Specification and Verification of Artifact Behaviors in Business Process Models*

Cagdas E. Gerede and Jianwen Su

Department of Computer Science
University of California at Santa Barbara
Santa Barbara, CA 93106
{gerede, su}@cs.ucsb.edu

Abstract. SOA has influenced business process modeling and management. Recent business process models have elevated data representation to the same level as control flows, for example, the artifact-centric business process models allow the life cycle properties of artifacts (data objects) to be specified and analyzed. In this paper, we develop a specification language ABSL based on computation tree logic for artifact life cycle behaviors (e.g., reachability). We show that given a business model and starting configuration, it can be decided if an ABSL sentence is satisfied when the domains are bounded, and if an ABSL-core (sublanguage of ABSL) sentence is satisfied when the domains are totally ordered but unbounded. We also show that if the starting configuration is not given, ABSL(-core) is still decidable if the number of artifacts is bounded with bounded (resp. unbounded but ordered) domains.

1 Introduction

Business process modeling has received considerable attention from research communities in and related to computer science. This is a natural consequence of the trend that computer and software systems have found rapidly increasing usage in all aspects of business process management. The fundamental principle of service oriented architecture (SOA) to design software systems based on composition of a *flexible* assembly of *services* has already influenced many business operations today. We argue that the SOA principle will continue to impact on several key aspects of business process management, including business process modeling, design, integration, and evolution aspects. This paper makes a significant step in advancing SOA techniques for business process management by focusing on how to specify *dynamic* properties on data being processed, and on how to verify these properties.

Business process modeling is a foundation for design and management of business processes. Two key aspects of business process modeling are a formal framework that well integrates both *control flow* and *data*, and a set of tools to assist all aspects of a business process life cycle. A typical business process life cycle includes at least a design phase where the main concerns are around "correct" realization of business logic in a resource constrained environment, and an operational phase where a main objective is to optimize and improve the realization during the execution (operation). Traditional business process models emphasize heavily on control flow, leaving the data design in an

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

auxiliary role if not as an afterthought. Recently, it has been argued that the consideration of data design should be elevated to the same level as control flows [14,9,4,11,1]. In our earlier efforts [9,4], we have developed artifact centric business process models and studied verification of ad hoc properties of the models.

Intuitively, *business artifacts* (or simply *artifacts*) are data objects whose manipulations define in an important way the underlying processes in a business model. Not only the past and current practice of business process specification naturally embodies the artifacts, recent engineering and development efforts (e.g., at IBM Services Division) have already adopted the artifact approach in the process of design and analysis of business models[5,12,10]. An important distinction between artifact centric models and traditional data flow (computational) models is that the notion of the life cycle of the data objects is prominent in the former, while not existing in the latter.

In the initial attempt [9], our main focus was on assembling together a business process and analyzing several execution properties including reachability. In doing so, we essentially augmented object oriented classes with states to represent artifact classes, and use guarded finite state automata to capture (the logic of) entities that carry out the work in a business model. In another approach [4], we focus more on the life cycle of artifacts and evolution of business (process) logic. In that study, we used services to model logical activities that can be executed and a declarative approach to represent a business model as a set of business rules. The two models are closely related but different. A detailed comparison of two models can be found in [8].

Business analysts need to verify whether artifact-centric business process models satisfy certain artifact properties. These properties reflect the requirements to meet business needs. Currently, a business analyst takes a process model and a property, and reason about the process model to see if the model satisfies the property. This reasoning process is not only tedious and nontrivial but it is also repetitive and can be automated. To address this problem, in this paper we develop a logic language based on computational tree logic [7], called Artifact Behavior Specification Language (ABSL). We also study the verification of properties specified in ABSL. We use the model of [9] (without finite functions and no “new" action) as the basis for the language, expecting that ABSL and technical results developed here be easily adapted to the model of [4].

The main technical results in this paper include:

1. The temporal logic based language ABSL for specifying life cycle properties of artifacts.
2. Decidability results of ABSL for a given operational model and a starting configuration for bounded domains.
3. Decidability results of ABSL-core (a sublanguage of ABSL) for a given operational model and a starting configuration for unbounded but ordered domains (i.e., with a total order).
4. Decidability results of ABSL (ABSL-core) for a given operational model and a bound for the number of artifacts, with bounded (resp. unbounded ordered) domains.

This paper is organized as follows. In Section 2, we overview artifact-centric operational model proposed in [9]. In Section 3, we propose a language for specifying artifact behaviors (ABSL). In Section 4.1, we show the decidability results of ABSL for bounded domains. In Section 4.2, we show the decidability results of ABSL-core for unbounded but ordered domains. In Section 4.3, we show the decidability results of ABSL (ABSL-core) for bounded (resp., unbounded ordered) domains. We conclude

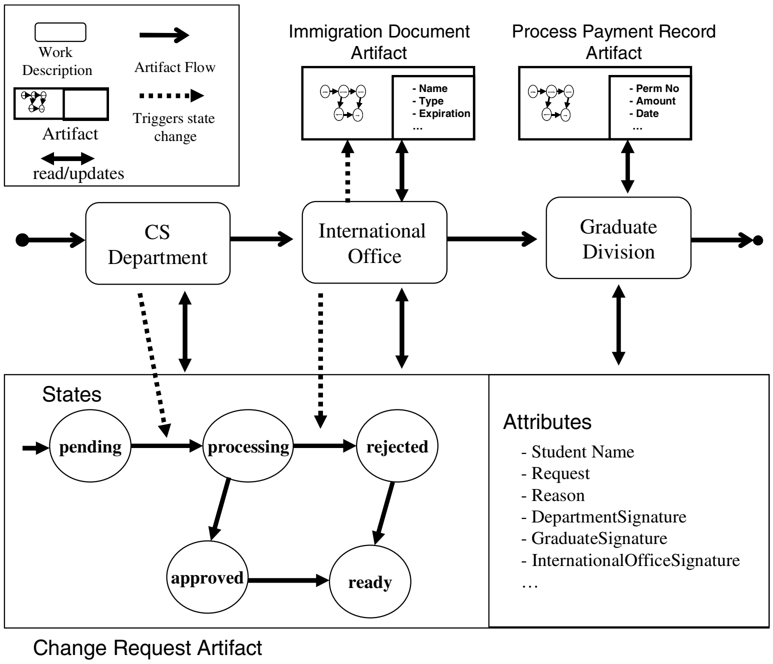

Fig. 1. An artifact-based view of a process model

the paper in Section 5. Due to space limitations, the detailed proofs and some technical definitions are omitted. More detailed discussion can be found in [8].

2 Overview: Artifact-Centric Operational Models

In this section, we briefly describe the terminology and the constructs in artifact-centric modeling. More formal and detailed discussion can be found in [8]. Artifact-centric models consist of 3 key constructs: Business artifacts, business work descriptions, and repositories [13] (In this study, the term task type and the term task are renamed to work description, and work).

Definition 1. *A* (business) artifact type T *is a tuple* (V, P, M) *such that*

- V *is a set of attributes of primitive types (such as String, real, or artifact ids).*
- P *is a set of* methods *with distinct names.*
- M *is a finite state machine and its transitions are labeled with method names from* P*.*
- *A* method *in* P *is a tuple* $(\text{name}, I, O, \text{body})$ *where* I *(*input paremeters*) and* O *(*output parameters*) are pairwise disjoint set of variables of primitive types, and the body is a sequence of statements each is of the form* $x := y$ *where* $x \in V \cup O$*,* $y \in V \cup I$*.*

An *artifact* $\boldsymbol{a}$ is an instance of an artifact type and it contains the current state of the artifact and the values of the attributes. It also has a unique id.

Repository: A *repository* describes a waiting shelf or a storage for an artifact.

Work Description: A *(business) work description* describes the work acting upon an artifact by which a business role adds measurable business value to this artifact.

Definition 2. *A* work description W *is a tuple* (V, M) *where*

- V *is a set of variables of primitive types (e.g., String, real, or artifact ids) and artifact types.*
- $M = (\Sigma, S, s_0, S_f, \delta, l)$ *is a deterministic finite state machine where:*
  - Σ *is a finite set of statements (*actions*) where a statement has one of the following forms:*

- $\mathtt{R}.checkOut(x)$	- $\mathtt{R}.checkIn(x)$
- $\mathtt{R}.checkOut(x)\ with\ id = y$	- $x.m(z_1, ..., z_k \mapsto z'_1, ..., z'_n)$
- $read(u)$	- $reset$

    where R *is a repository,* m *is a method name, and* $x, y, z_i, z'_j, u \in V$ *such that* x *is a variable of an artifact type,* y *is a variable of an artifact-id type,* z_i*'s are variables of primitive types and constants,* z'_j*'s are variables of primitive types,* u *is a variable of scalar types, and* $k \geq 0, n \geq 0$.
  - 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,*
  - δ*, the transition relation, is a subset of* $(S \times (\Sigma - \{reset\}) \times S) \cup (S_f \times \{reset\} \times \{s_0\})$,
  - $l : \delta \to G$ *is a labeling function where* G *is a set of guards.*
  - *A* guard *is defined inductively as follows:* false, true *are guards; for every* $x \in V$ *of scalar type (such as String, real), and a constant c, every scalar comparison between* x, c *is a guard (such as* $x > c$, $x = c$, $x \neq c$*); for every* $x, y \in L$ *of scalar type, every scalar comparison between* x, y *is a guard (such as* $x > y$, $x = y$, $x \neq y$*;* $\mathtt{R}.nonempty$ *is a guard for every repository* R*; and* $g_1 \wedge g_2$ *is a guard for every pair of guards* g_1 *and* g_2.

The type of actions in a work description and their intuitive meaning is provided below:

- $\mathtt{R}.checkOut(x)$: check out a (random) artifact from repository R;
- $\mathtt{R}.checkOut(x)\ with\ id = y$: check out an artifact from repository R with id y;
- $\mathtt{R}.checkIn(x)$: check an artifact in repostory R;
- $x.m(z_1, \ldots, z_k \mapsto z'_1, \ldots, z'_n)$: invoke the method m of the artifact held by the variable x with the input parameters $z_1, \ldots, z_k$ and expect the output in the variables $z'_1, \ldots, z'_n$;
- $read(u)$: read a scalar value (such as String, real) from external environment;
- $reset$: uninitialize the values of all variables.

A *work* $\boldsymbol{w}$ (an instance of a work description) contains the values of work description variables, and the current state of the work. There is one work for each work description at run time.

Operational Model: An *(artifact-centric) operational model* $\mathcal{O}$ is a tuple $(\mathbf{T}, \mathbf{R}, \mathbf{W})$ where $\mathbf{T}$ is a set of artifact types, $\mathbf{R}$ is a set of repositories, $\mathbf{W}$ is a set of work descriptions.

A *configuration* of an operational model can be thought as a snapshot of the process at runtime, and it contains a set of artifacts, a set of work, and a set of repositories. Let $\mathcal{C}, \mathcal{C}'$ be two configurations. We say $\mathcal{C}'$ can be derived from $\mathcal{C}$, denoted as $\mathcal{C}' \rightarrow \mathcal{C}$, if $\mathcal{C}'$ can be produced as a result of a work executing an action.

A *root configuration* is a configuration where all the artifacts are in the repositories, and each work is in its initial state.

An *execution graph* with respect to a root configuration $\mathcal{C}^0$ is a Kripke structure (G^0, G, H) where $G^0 = \{\mathcal{C}^0\}$ is the initial state, G is the set of configurations, and H is the transition relation such that $(\mathcal{C}, \mathcal{C}') \in H$ if $\mathcal{C} \rightarrow \mathcal{C}'$.

Example 1. The scenario in Figure 1 describes an operational model for the processing of the *Student Change Request Artifact*. This artifact is used by students to request various changes such as the addition of an emphasis to the student's degree, or a committee member addition, or extension of a degree deadline. The approval of this artifact requires the signatures of the student's department, the graduate division, and the international office if the student holds a student visa. The processing of this artifact requires the payment of a processing fee which is tracked through *Processing Payment Record Artifact*. In addition, if the student holds a student visa, then the international office requires the verification of the student's the *Immigration Document Artifact* before they approve the change request. In addition, the graduate division and international office partially rely on the decision of the department, therefore, they require the student's department to process the artifact first.

Based on the specification, some of the desirable properties of this process can be enumerated as follows:

– Every approved change request artifact must have Department and Graduate Division signatures. Every change request artifact submitted by international students requires a signature from International Office.
– International Office and Graduate Division should not sign a change request artifact until Department signs it.
– Every change request artifact that Department does not approve should not be approved by Graduate Division or International Office.
– Every change request artifact by students under 18 requires her parents' signature.
– If a change request artifact is rejected by one authority, then the artifact shouldn't be processed any further by any authority.
– Every change request artifact by international students requires an encounter with the student's immigration document artifact at International Office.
– The approval of every change request artifact requires an encounter with a paid processing payment record artifact at Graduate Division.
– When a change request artifact is approved, the next action on the artifact should be the delivery to the student.

The verification of an artifact-centric process model at design time is crucial to avoid higher costs of breakdown, debugging and fixing during runtime. The verification requires a formalism to describe artifact-centric models, and a specification language to describe the properties the model should have. Therefore, in the next section, we propose a language to specify artifact behaviors. We show how to verify an artifact-centric process model with respect to artifact behavior specifications.

3 A Language for Specifying Artifact Behaviors: ABSL

Given an operational model $\mathcal{O} = (\mathbf{T}, \mathbf{R}, \mathbf{W})$, the set of symbols of ABSL consists of, in addition to the standard logical symbols (,), $\wedge$, $\neg$, $\forall$, and constants:

- variables (each is associated with a type in $\mathbf{T}$ or with a scalar type such as String, real);
- propositions, one for each state of a work description
- unary predicate symbols:
  - one for each work description ($\mathtt{W}(x)$ if x is checked out by W);
  - one for each repository ($\mathtt{R}(x)$ if x is checked out by R);
  - one for each state of each artifact type;
  - two for each attribute
    - $*$ a *read*-predicate, ($Read_A(x)$ if the attribute A of artifact x is read);
    - $*$ a *defined*-predicate ($Defined_A(x)$ if the value of the attribute A of artifact x is defined);
- binary predicates $Equal, NotEqual, GreaterThan, LessThan$ (e.g., $Equal$ (x, y) if x, y are not undefined, and x equals to y);
- 0-ary function symbols one for each work description variable;
- unary function symbols:
  - one for each attribute ($A(x)$ is the value of the attribute A of artifact x);
  - ID ($\mathtt{ID}(x)$ is id of artifact x).

A *term* is a constant, or a variable, or an expression of the form v, or $\mathtt{ID}(x_1)$, or $A(x_2)$ where v is a 0-ary function symbol, A is a unary function symbol, x_1 is a variable of an artifact type, x_2 is a variable of an artifact type containing an attribute A.

An atomic formula is an expression of the following forms:

true	q_{state}	$\mathtt{R}(x)$	$\mathtt{W}(x)$
$p_{state}(x)$	$Read_A(x)$	$Defined_A(x)$	$Equal(t_1, t_2)$
$NotEqual(t_1, t_2)$	$GreaterThan(t_1, t_2)$	$LessThan(t_1, t_2)$	

where q_{state} is a proposition corresponding to the state of a work description; p_{state} is a unary predicate corresponding to the state of an artifact type; x is a variable of an artifact type; t_1, t_2 are terms.

An artifact centric model describes the behaviors of all artifacts related to the underlying business operations that is being designed[13]. The artifact-centric approach reflects itself in the way the desirable process model behaviors are described, and these descriptions “focus" on behaviors of individual artifacts. To capture this requirement in ABSL, we propose a temporal operators based on computational tree logic (CTL) [7] operators (also inspired by the clock operator of the temporal language Sugar[3]):

- $\boldsymbol{EN}_{@\boldsymbol{a}}\psi$ (“Next" w.r.t $\boldsymbol{a}$): requires that the formula ψ holds the next time the artifact $\boldsymbol{a}$ is involved;
- $\boldsymbol{EG}_{@\boldsymbol{a}}\ \psi$ (“Globally" w.r.t. $\boldsymbol{a}$) : requires that the formula ψ holds every time the artifact $\boldsymbol{a}$ is involved,
- $\boldsymbol{E}\psi_1\ \boldsymbol{U}_{@\boldsymbol{a}}\ \psi_2$ (“Until" w.r.t. $\boldsymbol{a}$): requires that there is a time when $\boldsymbol{a}$ is involved and ψ_1 holds, and at all preceeding times that $\boldsymbol{a}$ is involved, ψ_2 holds.

The *family of formulas* in ABSL is the set of expressions such that if ψ_1 and ψ_2 are formulas, then so are

$$\psi_1 \wedge \psi_2,\ \neg\psi_1,\ \forall x_1\, \psi_1,\ \boldsymbol{EN}_{@x}\psi_1,\ \boldsymbol{E}\psi_1\boldsymbol{U}_{@x}\psi_2,\ \boldsymbol{EG}_{@x}\psi_1$$

where x, x_1 are variables, and the type of the variable x is an artifact type.

The notion of free and bounded variables are defined in the standard manner. A *sentence* is a formula without any free variables.

Example 2. ABSL can formulate all the properties described in Example 1. Here we illustrate some of them. We use $\boldsymbol{F}_{@x}\psi$ to mean *true* $\boldsymbol{U}_{@x}\psi$.

– Every approved change request artifact must have the department signature.
$\forall x \neg \boldsymbol{EF}_{@x}\ (approved(x) \wedge \neg Defined_{DeptSignature}(x))$
– Any change request artifact that Department does not approve should not be approved by Graduate Division.
$\forall x\, \neg \boldsymbol{EF}_{@x}(AtDept(x) \wedge Equal(decision, \text{“}reject\text{”}) \wedge \boldsymbol{EF}_{@x} Defined_{DeptSignature}(x))$.
– International Office should not sign a form until Department signs it.
$\forall x\ \neg \boldsymbol{E}\ \neg Defined_{DeptSignature}(x)\ \boldsymbol{U}_{@x} Defined_{IntOfficeSignature}(x)$
– Any form by international students requires an *encounter* with the student's immigration document at International Office. Let $\psi_1 \vee \psi_2$ represent $\neg(\neg\psi_1 \vee \psi_2)$ and $\exists y$ represent $\neg\forall\neg y$.
$\forall x Equal(international(x), \text{“}false\text{”})\ \vee (\boldsymbol{EF}_{@x} AtIntOff(x) \wedge \exists y AtIntOff(y) \wedge Equal(immigrationDoc(x), ID(y))) \vee (\exists y \boldsymbol{EF}_{@y} AtIntOff(x) \wedge AtIntOff(y) \wedge Equal(immigrationDoc(x), ID(y)).$

Definition 3. ABSL-core *is a sub-language of ABSL consisting of formulas using only variables of artifact types.*

Before we explain the semantics of ABSL, we would like to mention the technical differences of ABSL from computational tree logic (CTL) [7] and the temporal language Sugar [3]. In ABSL, differently from CTL, we have the focus operator @. The focus operator may sound similar to the clock operator of Sugar; however there is a big semantic difference. The clock operator in Sugar causes the projection of execution paths with respect to a clock (causes to consider only the configurations at which the clock holds). On the other hand, the focus operator doesn't modify the original execution path but allows to skip configurations which are not focused.

3.1 Semantics

In order to describe the semantics of ABSL, we extend the concept of a configuration with two more pieces of information. First, we record the artifact an action on which leads to this configuration. This is used to point out the “artifact-focused" configurations. Second, we record the attributes that are read. Intuition behind this is to allow the formulation of the properties about the “usefulness" of attributes during the processing of the artifacts

These are formalized as follows:

Definition 4. (Extended Configuration) For a configuration $\mathcal{C}$ of an operational model $\mathcal{O}$, an *extended configuration* of $\mathcal{O}$ is a tuple $\mathcal{D} = (\mathcal{C}, \alpha, \theta)$ where α is a subset of $\{(\boldsymbol{a}, A) \mid \boldsymbol{a}$ is an artifact in $\mathcal{C}$ and A is an attribute of $\boldsymbol{a}\}$, θ is a constant partial function, and if is defined it is an artifact in $\mathcal{C}$ (Conceptually α represents the set of attributes that are read, and θ represents the artifact such that the configuration is reached as a result of an action on that artifact). An *extended root configuration* $\mathcal{D}^0 = (\mathcal{C}, \alpha, \theta)$ of $\mathcal{O}$ is an extended configuration of $\mathcal{O}$ such that $\mathcal{C}$ is a root configuration of $\mathcal{O}$, and α is empty, and θ is not defined.

Given two configurations $\mathcal{C}_1, \mathcal{C}_2$ such that $\mathcal{C}_2$ can be derived from $\mathcal{C}_1$, we say $\mathcal{C}_2$ *focuses on the artifact* a, if the derivation is due to an action on $\boldsymbol{a}$ (i.e., a check-in or check-out of $\boldsymbol{a}$, or a method invocation on $\boldsymbol{a}$).

Next we extend the concept of derivation to extended configurations. An extended configuration $\mathcal{D}_2 = (\mathcal{C}_2, \alpha_2, \theta_2)$ can be can be *derived* from $\mathcal{D}_1 = (\mathcal{C}_1, \alpha_1, \theta_1)$, denoted as $\mathcal{D}_1 \to \mathcal{D}_2$, if the configuration $\mathcal{C}_2$ can be derived from the configuration $\mathcal{C}_1$, and α_2 is the union of α_1 and the set of attributes which are read in the derivation, and θ_2 is defined and equals to the artifact that the derivation $\mathcal{C}_1 \to \mathcal{C}_2$ focuses on We say the derivation $\mathcal{D}_1 \to \mathcal{D}_2$ *focuses on the artifact* $\boldsymbol{a}$, if θ equals to $\boldsymbol{a}$.

An extended configuration $\mathcal{D}$ is *reachable* from another extended configuration $\mathcal{D}'$ if there exists a finite positive number of extended configurations $\mathcal{D}_1, \ldots, \mathcal{D}_k$ such that $\mathcal{D} \to \mathcal{D}_1, \mathcal{D}_1 \to \mathcal{D}_2, \ldots, \mathcal{D}_{k-1} \to \mathcal{D}_k, \mathcal{D}_k \to \mathcal{D}'$.

Next we describe the semantics of terms of the language with an example.

Example 3. For an extended configuration $\mathcal{D}$, we describe the semantics of formulas containing no path and temporal operators on a simple example formula $\forall x\ ready(x) \wedge AtStudent(x) \to \exists y\ Equal(DeptSignature(x), y) \wedge Read_{Reason}(x)$. Assuming that the type of x is a change request artifact, and the type of y is String, x ranges over all change request artifacts in $\mathcal{D}$, and y ranges over all String value domain. Then, for every artifact $\boldsymbol{a}$, $ready(\boldsymbol{a})$ is true when the state of $\boldsymbol{a}$ is "ready"; $AtStudent(\boldsymbol{a})$ is true if $\boldsymbol{a}$ is in the repository $AtStudent$; $DeptSignature(\boldsymbol{a})$, if defined, evaluates to the value of the attribute $DeptSignature$ of $\boldsymbol{a}$; $Equal$ is true there exists a String value that equals to $DeptSignature$ of $\boldsymbol{a}$; $Read_{Reason}(\boldsymbol{a})$ is true if $Reason$ attribute of $\boldsymbol{a}$ is in the read set. $\mathcal{D}$ *satisfies* the formula if the formula evaluates to true.

Definition 5. An *extended execution graph* $\mathscr{E}$ of an operational model $\mathcal{O}$ and an extended root configuration $\mathcal{D}^0$ is a Kripke structure (G^0, G, H) where $G^0 = \{\mathcal{D}^0\}$ is the initial state, G is the set of extended configurations, and H is the transition relation such that $(\mathcal{D}, \mathcal{D}') \in H$ if $\mathcal{D} \to \mathcal{D}'$, and $(\mathcal{D}, \mathcal{D}) \in H$ if $\neg\exists\ \mathcal{D}'$ s.t. $\mathcal{D} \to \mathcal{D}'$. A *path* ρ in $\mathscr{E}$ is an infinite sequence of extended configurations $\mathcal{D}_1, \mathcal{D}_2, \ldots$ such that for every $i \geq 0$, $(\mathcal{D}_i, \mathcal{D}_{i+1}) \in H$.

Next, we informally describe the semantics of formulas containing temporal operators. Let $\mathscr{E}$ be an extended execution graph of an operational model $\mathcal{O}$, and an extended root configuration $\mathcal{D}^0$. Also, let $\mathcal{D}$ be an extended configuration in $\mathscr{E}$, and $\boldsymbol{a}$ be an artifact in $\mathcal{D}$. Then, $(\mathscr{E}, \mathcal{D})$ *satisfies*

- $\boldsymbol{EN}_{@\boldsymbol{a}}\psi_1$ if there is a path from $\mathcal{D}$ in $\mathscr{E}$ on which the next $\boldsymbol{a}$-focused extended configuration satisfies ψ_1.
- $\boldsymbol{EG}_{@\boldsymbol{a}}\psi_1$ if there is a path from $\mathcal{D}$ in $\mathscr{E}$ s.t. every $\boldsymbol{a}$-focused extended configuration on the path satisfies ψ_1.
- $\boldsymbol{E}\psi_1\boldsymbol{U}_{@\boldsymbol{a}}\psi_2$ if there is a path from $\mathcal{D}$ in $\mathscr{E}$ s.t. there is an extended configuration satisfies ψ_2, and ψ_1 is true at all preceeding extended configurations.

$(\mathcal{O}, \mathcal{D}^0)$ *satisfies* a formula ψ, denoted as $(\mathcal{O}, \mathcal{D}^0) \models \psi$, if $(\mathscr{E}, \mathcal{D}^0)$ satisfies ψ.

4 Verification of Artifact Behaviors

Verifying artifact behaviors such as reachability is proven to be undecidable with the ability of creating new artifacts[9]. Although decidability result was obtained there when the ability of creating new artifacts is removed, extending the result to ABSL is not obvious because of two main reasons. First, the domains of the artifact attributes, and work description variables can be unbounded. Second, even the number of these attributes and variables are bounded, the work descriptions can read external values and invent infinite number of new values during the computation.

In the following sections, we develop decidability results for different cases.

4.1 Bounded Domains

The main result of this section is:

Theorem 1. *For an operational model $\mathcal{O}$, an ABSL sentence ψ, an extended root configuration $\mathcal{D}^0$ of $\mathcal{O}$, it is decidable to check whether $(\mathcal{O}, \mathcal{D}^0)$ satisfies ψ, when the domains are bounded.*

The rest of this section is devoted to prove this result.

Step 1: Given an ABSL sentence, we first eliminate the variables in the sentence. As an example, let our sentence be $\forall x\ pending(x) \wedge \forall y\ \neg Equal(signature(x), y)$ where x quantifies over artifacts, and y is over scalar domain Eq (e.g. Strings). Let the extended root configuration contains three artifacts $\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3$, and let Eq contains two elements c_1, c_2. Then, we eliminate x by replacing it with all possible values, and we take the conjunction of the expression since x is universally quantified. We can eliminate y similarly. The variable eliminated version of the sentence becomes $\bigwedge_{c_1,c_2} \bigwedge_{\boldsymbol{a}_1,\boldsymbol{a}_2}$ $pending(\boldsymbol{a}_i) \wedge \neg Equal(signature(\boldsymbol{a}_i), c_j)$. The approach is extended to the other expressions. The following can be proven:

Lemma 1. *For an operational model $\mathcal{O}$, an extended root configuration $\mathcal{D}^0$ of $\mathcal{O}$, and an ABSL sentence ψ, $(\mathcal{O}, \mathcal{D}^0)$ satisfies ψ iff $(\mathcal{O}, \mathcal{D}^0)$ satisfies the variable eliminated version of ψ.*

Step 2: For a variable eliminated sentence, we define a set of propositions. This set depends on the formula, the operational model, and the root configuration. For instance, for each repository R, for each artifact $\boldsymbol{a}$ appearing in the extended root configuration, we have a proposition $p[\mathtt{R}(\boldsymbol{a})]$ and this proposition is true in an extended configuration if $\boldsymbol{a}$ is located in R in the configuration. For every attribute A, every artifact $\boldsymbol{a}$, and every constant c appearing in the formula or in a work description of the operational model we have a proposition $p[A(\boldsymbol{a}) = c]$ (and $p[A(\boldsymbol{a}) < c]$ if the domain of A is ordered). $p[A(\boldsymbol{a}) = c]$ is true in an extended configuration if the value of A of $\boldsymbol{a}$ in the configuration equals to c (resp., if it is less than c). Also, for every artifact $\boldsymbol{a}$, we have a proposition $p[\boldsymbol{a}]$ and it is true in an extended configuration if the configuration is $\boldsymbol{a}$-focused. The approach is extended to other predicates including artifact states, work description states, and work description variables.

Step 3: We translate a variable eliminated ABSL sentence to a propositional branching temporal logic (CTL) formula. We assume some familiarity with CTL [6]. We, first, replace each atomic formula by a propositional formula. For example, if the atomic formula is $\mathtt{R}(\boldsymbol{a})$ where R is a predicate corresponds to a repository, and $\boldsymbol{a}$ is an artifact, then we replace it with the proposition $p[\mathtt{R}(\boldsymbol{a})]$. The same technique is naturally extended to the other atomic formulas involving unary predicates and binary predicates. For each atomic formula involving a binary predicate, we replace the binary predicate with a propositional formula. For instance, $Equal(A_1(\boldsymbol{a}_1), A_2(\boldsymbol{a}_2))$ is replaced by $p[Defined_{A_1}(\boldsymbol{a}_1)] \wedge p[Defined_{A_2}(\boldsymbol{a}_2)] \wedge p[A_1(\boldsymbol{a}_1) = A_2(\boldsymbol{a}_2)]$.

For the path and temporal operators, we do the following translation:

– $\boldsymbol{EN}_{@\boldsymbol{a}}\, \psi \;\; \Rightarrow \;\; \boldsymbol{E}\,\boldsymbol{X}\,\boldsymbol{E}\neg p[\boldsymbol{a}]\,\boldsymbol{U}\,(p[\boldsymbol{a}] \wedge \psi)$
– $\boldsymbol{EG}_{@\boldsymbol{a}}\, \psi \Rightarrow \;\; (p[\boldsymbol{a}] \wedge \psi \wedge \boldsymbol{EXEG}(p[\boldsymbol{a}] \rightarrow \psi)) \vee$
$(\neg p[\boldsymbol{a}] \wedge (\boldsymbol{EXE}\neg p[\boldsymbol{a}]\boldsymbol{U}(p[\boldsymbol{a}] \wedge \psi)) \wedge (\boldsymbol{EXEG}(p[\boldsymbol{a}] \rightarrow \psi)))$
– $\boldsymbol{E}\,\psi_1\,\boldsymbol{U}_{@x}\,\psi_2 \Rightarrow \;\; \boldsymbol{E}\,(p[\boldsymbol{a}] \rightarrow \psi_1)\,\boldsymbol{U}\,(p[\boldsymbol{a}] \wedge \psi_2)$

As a result, we obtain a propositional CTL formula. Then, we create a *labeled* version of the extended execution graph such that each extended configuration $\mathcal{D}$ in the graph is labeled with the set of propositions that hold in $\mathcal{D}$.

We can prove the following:

Lemma 2. *For an operational model $\mathcal{O}$, an extended root configuration $\mathcal{D}^0$ of $\mathcal{O}$, Let $\mathcal{E}$ be the extended execution graph of $\mathcal{O}$ and $\mathcal{D}^0$. For a variable eliminated ABSL sentence ψ, $(\mathcal{E}, \mathcal{D}^0)$ satisfies ψ iff $(\mathcal{E}^L, \mathcal{D}^0)$ satisfies the CTL version of ψ, where $\mathcal{E}^L$ is the labeled version of $\mathcal{E}$.*

Coming back to Theorem 1, the proof idea is as follows: Since the domains are bounded, the size of the extended execution graph is finite. Due to Lemma 1 and Lemma 2, the problem of checking if an extended execution graph satisfies an ABSL sentence can be translated to a CTL model checking problem. The size of the extended execution graph is finite, and the decidability of model checking on finite structures is known[6]; therefore, the verification of an ABSL sentence is decidable.

4.2 Unbounded Domains

The main result of this section is:

Theorem 2. *For an operational model $\mathcal{O}$, an ABSL-core sentence ψ, an extended root configuration $\mathcal{D}^0$ of $\mathcal{O}$, it is decidable to check whether $(\mathcal{O}, \mathcal{D}^0)$ satisfies ψ, when the domains are unbounded.*

The rest of this section is devoted to prove this result.

Given an ABSL-core sentence and an extended root configuration, similar to the bounded domains case, we eliminate the variables from the sentence and obtain a CTL formula. Note that the CTL formula we obtain has a finite length, because the quantifiers in the ABSL-core sentences can only be used over artifact variables and the number of artifacts in the extended root configuration is finite.

The following can be proven:

Lemma 3. *For an operational model $\mathcal{O}$, an extended root configuration $\mathcal{D}^0$ of $\mathcal{O}$, Let $\mathcal{E}$ be the extended execution graph of $\mathcal{O}$ and $\mathcal{D}^0$. For a variable eliminated ABSL-core sentence ψ, $(\mathcal{E}, \mathcal{D}^0)$ satisfies ψ iff $(\mathcal{E}^L, \mathcal{D}^0)$ satisfies the CTL version of ψ where $\mathcal{E}^L$ is the labeled version of $\mathcal{E}$.*

It is not straightforward to obtain a result like Theorem 1, because the domains are not bounded and therefore the size of the extended execution graph is not finite. Interestingly we show that we can obtain a finite abstraction of the infinite space and verify a given sentence on this finite abstraction. In order to do this, we use an approach similar to the region approach used for the decidability results of timed-automata [2]. Timed-automata were introduced to model the behavior of real-time systems, which annotates state-transition graphs with timing constraints using finitely many real-valued clock variables. While in timed-automata, the infiniteness results from incrementing the clocks, in our model, it results from reading values from external environment.

We first define a binary relation among extended configurations. For an extended root configuration $\mathcal{D}^0$, let $R_{\mathcal{D}^0}$ be a binary relation over extended configurations such that two extended configurations are in the relation if they have the same set of artifacts with $\mathcal{D}^0$, and they satisfy the same set of propositions (Conceptually, two configurations obey the same *total ordering* of the artifact attribues and work description variables).

Lemma 4. *For an extended root configuration $\mathcal{D}^0$, $R_{\mathcal{D}^0}$ is an equivalence relation.*

The equivalence class[7] of an extended configuration $\mathcal{D}$, denoted as $[\mathcal{D}]_{\mathcal{D}^0}$, with respect to an extended root configuration $\mathcal{D}^0$ and the relation $R_{\mathcal{D}^0}$, is defined in the standard manner. Note that the number of such equivalence classes is finite.

Lemma 5. *For every pair of extended configurations $\mathcal{D}, \mathcal{D}'$ with $[\mathcal{D}]_{\mathcal{D}^0} = [\mathcal{D}']_{\mathcal{D}^0}$, it is true that for every extended configuration $\mathcal{D}'$ that satisfies $\mathcal{D} \rightarrow \mathcal{D}'$, there exists an extended configuration $\mathcal{D}'_1$ that satisfies both $\mathcal{D}' \rightarrow \mathcal{D}'_1$ and $[\mathcal{D}']_{\mathcal{D}^0} = [\mathcal{D}]_{\mathcal{D}^0}$.*

Definition 6. For an extended execution graph $\mathscr{E} = (\{\mathcal{D}^0\}, G, H)$, the *region graph* of $\mathscr{E}$ is a Kripke structure $(\{\mathcal{D}^0\}, G', H')$ where G' and H' is defined inductively as follows: $\mathcal{D}^0$ is in G'; for every $\mathcal{D}_1 \in G'$, and for every $\mathcal{D}_2$ such that $(\mathcal{D}_1, \mathcal{D}_2) \in H$, if there does not exist $\mathcal{D}_3$ such that $(\mathcal{D}_1, \mathcal{D}_3) \in H'$ and $[\mathcal{D}_2]_{\mathcal{D}^0} = [\mathcal{D}_3]_{\mathcal{D}^0}$, then $(\mathcal{D}_1, \mathcal{D}_2) \in H'$ and $\mathcal{D}_2 \in G$.

The size of the region graph is finite, because the number of equivalence classes is finite. We extend the region graph to a *labeled region graph* by labeling its configurations with the set of atomic propositions that hold in that configuration.

Based on Lemma 4 and Lemma 5, we can show that there is a bisimulation relation[7] between every labeled execution graph and its corresponding labeled region graph.

Lemma 6. *Given an extended root configuration $\mathcal{D}^0$, $R_{\mathcal{D}^0}$ forms a bisimulation relation between $\mathcal{D}^0$'s (labeled) extended execution graph and $\mathcal{D}^0$'s (labeled) region graph.*

Coming back to Theorem 2, the proof idea is as follows: As it is given in Lemma 6, there is a bisimulation relation between a labeled execution graph and its labeled region region graph. Therefore, a CTL formula is satisfied by a labeled execution graph iff it is satisfied by its labeled region graph[6]. The size of a region graph is finite, and the decidability of the the model checking on finite structures is known. This is combined with Lemma 3 concludes that the verification of an ABSL-core sentence with unbounded domains is decidable.

4.3 Verification with Bounded Number of Artifacts

An extended root configuration $\mathcal{D}^0$ is k-bounded if the number of artifacts in $\mathcal{D}^0$ is at most k, where k is a positive integer. For an operational model $\mathcal{O}$, and a positive integer

k, $(\mathcal{O}, k)$ *satisfies* an ABSL(-core) sentence ψ, if for every extended root configuration $\mathcal{D}^0$, $(\mathcal{O}, \mathcal{D}^0)$ satisfies ψ.

The following can be proven:

Theorem 3. *For an operational model $\mathcal{O}$, a positive integer k,*

- *it is decidable to check whether $(\mathcal{O}, k)$ satisfies an ABSL sentence ψ with bounded domains;*
- *it is decidable to check whether $(\mathcal{O}, k)$ satisfies an ABSL-core sentence ψ with unbounded domains.*

5 Conclusion

In this paper, we proposed a logic language based on computational tree logic [7], to specify artifact behaviors in artifact-centric process models. We showed decidability results of our language for different cases. While we provide key insights on how artifact-centric view can affect the specification of desirable business properties, extensions and refinements of our language and results will be beneficial.

References

1. Aalst, W., Weske, M., Grnbauer, D.: Case handling: a new paradigm for business process support. Data and Knowledge Engineering 53, 129–162 (2005)
2. Alur, R.: Timed automata. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 8–22. Springer, Heidelberg (1999)
3. Beer, I., Ben-David, S., Eisner, C., Fisman, D., Gringauze, A., Rodeh, Y.: The temporal logic sugar. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, Springer, Heidelberg (2001)
4. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. Business Process Management (BPM) (2007)
5. Bhattacharya, K., Guttman, R., Lymann, K., Heath III, F.F., Kumaran, S., Nandi, P., Wu, F., Athma, P., Freiberg, C., Johannsen, L., Staudt, A.: A model-driven approach to industrializing discovery processes in pharmaceutical research. IBM Systems Journal 44(1), 145–162 (2005)
6. Clarke, E., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge, Massachusetts (2000)
7. Emerson, E.A.: Temporal and modal logic. In: Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B,ch. 7, pp. 995–1072. North Holland, Amsterdam (1990)
8. Gerede, C.E.: Modeling, Analysis, and Composition of Business Processes. PhD thesis, Dept. of Computer Science, University of California at Santa Barbara (2007)
9. Gerede, C.E., Bhattacharya, K., Su, J.: Static analysis of business artifact-centric operational models. In: SOCA. IEEE International Conference on Service-Oriented Computing and Applications, IEEE Computer Society Press, Los Alamitos (2007)
10. Kumaran, S., Nandi, P., Heath, T., Bhaskaran, K., Das, R.: Adoc-oriented programming. In: Symposium on Applications and the Internet (SAINT) (2003)
11. Liu, R., Bhattacharya, K., Wu, F.Y.: Modeling business contexture and behavior using business artifacts. In: CAiSE. LNCS, vol. 4495, Springer, Heidelberg (2007)
12. Nandi, P., Kumaran, S.: Adaptive business objects a new component model for business integration. In: Int. Conf. on Enterprise Information Systems (2005)
13. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
14. Wang, J., Kumar, A.: A framework for document-driven workflow systems. In: Business Process Management, pp. 285–301 (2005)