Static Analysis of Business Artifact-centric Operational Models

Cagdas E. Gerede*
University of California at Santa Barbara
gerede@cs.ucsb.edu

Kamal Bhattacharya
IBM T.J. Watson Research Center
kamalb@us.ibm.com

Jianwen Su*
University of California at Santa Barbara
su@cs.ucsb.edu

Abstract

Business Artifacts are the core entities used by businesses to record information pertinent to their operations. Business operational models are representations of the processing of business artifacts. Traditional process modeling approaches focus on the actions taken to achieve a certain goal (verb-centric). Business artifact-centric modeling starts by identifying what is acted upon (noun-centric), and constructs business operational models by identifying the tasks/actions that business actors execute to add business value. In this paper, we identify important classes of properties on artifact-centric operational models. In particular, we focus on persistence, uniqueness and arrival properties. To enable a static analysis of these properties, we propose a formal model for artifact-centric operational models. We show that the formal model guarantees persistence and uniqueness. We prove that, while checking an arrival property is undecidable in general, under a restricted version of the formalism, an arrival property can be checked in EXPTIME .

1 Introduction

Any business needs to maintain records of information pertinent to their operations. Information about the key aspects of a business operation is used to evaluate the effectiveness in producing the business products. In recent years regulatory measures require significant rethinking about the level of traceability to ensure accountability of business actions that meet compliance standards. Businesses of today, small or large, maintain such records, sometimes surprisingly still in paper format, sometimes in more electronic formats such as spreadsheets and sometimes within larger management systems such as ERP solutions.

In our day-to-day experience one can observe many uniquely identifiable entities that are used by businesses to capture information pertinent to a business operation. For example, consider a standard banking process where a customer withdraws money from his/her banking account. The customer fills out a withdrawal slip which captures the customers account number, name, date and the amount s/he wants to withdraw. A teller receives this request document and matches it against the customers account status and properly fulfills the request by withdrawing the requested amount from the customers account. The withdrawal slip is then filed, eventually scanned and added to the customers records. Another common example is the management of orders in a restaurant. The key entity here is the guest check, which captures all orders, the price for each order, the total, when it has been paid and what the means of payment were. The restaurant thereby records all important information on guest check to account for all orders placed and money received from the customers, which is matched against the actual cash receipts in the register. The Withdrawal Slip and the Guest Check are the core entities used to manage business operations for the bank or the restaurant respectively. We call these core entities Business Artifacts[22].

In recent years several teams within IBM Research have conducted over a dozen studies with a variety of enterprises from different industry vertical. The core aspect of these studies was to identify business artifacts and how they are used to manage business operations. A business artifact is an identifiable, self-describing unit-of information through which business stakeholders add value to the business. A Withdrawal Slip, a Receipt, a Purchase Order, a Service Order, etc. are typical examples of business artifacts. A purchase order is a unique entity with well-defined business intent, namely capturing the order of a customer and recording the status of the order as it is processed. The business stakeholders involved in processing the purchase order will update the document in the course of the purchasing operation. The purchase order thereby has a life-cycle from creation by the customer, received by the purchasing department, validated by the credit department and upon delivery of goods, closed and eventually archived.

Traditional business process modeling techniques focus

---

*Supported in part by NSF grants IIS-0415195 and CNS-0613998.

on the actions taken to achieve a certain goal (often referred to as verb-centric). Hence, business stakeholders describe their business by stating first we do A, then B, then C, and while doing C we also do D. Our approach proposes to focus on what is acted upon, thus describing business operations by first identifying the things that matter to their business (e.g. Purchase Order, Insurance Claim), and second how these things are processed to achieve a certain goal. Modeling business operations using business artifacts is thus a noun-centric approach.

In our client engagements we have made several observations with respect to the value of artifact-centric operational modeling. First, consolidating business processes in enterprises of today is an arduous task. Large enterprises are often grown out of mergers and acquisitions and thus many business processes exist in various different forms within the same enterprise. Consolidation efforts are often stalling the unification process as business process owners cannot agree on the one-for-all process. The reluctance of business stakeholders to agree on one process is somewhat a cultural problem, yet we have found that introducing the noun-centric artifact approach helps business stakeholders more easily to agree on the business operations. We empirically observed that it is easier for business stakeholders to agree on the things worked upon to managing their business rather than on how they want to establish a business process. Agreeing on the artifacts that capture key information pertinent to the business context is the basis for asking how these artifacts are being processed to achieve the business goal. Second, over the last few years a research team at IBM Research has developed a toolkit containing tools, technologies and methods to use the notion of business artifacts and build SOA solutions by maintaining the abstractions in terms of artifacts from operational modeling down to solution design to implementation. This has been done in the context of Model-driven Business Transformation[3] and applied in various real-world engagements.

In this study we investigate a formalization of artifact-centric modeling of business operations (also referred to as *business operational modeling* as opposed to *business process modeling*) and study some static analysis questions.

The main technical contributions of this paper are:

- We investigate business artifact-centric operational modeling approach of [22, 3], and for each construct we propose an appropriate formalism that enables automated analysis of such models.
- We identify some important classes of properties such as persistence, uniqueness, arrival, encounter, semi-synchronizability, redundancy, access independence, data independence. We focus on persistence, uniqueness, and arrival.
- We show that the proposed formalism guarantees persistence and uniqueness. We prove that checking an arrival property is undecidable in general. Under a restricted version of the formalism, we demonstrate that an arrival property can be checked in EXPTIME.

The rest of the paper is organized as follows: In Section 2, we give an informal introduction to artifact-centric operations modeling with an example. In Section 3, we describe a set of important properties one may want to check against operational models. In Section 4, we present our formalism. In Section5, we explain our results. The conclusion and future work are provided at the end.

1.1 Related Work

Many tools and techniques have been used for the development of business process models over the last two decades[14, 19, 27, 17]. These approaches have followed a process-centric approach focused on the control and coordination of tasks[9]. The importance of a data-centric view of processes is advocated in [2] and [12]. In [2], the authors encourage an “object view” of scientific workflows where the data generated and used is the main focus. In [12], the author investigates “attribute-centric” workflows where attributes and modules have states. [18] proposes a mixed approach which can express both control and data flow. Another mixed approach is proposed by BPMN[4]. Compared to these approaches, our work also follows a data-centric view, but the granularity of our notion of data is different.

Our work is also related to object-oriented programming formalisms that have been proposed in [1] and [13]. [1] proposes a model for object-oriented programming and studies consistency of method calls with respect to method signatures. Our method formalism is syntactically similar to the one of [13] where an arrival-like question was studied. In our case, however, the methods are nonrecursive and the arrival is formulated between configurations instead of methods.

Other related work includes object histories of [11], Inflow schemas of [15], the scripts of [20], and object migrations of [26]. [11] defines a mathematical formalism that captures the main aspects of the historical data for objects. Our notion of a run is similar to an “object history” of [11], but in our case a computation-tuple sequence also includes states of tasks. The computation model for tasks can be viewed as a proper subset of behavior models such as the scripts of TAXIS [20]. Infoflow [15] proposes a data centric modeling but a computation model is not provided for tasks. [26] investigates the formal characterization of object migration patterns under different update languages.

Our intention to use finite state machines for business artifacts resembles the typestate concept proposed in [25]. Typestate is a refinement of a type and a machine over typestates is used to define syntactically well-formed but semantically undefined sequences of operations on variables of a type. In this work, we do not provide an extensive language to describe constraints on the artifacts and their relationships; however, the results on the constraint languages for object models such as [23] and [6] can give some hints along this direction.

There have been studies also on static analysis of e-business process specifications. In [21] and [24], the authors analyze BPEL specifications via converting them to Petri Nets. In [8], the authors present a set of tools and techniques for analyzing interactions of composite asynchronous BPEL services. [16] provides a verification tool for web service orchestration, process algebra as the underlying formalism. In [5], the authors use a mix of techniques from logic and automated verification, and propose a tool to verify data-driven Web services. In [7], specifications are modeled in UML, in the form of Message Sequence Charts (MSCs), and transformed into the Finite State Process notation (FSP) to concisely describe and reason about the concurrent executions.

2 Business Artifact-centric Operational Modeling

In this section, we briefly describe the terminology and the constructs in artifact-centric operational modeling. Artifact-centric models consist of 3 key constructs: Business artifacts, business tasks, and repositories [22].

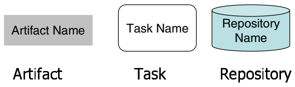

Figure 1. Graphical Notations of Modeling Primitives

A *business artifact* is an identifiable, self-describing unit-of-information. Business stakeholders add value to the business through business artifacts. For instance, in a restaurant, guest checks are the key business artifacts. A guest check captures what a customer orders and the status of these orders as they are processed. Business stakeholders such as waiters and cashiers update the check in the course of a dining experience. A guest check thereby has a lifecycle from the creation when the customer arrives to the archival when the customer leaves.

A *business task* (or simply task) describes the work acting upon an artifact by which a business role adds measurable business value to this artifact. We require a task to generate business value and hence, update an artifact. This condition helps in defining the granularity of a task or the task boundaries. Imagine a simple scenario where two tasks, T1 and T2 work on an artifact consisting of ten name-value pairs. A business stakeholder could determine that completion of T1 will require an update of, say, attributes 2-5 of the artifact and T2 requires an update of attributes 6-10. Therefore, adding business value in this case can be clearly defined by the business stakeholder who thereby determines the boundaries of a task.

A *repository* describes a waiting shelf or a buffer for an artifact. Tasks can push an artifact into a repository and pull an artifact out of a repository.

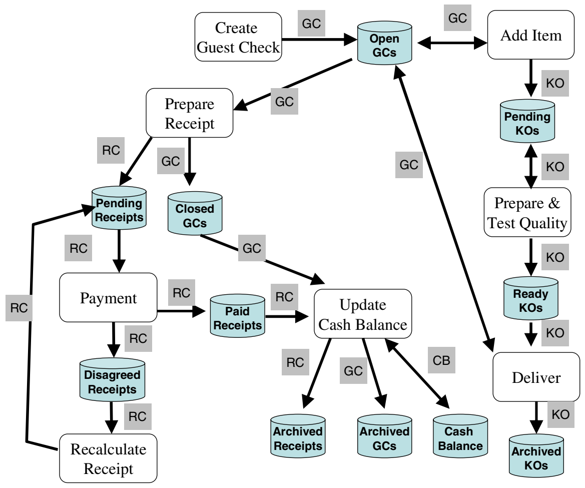

Figure 2. Operational Model of Resturant(GC= GuestCheck, RC= Receipt, KO=Kitchen Order, CB=Cash Balance)

Next we illustrate these constructs with an example.

Example:[1] *Figure 2 illustrates an operational model of a restaurant. This restaurant uses four artifact types while conducting its business: "guest checks", "receipts", "kitchen orders", and "cash balance".*

We can informally describe the operations of this restaurant as follows: When a customer walks in the restaurant, the customer is greeted and a guest check artifact is created with the customer information such as the customer name and table number. It is then placed into the repository for "open" guest checks (Open GCs). When the customer gives an order, "AddItem" task is triggered. This task retrieves the customer's guest check, creates a kitchen order and adds it into the "pending kitchen orders" repository. After the kitchen order is prepared, if it passes the quality test, it is inserted into the "ready kitchen orders" repository. Otherwise, it is sent back to the pending kitchen orders repository to be re-prepared. The required quality by the quality test depends on the customer. If the customer is a "special" customer then the order must have a high quality, otherwise, a regular quality is "ok". The kitchen orders passing the quality test is delivered, and the delivery time of the kitchen order is recorded on the guest check and the kitchen order artifact is "archived".

When the check is requested, a receipt is prepared and the guest check is "closed". If the customer disagrees on the receipt amount, the receipt is recalculated. When the payment of the receipt is successfully done, the receipt is added to the cash balance of the restaurant and then, the receipt and the guest check are archived.

3 Static Analysis of Operational Models

Artifact-centric operational models are specifications of businesses at an abstract level and they are used as the start-

---

[1] This example is adapted from the one in [22]

ing point to obtain implementations. Therefore, the "correctness" of models is of critical importance to businesses. By correctness, we mean models should satisfy "desired" properties derived from the business logic and context. Towards this direction, we identified several classes of properties. These properties help to understand the discrepancies between what is desired and what a model satisfies.

Arrival: Can an artifact A arrive into a repository R? In our example, a guest check should be able to arrive into the *archived-guest-checks* repository. This shows the completion of the guest check processing is possible.

Persistence: Once an artifact is created, does it persist or can it disappear? In order for a business to operate, the "important" business data should never disappear and should always be locatable. For example, a guest check should never be "destroyed" or get "lost", because it contains key information pertinent to the operations of the restaurant.

Uniqueness: Can an artifact appear in more than one "places" at once? An "unambiguous" specification of business operations require "important" business data to be in one business state at any time. For instance, a receipt shouldn't be in both *paid-receipts* and *disagreed-receipts* repositories at the same time.

Encounter: Can two different artifacts encounter? For instance, in our example, a receipt and a kitchen order never encounter, while a receipt and a guest check may encounter in *update-cash-balance* task.

Semi-synchronizability: An artifact's "progress" can be dependent on the existence of another artifact. For instance, the archival of a kitchen order requires the existence of the guest check; therefore, the kitchen order is semi-synchronizable with the guest check. On the other hand, the guest check can be moved to *closed-guest-checks* repository without the arrival of the kitchen order. Therefore, the guest check is not semi-synchronizable with the kitchen order.

Task/Repository Redundancy: Is a task (or repository) redundant? (The removal of it doesn't "impact" the "behavior" of the model). An operational model can be optimized by eliminating the redundancies.

Access Independence of Tasks: Do two tasks access different parts of the same artifact? For instance, the tasks *prepare-receipt* and *recalculate-receipt* are not access independent, because they access the receipt amount.

Data Independence: Do two artifacts have a data relationship? For instance, the receipt and the cash balance are not data independent because the cash balance amount depends on the receipt amount. On the other hand, the guest check and the cash balance have no data relationship; therefore, they are data independent.

In this paper, we focus on *arrival, persistence*, and *uniqueness* properties. Automated analysis of these properties require a formalization for operational models. To achieve this, in the next section, we propose a formal model.

4 A Formal Model For Business Operations

In this section, we propose a formal model for artifact-centric business operations and the key concepts including "artifacts" and "operational models".

We assume two pairwise disjoint domains: an infinite set of uninterpreted elements denoted by $\mathrm{dom}_=$ on which only the equality relation is defined, and an infinite densely ordered set $\mathrm{dom}_\leqslant$. The values in $\mathrm{dom}_= \cup \mathrm{dom}_\leqslant$ are called *constants*. We assume an infinite set of *business artifact instance identifiers* denoted by $\mathrm{dom}_{\mathrm{id}}$ on which only the equality relation is defined. We also assume an infinite set $\mathcal{N}_A$ of business artifact type names.

We define three kinds of *atomic types*: `Dle`, `Deq`, and a *business artifact type* to which a unique name in $\mathcal{N}_A$ is associated (more explanation comes later). We define a mapping *dom* between these types and the domains such that *dom*(`Deq`) $= \mathrm{dom}_=$, *dom*(`Dle`) $= \mathrm{dom}_\leqslant$, *dom*$(\tau) = \mathrm{dom}_{\mathrm{id}}$ where $\tau \in \mathcal{N}_A$.

We also define a data function type. A *data function type* has the form $S_1, \dots, S_l \rightarrow T_1, \dots, T_m$ where S_i, T_i's are atomic types. An instance f of such a type is a mapping $f : S_1 \times \cdots \times S_l \rightarrow T_1 \times \cdots \times T_m$ with a finite domain such that f maps a tuple $(a_1, \dots, a_l) \in$ *dom(S_1)*$\times \cdots \times$*dom(S_l)* to a tuple $(b_1, \dots, b_m) \in$ *dom(T_1)*$\times \cdots \times$*dom(T_m)*.

An *artifact comparison over* x, y is of the form $x = y$ where x, y are variables of the same artifact type. *Scalar comparisons* over x, y, c are of the forms $x\theta y$ and $x\theta c$ where x, y are variables of the type `Deq` (resp. `Dle`), c is a constant in $\mathrm{dom}_=$ (resp. $\mathrm{dom}_\leqslant$), and $\theta \in \{=, \neq\}$ (resp. $\theta \in \{<, \leqslant, >, \geqslant, =, \neq\}$).

4.1 Business Artifact

We now present the central notion of an artifact. A business artifact contains information pertinent to the business context at hand and business operations represent the lifecycle of artifacts from creation to completion. Information is represented using attributes. Furthermore a set of methods and a state machine describe the act of processing on the artifact from the artifact's perspective. Our intention to use state machines is to define semantically defined sequences of operations on artifacts similar to the typestate concept of [25]. Although the notion except for the state machine resembles a class definition in object oriented models, our key focus is on the life cycles of artifacts or "process organization" rather than the ease of programming or data organization.

Definition: A *business artifact type* (shortly an artifact type) is a tuple (L, P, M) such that

- L is a set of attribute names with associated types.
- P is a set of methods with distinct names.
- M is a (possibly nondeterministic) finite state machine and its transitions are labeled with method names from

P such that M recognizes the language of the allowed sequences of method invocations.

- A *method* has four components: 1) *name*; 2) *input parameters*: $u_1, \dots, u_m$ with atomic types $S_1, \dots, S_m$; 3) *output parameters*: $w_1, \dots w_n$ with atomic types $T_1, \dots, T_n$; 4) *body*: A sequence of statements of the following forms:
  - $x := y$ where x is an output parameter or in L, y is an input parameter or in L, and *typeOf(x)= typeOf(y)*,
  - $x \cup = (u_{i_1}, \dots, u_{i_j} \mapsto u_{k_1}, \dots, u_{k_l})$ where $x \in L$ and *typeOf(x)* $= S_{i_1}, \dots, S_{i_j} \rightarrow S_{k_1}, \dots, S_{k_l}$, and $i_x, k_y \in \{1, \dots, m\}$,
  - $(w_{k_1}, \dots, w_{k_l}) := x(u_{i_1}, \dots, u_{i_j})$ where $x \in L$ and *typeOf(x)=* $S_{i_1}, \dots, S_{i_j} \rightarrow T_{k_1}, \dots, T_{k_l}$, and $i_x \in \{1, \dots, m\}, k_y \in \{1, \dots, n\}$,
  - $w_i := (u_{i_1}, \dots, u_{i_j} \mapsto u_{k_1}, \dots, u_{k_l}) \in x$ where $x \in L$, *typeOf(w_i)*=`Dle`, *typeOf(x)=* $S_{i_1}, \dots, S_{i_j} \rightarrow S_{k_1}, \dots, S_{k_l}$, and $i_x, k_y \in \{1, \dots, m\}$.

An *artifact instance of an artifact type* (L, P, M) is a pair (id, λ) where $id \in \mathrm{dom}_{\mathrm{id}}$, and λ is a partial mapping such that λ(*state*) is in *statesOf(M)*, and for every $x \in L$, if defined, $\lambda(x)$ is in *dom(*typeOf(x)*)*. Let *x.id* represent the id of an artifact instance. *A set of artifact instances is consistent*, if for every different x, y in the set, $x.id \neq y.id$. Let $\overline{\mathbb{A}}$ represent a set of artifact types with the following property: For every type τ in $\overline{\mathbb{A}}$, any type referenced by τ is also in $\overline{\mathbb{A}}$. Let $\overline{\Lambda}$ represent the set of all consistent sets of artifact instances.

The details of formal semantics of method executions can be found in [10]. Briefly, a method includes four types of operations: An assignment from input parameters and artifact attributes to output parameters and artifact attributes; an addition of a new mapping to a data function type attribute; accessing the value of a mapping in a data function; and checking whether a given mapping is defined for a data function typed attribute. The execution semantics of a method is defined as the sequential execution of the statements in the body. The form $m(x_1, \dots, x_k \mapsto y_1, \dots, y_l)$ represents the invocation of a method m. x_i's represent the input values. When the invocation ends, y_j's contain the output values. The method execution is assumed to be atomic and the intermediate states of the computation are not visible.

4.2 Repository, Task, Operational Model

Repositories are containers for artifacts where they await future processing. Each repository can keep many artifacts of a single artifact type. A repository provides operations for tasks to access the artifacts it contains. We assume a set $\overline{\mathbb{R}}$ of repositories each has an associated artifact type in $\overline{\mathbb{A}}$.

Definition: The following operations are defined on a repository $\mathcal{R}$ (The semantics are defined later.):

- $checkIn$: adds an artifact in $\mathcal{R}$,
- $checkAnyOut$: retrieves an artifact from $\mathcal{R}$,
- $checkOut$: retrieves a specific artifact from $\mathcal{R}$,
- $contains$: checks if $\mathcal{R}$ contains a specific artifact,
- $nonempty$: checks if $\mathcal{R}$ contains any artifact .

Let *related*($\mathbb{A}$,$\mathbb{R}$) be true if $\mathbb{A} \subseteq \overline{\mathbb{A}}$, $\mathbb{R} \subseteq \overline{\mathbb{R}}$, the associated type of every $\mathcal{R} \in \mathbb{R}$ is in $\mathbb{A}$, and for every type τ in $\mathbb{A}$, there is at least one $\mathcal{R} \in \mathbb{R}$ such that its associated type is τ.

In the remainder of the paper, we use $\mathbb{A}$, $\mathbb{R}$, Λ so that *related*($\mathbb{A}$, $\mathbb{R}$) is true and Λ is a consistent set of artifact instances each is of a type in $\mathbb{A}$.

A *repository mapping* γ *over* $\mathbb{R}$ *and* Λ is a mapping $\gamma : \mathbb{R} \rightarrow 2^{\Lambda}$ with the following properties: for every repository $\mathcal{R} \in \mathbb{R}$, $\gamma(\mathcal{R}) \subseteq \Lambda$, and for all $x \in \gamma(\mathcal{R})$, x's type is the same with $\mathcal{R}$'s associated type, and for all different $\mathcal{R}_1$, $\mathcal{R}_2 \in \mathbb{R}$, for all $y \in \gamma(\mathcal{R}_1)$, for all $z \in \gamma(\mathcal{R}_2)$, $y.id \neq z.id$.

Tasks are functional units of work carried out on artifacts. Tasks retrieve artifacts from repositories, read/update artifact data via invoking the artifact methods, and inserts artifacts into repositories.

Definition: A *task over* $\mathbb{A}$, $\mathbb{R}$ is a tuple (L, M) where

- L is a set of variables each is of an artifact type in $\mathbb{A}$, or of the type `Deq` or `Dle`.
- M = (Σ, S, s_0, S_f, T, l) is a deterministic finite state automata where:
  - Σ is a set of statements where a statement has one of the following forms:
    - $A.new(x)$
    - $\mathcal{R}.checkAnyOut(x)$
    - $\mathcal{R}.checkOut(x)$
    - $\mathcal{R}.checkIn(x)$
    - $reset(z_1, \dots, z_l)$
    - $x.m(y_1, ..., y_k \mapsto z_1, ..., z_l)$

    where $x \in L$ and *typeOf(x)=* τ for some artifact type $\tau \in \mathbb{A}$; $\mathcal{R} \in \mathbb{R}$and $\mathcal{R}$'s associated type is τ; m is a method defined by τ; each y_i is in L or a constant; each z_i is in L,
  - S is a set of states, $S_f \subseteq S$ is a set of final states, $s_0 \in S_f$ denotes the "initial" state,
  - T, the transition relation, is a subset of $(S \times (\Sigma - \{reset(\dots)\}) \times S) \cup (S_f \times \{reset(\dots)\} \times \{s_1\}) \cup (\{s_0\} \times \{reset(\dots)\} \times \{s_1\})$ for some $s_1 \in S - \{s_0\}$,
  - $l : T \rightarrow G$ is a labeling function where G is a set of guards.
- A *guard* is defined inductively as follows: *true* is a guard; for every $x, y \in L$ and a constant c, every artifact comparison over x, y and every scalar comparison over x, y, c are guards; $\mathcal{R}.contains(x)$ and $\mathcal{R}.nonempty$ are guards where $x \in L$ and *typeOf(x)*$\in \mathbb{A}$, $\mathcal{R} \in \mathbb{R}$ and $\mathcal{R}$'s associated type equals to *typeOf(x)*; $g_1 \wedge g_2$ is a guard where g_1, g_2 are guards.

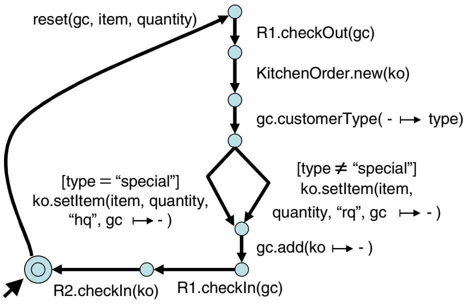

Figure 3. AddItem Task

Example: *Figure 3 illustrates our guarded finite state machine representation for the task AddItem. The task is triggered with a “reset” action which resets all the variables in the context of a task. This action also models the input from the external world; therefore, “gc”, “itemNo”, and “quantity” contains the external input values. A guest check from “open guest checks” repository is retrieved using the value of “gc”. Then, a kitchen order is created and its item number and quantity are set. If the customer is a special customer, then the order is set to have a high quality (“hq”); otherwise, it is set to have a regular quality (“rq”). Finally, both artifacts are checked in.*

A *task instance of a task* (L, M) *over* $\mathbb{A}$ *and* $\mathbb{R}$ *with respect to* Λ is a mapping π such that *(i)* $\pi(\textit{state})$ is in *statesOf*(M); *(ii)* for every $x \in L$, if defined, $\pi(x)$ is in *dom(*typeOf(x)*)*; *(iii)* if $\pi(x)$ is defined and *typeOf*$(x)$$\in \mathbb{A}$, then $\pi(x) = y.\textit{id}$ for some $y \in \Lambda$.

A task instance π of a task (L, M) *satisfies*

- an artifact comparison $x = y$ where $x, y \in L$ if $\pi(x)$ and $\pi(y)$ are defined, and $\pi(x) = \pi(y)$.
- a scalar comparison $x\theta y$ where $x, y \in L$, if $\pi(x)$ and $\pi(y)$ are defined, and $\pi(x)\theta\pi(y)$ is true.
- a scalar comparison $x\theta c$ where $x \in L$ and c is a constant, if $\pi(x)$ is defined and $\pi(x)\theta c$ is true.
- $\mathcal{R}.contains(x)$ if $\pi(x)$ is defined, and $\pi(x) = y.\textit{id}$ for some $y \in \gamma(\mathcal{R})$.
- $\mathcal{R}.nonempty$ if $\gamma(\mathcal{R})$ is not empty.
- a guard $g_1 \wedge g_2$ if π satisfies g_1 and g_2.

Let Π be a set of task instances. A *task mapping* ϕ *over* Π *and* Λ is a mapping $\phi : \Pi \rightarrow 2^{\Lambda}$ with the following property: for every task instance $\pi \in \Pi$ of a task (L, M), $\phi(\pi) \subseteq \{x \mid x \in \Lambda$ and $x.\textit{id}{=}\ \pi(y)$ for some $y \in L\}$.

We now proceed to define operational models that describe the operations of a business.

Definition: An *operational model* is a tuple ($\mathbb{A}$, $\mathbb{R}$, $\mathbb{T}$) such that *related*($\mathbb{A}$, $\mathbb{R}$) and $\mathbb{T}$ is a set of tasks over $\mathbb{A}$ and $\mathbb{R}$.

A *configuration of an operational model* ($\mathbb{A}$, $\mathbb{R}$, $\mathbb{T}$) is a tuple $(\Lambda, \Pi, \phi, \gamma)$ where $\Lambda \subseteq \overline{\Lambda}$ is a consistent set of artifact instances each is of a type in $\mathbb{A}$; Π is a set of task instances of tasks in $\mathbb{T}$ with respect to Λ; ϕ is a task mapping over Π and Λ; γ is a repository mapping over $\mathbb{R}$ and Λ such that *i)* for every task in $\mathbb{T}$, there is exactly one task instance in Π; *ii)* for all $\mathcal{R} \in \mathbb{R}$, for all $\pi \in \Pi$, for all $x \in \gamma(\mathcal{R})$, for all $y \in \gamma(\pi)$, $x.\textit{id} \neq y.\textit{id}$); *iii)* $\Lambda = \{x \mid x \in \phi(\pi)$ for $\pi \in \Pi\}$ $\cup$ $\{x \mid x \in \gamma(\mathcal{R})$ for $\mathcal{R} \in \mathbb{R}\}$. Intuitively, ϕ refers to the artifact intances that are checked out by the task instances and γ refers to the instances that are in the repositories.

4.3 Computation Semantics

In this section we sketch the semantics of the proposed formalism. Because of the page limitations, some of the details are left out. The full description of the semantics can be found in [10].

Let $\mathcal{C} = (\Lambda, \Pi, \phi, \gamma)$, $\mathcal{C}' = (\Lambda', \Pi', \phi', \gamma')$ be two configurations. Let $\pi \in \Pi$ be a task instance of a task (L, M), s be a statement such that $(\pi(\textit{state}), s,\ q) \in$ *transitionRelationOf*(M) for some $q \in$ *statesOf*(M), and g be the guard assigned to s in M. We say $\mathcal{C}'$ can be derived from $\mathcal{C}$ with s of π, denoted by $\mathcal{C} \xrightarrow{\pi, s} \mathcal{C}'$, if $\Lambda', \Pi', \phi', \gamma'$ can be derived from $\Lambda, \Pi, \phi, \gamma$ according to the rules below:

- If s has the form $A.new(x)$, and $\pi(x)$ satisfies g, and $\pi(x)$ is not defined, then:
  - Let (t, λ) be an artifact instance of the artifact type $A = (L', P', M')$ such that $t \in \mathrm{dom}_{\mathrm{id}} - \{y.\textit{id} \mid y \in \Lambda\}$, $\lambda(\textit{state}) =$ *initialStateOf*(M'), and for every $z \in L'$, $\lambda(z)$ is not defined.
  - Let π' be a task instance of the task (L, M) such that $\pi'(\textit{state}) = q$, $\pi'(x) = t$, and for every $y \in L - \{x\}$, $\pi'(y)$ is not defined if $\pi(y)$ is not defined, and $\pi'(y) = \pi(y)$, otherwise.
  - $\Pi' = \Pi - \{\pi\} \cup \{\pi'\}$, and $\Lambda' = \Lambda \cup \{(t, \lambda)\}$.
  - $\forall y \in \mathbb{R}, \gamma'(y) = \gamma(y)$. $\forall y \in \Pi - \{\pi\}, \phi'(y) = \phi(y)$, and $\phi'(\pi') = \phi(\pi) \cup \{(t, \lambda)\}$.
- For the other cases, please see [10].

A configuration $\mathcal{C} = (\Lambda, \Pi, \phi, \gamma)$ is a *root configuration* iff for every $\pi \in \Pi$, the following properties are true: *i)* Let (L, M) be the task of which π is an instance. Then, $\pi(y)$ is undefined for all $y \in L$, *ii)* $\pi(\textit{state}) =$ *initialStateOf*(M), *iii)* $\phi(\pi)$ is empty. The third property implies all artifact instances are in the repositories (i.e., Λ is $\{x \mid x \in \gamma(\mathcal{R})$ and $\mathcal{R} \in \mathbb{R}\}$, and $\{x \mid x \in \phi(\pi)$ and $\pi \in \Pi\}$ is empty).

An *execution graph* with respect to a root configuration and an operation model is a directed labeled graph defined inductively as follows: *i)* the root configuration is a node in the graph, *ii)* a configuration $\mathcal{C}'$ is in the graph if $\mathcal{C} \xrightarrow{\pi, s} \mathcal{C}'$ for some $\mathcal{C}, \pi, s$. A configuration $\mathcal{C}'$ is *reachable* from a configuration $\mathcal{C}$ if there is a path in the execution graph from $\mathcal{C}$ to $\mathcal{C}'$.

5 Analysis of Properties

In this section first we show that persistence and uniqueness are guaranteed by our formalism. Then we describe our results for checking arrival properties. Henceforth, we refer to our formalism as $OpMod$.

5.1 Persistence and Uniqueness

In this section, we define uniqueness and persistence and show that our formalism guarantees these properties.

Informally, an artifact instance is persistent with respect to an operational model and a root configuration if it is always either in a repository or referenced by a task. Let $\mathscr{F}= (\mathbb{A}, \mathbb{R}, \mathbb{T})$ be an operational model in $OpMod$, $\mathcal{C} = (\Lambda, \Pi, \phi, \gamma)$ be a root configuration of $\mathscr{F}$, $x \in \Lambda$ be an artifact instance of type (L, P, M). We say x is *never destroyed with respect to* $\mathscr{F}$ *and* $\mathcal{C}$ if there is a reachable configuration $\mathcal{C}' = (\Lambda', \Pi', \phi', \gamma')$ where $y.id = x.id$ for some $y \in \Lambda'$. We say x is *always accessible with respect to* $\mathscr{F}$ *and* $\mathcal{C}$ if there is a reachable configuration $\mathcal{C}' = (\Lambda', \Pi', \phi', \gamma')$ such that there exists an artifact instance $y \in \Lambda'$ where $x.id = y.id$ and one of the following is true: *i)* $y \in \gamma'(\mathcal{R})$ for some $\mathcal{R}\in \mathbb{R}$; *ii)* $y \in \phi'(\pi')$ and $y.id = \pi'(z)$ for some $\pi' \in \Pi'$ and $z \in L$. Finally, x *is persistent with respect to* $\mathscr{F}$ *and* $\mathcal{C}$, if x is never destroyed and always accessible with respect to $\mathscr{F}$ and $\mathcal{C}$.

Informally, an artifact instance is unique with respect to an operational model and a root configuration if it never appears in two places throughout its life cycle. Formally, x *is unique with respect to* $\mathscr{F}$ *and* $\mathcal{C}$, if there is no reachable configuration $\mathcal{C}' = (\Lambda', \Pi', \phi', \gamma')$ such that there exists an artifact instance $y \in \Lambda'$ where $x.id = y.id$, and one of the following is true: *i)* $y \in \gamma'(\mathcal{R}_1)$ and $y \in \gamma'(\mathcal{R}_2)$, for different $\mathcal{R}_1, \mathcal{R}_2 \in \mathbb{R}$; *ii)* $y \in \phi'(\pi')$ and $y \in \phi'(\pi'')$ for different $\pi', \pi'' \in \Pi'$; *iii)* $y \in \gamma'(\mathcal{R}_1)$ and $y \in \phi'(\pi')$ for $\mathcal{R}_1 \in \mathbb{R}$, $\pi' \in \Pi'$.

We have the following result.

Theorem 5.1 *For each operational model* $\mathscr{F}$ *defined in OpMod, for each root configuration* $\mathcal{C}$ *of* $\mathscr{F}$*, and for each artifact instance* x *in* $\mathcal{C}$*,* x *is unique and persistent with respect to* $\mathscr{F}$ *and* $\mathcal{C}$*.*

5.2 Verification of Arrival Properties

Informally, an artifact can arrive into a repository if there is a way to process the artifact so that it eventually appears in a repository. Let $\mathscr{F}= (\mathbb{A}, \mathbb{R}, \mathbb{T})$ be an operational model in $OpMod$, $\mathcal{C} = (\Lambda, \Pi, \phi, \gamma)$ be a root configuration of $\mathscr{F}$, $x \in \Lambda$ be an artifact instance of type (L, P, M), $\mathcal{R}$ be a repository in $\mathbb{R}$. We say x *can arrive into* $\mathcal{R}$ *with respect to* $\mathscr{F}$ *and* $\mathcal{C}$, if there is a reachable configuration $\mathcal{C}' = (\Lambda', \Pi', \phi', \gamma')$, and an artifact instance $y \in \Lambda'$ such that $x.id = y.id$ and $y \in \gamma'(\mathcal{R})$.

We first show that the expressiveness of the proposed formalism leads the verification of the arrival properties to be unsolvable. We obtain this result using a restricted version of the formalism. Let $OpMod^{f,i,=}$ represent the operational models defined in $OpMod$ where $\mathrm{dom}_{\leqslant}$, $\mathrm{dom}_{=}$ are finite, $\mathrm{dom}_{\mathrm{id}}$ is infinite, only guard expression allowed is a scalar comparison, and no data function type is used. The next result shows that answering an arrival property is as difficult as answering the halting problem of Turing Machines.

Lemma 5.2 *For every Turing Machine* M *and an input* w*, there exists an operational model* $\mathscr{F}$ *defined in* $OpMod^{f,i,=}$ *and a root configuration* $\mathcal{C}$ *of* $\mathscr{F}$ *such that* M *halts on* w *if and only if an artifact instance* x *can arrive into a repository* $\mathcal{R}$ *with respect to* $\mathscr{F}$ *and* $\mathcal{C}$*.*

Proof: [Sketch] The main idea is to show that a Turing machine can be simulated with an operational model. Each machine cell can be represented with an artifact instance. The connection of each cell to its neighbor cells can be represented with 2 attributes of type $\mathrm{dom}_{\mathrm{id}}$, pointing to left and right cells. Cell contents can be stored in another artifact attribute of type $\mathrm{dom}_{\leqslant}$. The current state of the machine can be stored as another artifact instance with one attribute. All the cells are stored in the same repository except the current cell, which is stored in a separate repository. M's finite control can be represented with a task. The task checks out the current cell, and the current state, and depending on the content of the cell, it makes the decision about the next cell to be current, and makes the appropriate check-ins and check-outs to be ready for the next configuration. If the finite control specifies that M should halt, then the task moves an artifact to a special repository. Therefore, an artifact arrives into the repository if and only if M halts. ∎

Lemma 5.2 leads to the following result:

Theorem 5.3 *Given an operational model* $\mathscr{F}$ *defined in* $OpMod$*, a root configuration* $\mathcal{C}$ *of* $\mathscr{F}$*, an artifact instance* x *of* $\mathcal{C}$*, and a repository* $\mathcal{R}$ *of* $\mathscr{F}$*, it is not decidable to check if* x *can arrive into a repository* $\mathcal{R}$ *with respect to* $\mathscr{F}$ *and* $\mathcal{C}$*.*

Next we show that arrival properties are decidable in a restricted version of $OpMod$. Let $OpMod^{b,T}$ represent the operational models that are defined in $OpMod$ such that the number of artifacts that can be created (i.e., the number of invocations of the action new is bounded) and the only guard expression allowed is *true*. An arrival property can be checked with a reachability analysis on the execution graph. During the construction of the execution graph, the parameters of the $reset$ actions should be replaced with all possible values from the domains to make sure that the reachability analysis works on all possible cases. Since the domains are infinite, the number of configurations can be infinite. However, it can be shown that the values of the artifact attributes and task variables do not affect the arrival properties when the only guard allowed is *true*. Combining this observation with the restriction that there can be only bounded number of artifact instances leads to the fact that we only need a bounded number of values when we construct the execution graph to check an arrival property. Therefore, we have the following result:

Theorem 5.4 *Given an operational model* $\mathscr{F}$ *defined in* $OpMod^{b,T}$*, a root configuration* $\mathcal{C}$ *of* $\mathscr{F}$*, an artifact instance* x *of* $\mathcal{C}$*, and a repository* $\mathcal{R}$ *of* $\mathscr{F}$*, it is decidable to check if* x *can arrive into* $\mathcal{R}$ *with respect to* $\mathscr{F}$ *and* $\mathcal{C}$ *in EXPTIME.*

6 Conclusion and Future Work

We believe the results we presented in this paper will help the development of automatic static analysis tools for operational models. Businesses today have to quickly respond to changing business requirements. Having a formal model for business operations not only can help in the design of accurate implementations but also can enable the ability to analyze the effects of a change in the business operations. Such an analysis would ease the evolution of business operations and facilitate the quick response to the new business requirements. Another direction this work can lead to is the development of formal analysis techniques for business performance management. Artifacts provide a good granularity in terms of a unit of data that a business processes; this facilitates the formulation of many performance related questions at the right granularity with respect to a business user such as how many guest checks the restaurant can process, how long it would take the restaurant to process a kitchen order from creation to archival.

Acknowledgements: The authors benefit from the conversations with Kumar Bhaskaran, Henry Chang, Rick Hull, Oscar H. Ibarra, Jun-Jang Jeng, Santhosh Kumaran, Anil Nigam, and Frederick Wu.

References

[1] S. Abiteboul, P. Kanellakis, and E. Waller. Method schemas. In *PODS*, pages 16–27, 1990.

[2] A. Ailamaki, Y. Ioannidis, and M. Livny. Scientific workflow management by database management. In *SSDBM*, 1998.

[3] K. Bhattacharya, R. Guttman, K. Lymann, F. F. H. III, S. Kumaran, P. Nandi, F. Wu, P. Athma, C. Freiberg, L. Johannsen, and A. Staudt. A model-driven approach to industrializing discovery processes in pharmaceutical research. *IBM Systems Journal*, 44(1):145–162, 2005.

[4] Business Process Modeling Notation (BPMN).

[5] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. A system for specification and verification of interactive, data-driven web applications. In *SIGMOD*, 2006.

[6] J. Edwards, D. Jackson, and E. Torlak. A type system for object models. In *Proc. of Foundations of Software Engineering (FSE)*, 2004.

[7] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of web service compositions. In *Proc. of Int. Conf. on Automated Software Engineering*, 2003.

[8] X. Fu, T. Bultan, and J. Su. Analysis of interacting bpel web services. In *Proc. Int. World Wide Web Conf. (WWW)*, 2004.

[9] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–154, Apr. 1995.

[10] C. E. Gerede, K. Bhattacharya, and J. Su. Static analysis of business artifact-centric operational models: Extended version. `www.cs.ucsb.edu/~gerede/soca2007ExtendedVersion.pdf`.

[11] S. Ginsburg and K. Tanaka. Computation-tuple sequences and object histories. *ACM Transaction on Database Systems (TODS)*, 11(2), 1986.

[12] 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.

[13] R. Hull, K. Tanaka, and M. Yoshikawa. Behavior analysis of object-oriented databases: Method structure, exeuction trees, and reachability. In *FODO*, 1988.

[14] M. Jackson and G. Twaddle. *Business Process Implementation Building Workflow Systems*. Addison-Wesley, ACM Press Books, Boston, 1997.

[15] R. King and D. McLeod. A database design methodology and tool for information systems. *ACM Trans. on Information Systems*, 3(1):2–21, Jan. 1985.

[16] M. Koshkina and F. van Breugel. Modeling and verifying web service orchestration by means of the concurrency workbench. 29(5), 2004.

[17] F. Leymann and D. Roller. Business process management with flowmark. In *Proc. of COMPCON*, 1994.

[18] C. Medeiros, G. Vossen, and M. Weske. Wasa: a workflow-based architecture to support scientific database applications. In *Proc. 6th DEXA Conference*, 1995.

[19] J. P. Morrison. *Flow-Based Programming*. Van Nostrand ReinHold, New York, 1994.

[20] J. Mylopoulos, P. Bernstein, and H. Wong. A language facility for designing database-intensive applications. *ACM Trans. Database Syst.*, 5(2):185–207, June 1980.

[21] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of web services. In *Proc. Int. World Wide Web Conf. (WWW)*, 2002.

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

[23] Object Constraint Language v2.0. `http://www.omg.org/docs/formal/06-05-01.pdf`, May 2006.

[24] B. Schlingloff, A. Martens, and K. Schmidt. Modeling and model checking web services. *Electronic Notes in Theoretical Computer Science: Issue on Logic and Communication Multi-Agent Systems*, 126:3–26, 2005.

[25] R. E. Strom and S. Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Transaction on Software Engineering (TSE)*, 12(1), 1986.

[26] J. Su. Dynamic constraints and object migration. *Theoretical Comput. Sci.*, 184(1-2):195–236, 1997.

[27] Workflow management coalition.