p,q,m) \wedge \\ &\; \exists c,n \; R_O(o,c,n,p,q) \wedge \\ &\; \neg\text{CUSTOMER}(c, \text{"}\mathit{preferred}\text{"}, \text{"}\mathit{excellent}\text{"})) \end{aligned}$$

  $$\begin{aligned} &\text{archive} \leftarrow \\ &(\exists o,d,l,a,f,app,p,q,m \; R_Q(o,d,l,a,a,f,app,p,q,m) \wedge \\ &\; \exists c,n \; R_O(o,c,n,p,q) \wedge \\ &\; \text{CUSTOMER}(c, \text{"}\mathit{preferred}\text{"}, \text{"}\mathit{excellent}\text{"})) \end{aligned}$$

  $$\begin{aligned} &\neg\text{negotiation} \leftarrow (\exists o,d,l,a,f,app,p,q,m \\ &\quad R_Q(o,d,l,a,a,f,app,p,q,m)) \end{aligned}$$

  Note that state flag negotiation must be set before the state update rules execute (since pre-condition π^{rn} is satisfied). If neither of the state rule bodies is satisfied, then according to the possible successor semantics, the negotiation flag remains set, enabling another negotiation round.

Finally, service $quote\_approval = \langle \pi^{qa}, \psi^{qa}, \mathcal{S}^{qa} \rangle$ models the human supervisor who reviews the quote on behalf of the manufacturer. The process is a black box, about which is only known that it switches the QUOTE artifact to archive stage, and it sets the *approved* attribute to either *"yes"* or *"no"*.

- $\pi^{qa} \doteq$ approval_pending.
- ψ^{qa} is given by

  $$\begin{aligned} R_Q(o,d,l,f,f,f,app,p,q,m) :=& \\ & R_Q(o,d,l,f,f,f,\omega,p,q,m) \wedge \\ & (app = \text{"}\mathit{yes}\text{"} \vee app = \text{"}\mathit{no}\text{"}). \end{aligned}$$
- $\mathcal{S}^{qa}$ comprises the state rules that switch the artifact to stage archive, and are omitted.

We illustrate desirable properties for Γ_{ex}, which pertain to its global evolution, as well as to the consistency of the specification.

One such consistency property requires the state flags $\mathsf{in\_process}$ and done in class ORDER to always be mutually exclusive:

$$\mathbf{G}(\neg(\mathsf{in\_process} \wedge \mathsf{done})).$$

A more data-centric consistency property requires line item quotes archived in state $\mathsf{li\_quotes}$ of the QUOTE artifact to pertain only to tentative line items previously input by the customer (into attributes *li_prod* and *li_qty* of the ORDER artifact), and which underwent successful negotiation and approval. Successful negotiation occurs when ask, bid and final price coincide and the QUOTE artifact is in state $\mathsf{archive}$:

$$\begin{array}{l}
\forall pid, qty, prc \\
\quad \mathbf{G}\ ((\exists o, c, n\ R_O(o, c, n, pid, qty) \wedge \\
\qquad\quad \exists d, l, m\ R_Q(o, d, l, prc, prc, prc, "yes", pid, qty, m) \wedge \\
\qquad\quad \mathsf{archive}) \\
\qquad\qquad\qquad \mathbf{B} \\
\qquad\quad \neg\mathsf{li\_quotes}(pid, qty, prc))
\end{array}$$

Notice the use of the *before* operator **B** (requiring its first argument to hold before its second argument fails).

The following property is more semantic in nature, capturing part of the manufacturer's business model. It requires that if the customer's status is not *"preferred"* and the credit rating is worse than *"good"*, then before archiving a line item with final negotiated price lower than the manufacturer's desired price, explicit approval from a human executive must have been requested. We assume the following ordering on the constants indicating the credit rating:

$$\textit{"poor"} < \textit{"fair"} < \textit{"good"} < \textit{"excellent"}.$$

$$\begin{array}{l}
\varphi_3 : \\
\quad \forall o, c, n, p, q, d, l, f, m, s, r \\
\qquad \mathbf{G}\ ((R_O(o, c, n, p, q)\ \wedge \mathsf{in\_process}\ \wedge \mathsf{negotiation} \wedge \\
\qquad\qquad R_Q(o, d, l, f, f, f, \omega, p, q, m)\ \wedge\ f < d \wedge \\
\qquad\qquad CUSTOMER(c, s, r) \wedge s \neq \textit{"preferred"} \wedge \\
\qquad\qquad r < \textit{"good"}) \rightarrow \\
\qquad\qquad (\mathsf{approval\_pending} \\
\qquad\qquad\qquad\quad \mathbf{B} \\
\qquad\qquad \neg(\mathsf{archive} \wedge \mathsf{li\_quotes}(p, q, f))))
\end{array}$$

Note that φ_3 involves both artifacts and the underlying database. If the negotiation process is described by service *refined_negotiation*, then the property happens to be satisfied: indeed, recall from its state rules that this service requests approval whenever the customer's status is not preferred and his credit rating is not excellent. In particular, this applies to customers whose rating is worse than good, according to the above ordering of credit ratings.