Information and Software Technology 62 (2015) 187–197

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Automating correctness verification of artifact-centric business process models

Diana Borrego [*], Rafael M. Gasca, María Teresa Gómez-López

University of Seville, Department of Computer Languages and Systems, Av Reina Mercedes S/N, 41012 Seville, Spain

ARTICLE INFO

Article history:
Received 25 July 2014
Received in revised form 19 February 2015
Accepted 21 February 2015
Available online 3 March 2015

Keywords:
Artifact-centric business process model
Verification
Constraint programming

ABSTRACT

Context: The artifact-centric methodology has emerged as a new paradigm to support business process management over the last few years. This way, business processes are described from the point of view of the artifacts that are manipulated during the process.
Objective: One of the research challenges in this area is the verification of the correctness of this kind of business process models where the model is formed of various artifacts that interact among them.
Method: In this paper, we propose a fully automated approach for verifying correctness of artifact-centric business process models, taking into account that the state (lifecycle) and the values of each artifact (numerical data described by pre and postconditions) influence in the values and the state of the others. The lifecycles of the artifacts and the numerical data managed are modeled by using the Constraint Programming paradigm, an Artificial Intelligence technique.
Results: Two correctness notions for artifact-centric business process models are distinguished (reachability and weak termination), and novel verification algorithms are developed to check them. The algorithms are complete: neither false positives nor false negatives are generated. Moreover, the algorithms offer precise diagnosis of the detected errors, indicating the execution causing the error where the lifecycle gets stuck.
Conclusion: To the best of our knowledge, this paper presents the first verification approach for artifact-centric business process models that integrates pre and postconditions, which define the behavior of the services, and numerical data verification when the model is formed of more than one artifact. The approach can detect errors not detectable with other approaches.

1. Introduction

Nowadays, organizations model their operations with business processes. Traditionally, business processes are modeled as activity-centric business process models [1] in which activities are focused on and data just serve as inputs and outputs of some services. They follow the imperative principles, implying that the workflow of the activities can be defined at design time. But for some types of problems, it is easier to represent how the data are modified during the process execution instead of the activities that execute the data evolution.

For this reason, the artifact-centric methodology (data-centric approach) has emerged as a new paradigm to support business process management, where business artifacts appeared for the necessity of enrich the business process model with information about data [2], providing a way for understanding the interplay between data and process. Artifacts are business-relevant objects that are created, evolved, and (typically) archived as they pass through a business, combining both data aspects and process aspects into a holistic unit [3].

Artifact-centric modeling establishes data objects (called artifacts) and their lifecycles as focus of the business process modeling. This type of modeling is inherently declarative: the control flow of the business process is not explicitly modeled, but follows from the lifecycles of the artifacts [4].

The lifecycle represents how the state of an artifact may evolve over the time. The different activities change the state of the artifact and the values of the data associated to each artifact; these may be manual (i.e. carried out by a human participant of the process) or automatic (i.e. by a web service). The evolution of the artifacts implies a change of the state and the values of the data, until a goal state of an artifact is reached. One of the reasons

* Corresponding author. Tel.: +34 954 556 234; fax: +34 954 557 139.
E-mail addresses: dianabn@us.es (D. Borrego), gasca@us.es (R.M. Gasca), maytegomez@us.es (M.T. Gómez-López).

http://dx.doi.org/10.1016/j.infsof.2015.02.010
0950-5849/© 2015 Elsevier B.V. All rights reserved.

why the artifact-centric paradigm facilitates the process description is the capacity to model the relations between objects with different cardinalities, not only 1-to-1 relations. This modeling capabilities are not entirely supported in activity-centric scenarios. For instance, BPMN 2.0 [5] (currently wide accepted activity-centric notation) allows to easily represent multi-instance activities and pools (processes), but with some limitations: (i) relations between different processes can only be expressed as hierarchies, where a process can invoke multiple instances of its subprocess; (ii) the return value of an executed sub-process instance is only accessible when its execution finishes, not allowing the interaction of another process during the execution; and (iii) the definition of Data Objects, Data Inputs, Data Outputs, Sets, and Data Associations in BPMN 2.0 allows to specify collections of elements, but it does not permit data instance differentiation or to include the relation between the data objects between them. Regarding this last limitation, the proposal in [6] proposes an extension of BPMN data objects adding annotations to manage data dependency and instance differentiation. However, these annotations are very low level representations and no significant for business stakeholders. Based on this idea, the work in [7] uses more complex objects that can involve N-to-M relations, using all the advantages of ORM to incorporate the data objects in a more natural way into the activity-centric business processes.

When more than one artifact is involved in the process, it is possible that a combination of services and data values violate the policies of the business. In order to avoid this situation at runtime, it is possible to detect some of these possible errors even at design time. Specifically, the errors derived from an incorrect design of the model. In spite of the unknown runtime data in the design time phase, our proposal is able to perform a data verification of the models by means of the use of mandatory domains of values, which can be obtained from previous executions and/or knowledge from experts. Making use of this information, it is possible to determine the existence of certain errors in the structural and data perspectives of the model before it is deployed.

The goal of this paper is to develop an approach for verifying the correctness of artifact-centric business process models at design time, including the state relation between the artifacts of the model, and the data values that define the relations between them. To develop the automatic verification, we model the services formally using pre and postconditions over the data associated to the artifacts' states. To analyze the correctness of the model, it is necessary to study when the services can be executed. A service can be executed if the evolution of the lifecycle of the artifact arrives at the service and its precondition is satisfied. Upon completion, the service delivers data that satisfies its postconditions. The no satisfiability of a pre or postcondition can cause that the lifecycle gets stuck at a service and fails.

The automatic verification is performed using Artificial Intelligence techniques, both to compute the possible evolutions on the lifecycles and to model the pre and postconditions of the services as numerical constraints.

This paper is organized as follows. Section 2 presents a motivating example to illustrate the concepts of reachability and weak-termination. Section 3 introduces artifact graphs and artifact union graphs as a formal model for artifact-centric business process models, and defines reachability and weak-termination on artifact union graphs. Section 4 defines the CSP formulation of an artifact union graph. The process of verification is explained, and two algorithms are presented. The verification of the motivating example is performed, and their tractability is discussed. Section 5 presents an overview of related work found in the literature. And finally, conclusions are drawn and future work is proposed in Section 6.

2. A motivating example

In this paper, the example presented in [8] has been enriched, including characteristics that cannot be described using the activity-centric paradigm. The original example describes the handling of a conference by an organizing committee. At the beginning of the process, the establishment of the conference rate is performed, even before the submission period is open. Then, the external services that are needed during the conference are booked (e.g. gala dinner, coffee breaks and proceedings), at the same time that the sponsorship money collection is carried out and the origin of the guest speaker is decided.

Likewise, the authors submit the papers which are received by the organizing committee, and are reviewed by members of the scientific committee (reviewers) in order to select the papers accepted for the conference. The decision about the approval or rejection of the papers is notified to the authors. Meanwhile, the conference registration period is open, which will remain open until the conference ends, so that the authors can register when they already know if their papers will be presented at the conference. Finally, the number of conference attendees is known, and the payment for the booked services is performed.

Since the relations between the execution instances of the different processes are not all 1-to-1, this presented scenario is not modellable by means of the activity-centric paradigm due to the limitations explained in Section 1. That is, while the tasks performed by the organizing committee only requires an instance of execution, several instances of the different reviewers and submitted papers are running simultaneously.

The described process can be represented with five artifacts: (1) the Finances artifact, involving the tasks regarding the economic decisions, performed by the organizing committee; (2) the Organization artifact, entailing the tasks concerning the publication of papers and registration of attendees, performed by the program committee; (3) the Paper artifact, including the tasks performed by the authors of the submitted papers; (4) the Reviewer artifact, containing the tasks executed by the reviewers of the papers; and (5), the Registration artifact, allowing the registration of attendees to the conference. The associations between these five artifacts present different cardinalities, existing relations 1-to-1, 1-to-N and N-to-M between the artifact instances, as it is shown in Fig. 1.

As the execution of the tasks changing the states of the artifacts takes place, some data are consumed and produced by reading and writing the attributes of the mentioned artifacts. Those attributes are listed in Table 1 with their corresponding meaning. As mentioned, the behaviors of the tasks are defined by means of pre and postconditions over the artifacts and attributes.

Semantically ordered tasks of different and independently modeled artifacts may be executed in any order. But other combinations are not desirable, for instance, it makes no sense to perform the payment of the gala dinner at the restaurant before we know the number of conference attendees. Therefore, the executions have to be constrained using policies and goal states.

A policy tells us the constraints that must comply the state changes within one artifact, or between different artifacts. These changes can be related to the values of the attributes, to the number of artifact instances, or because a service has been executed over the artifact. Finally, goal states restrict final states by reducing those combinations of artifacts' final states that should be considered successful.

For our motivating example, there are ten policies restricting the inter-artifact behavior:

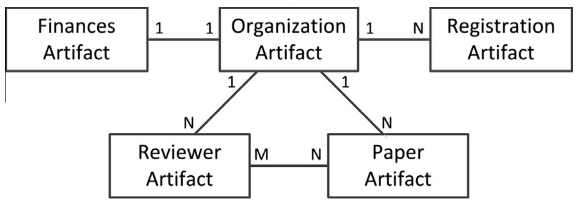

Fig. 1. Cardinalities between artifacts.

- **P1**: The payment of the booked services cannot be carried out until the number of conference attendees is known.
- **P2**: The process of reception of papers starts once that the registration fee has been established.
- **P3**: The registration process can start once it has been opened.
- **P4**: All registration processes should finished before the registration period is closed.
- **P5**: Papers are received by the organizing committee once the authors have submitted them.
- **P6**: The reviewers start their revisions once the organizing committee send them the papers to review.
- **P7**: A paper is approved or rejected based on the decision of the reviewers.
- **P8**: The selected papers are known after all reviewers inform about their revisions.
- **P9**: The established registration fee should be less than or equal to the cost of registering at the conference.
- **P10**: All submitted papers should be reviewed.

The objective of this paper applied to the example is to know if the conference described by means of artifacts can be celebrated successfully and satisfying the policies defined.

3. Computational model for representation of artifact-centric business process models

In order to verify the artifact-centric business process models at design time, we distinguish between two types of correctness to check:

- Reachability. An artifact-centric business process model is reachable if there is a possible trace of execution where every state can be reached, so there is an evolution of the lifecycle in which the state is visited.
- Weak-termination. It is a correctness criterion that ensures that a goal state is always reachable from every reachable state.

Table 1
Artifacts with their corresponding attributes.

Attribute	Representation
Finances artifact	
regFee	Conference registration fee
sponsorship	External contributions to support the event
dinner	Gala dinner cost
lunch	Cost of each lunch served during the conference
others	Budget for other expenses, such as social events
guestSpeaker	Money to spend in inviting a guest speaker
Organization artifact	
selectedPapers	List of papers to be published
Paper artifact	
oid	Unique identifier of each submitted paper
approval	State of the submitted paper
Reviewer artifact	
papersToReview	List of papers to review by each reviewer
Registration artifact	
regCost	Cost of registering at the conference

In order to analyze the reachability and weak-termination of artifact-centric business process models, we propose an approach based on the combination of graph-theory and Constraint Programming [9].

In this section, we define the artifact graphs, including the definition of the union of artifacts to obtain a global model. Next, the notions of *data instance subgraph* and correctness for artifact graphs are introduced. Finally, the concepts of *partial* and *complete instance subgraph* are presented.

3.1. Definitions

In order to develop our computational model, we use the specification defined in the framework BALSA [10] as a basis.

In order to verify the correctness of the artifact-centric model, it is necessary to verify that each state of the artifacts' lifecycles are reachable. Likewise, to ensure the correctness of a model, it should be weakly terminating, which implies that a goal state is always reachable from every reachable state. To perform the verification of both aspects, we represent artifacts as graphs, where the nodes are the corresponding states and services of the artifacts, and the lifecycle is defined by means of directed edges. This kind of formalization is chosen due to it facilitates the modeling to associate pre and postconditions to the services.

Definition 1 (*Artifact graph*). An artifact graph is a tuple $AG = \langle G, Data, \Omega\rangle$ where:

- The tuple $G = \langle St, Ser, E\rangle$ is a graph, where:
  - St is a set of artifact states;
  - Ser is a set of actions or services;
  - $E \subseteq (St \times Ser) \cup (Ser \times St)$ is a set of edges which determines precedence relation;
- The tuple $Data = \langle ID, At, pre, post\rangle$ collects all information regarding the data managed in the artifact, where:
  - ID is the identifier of the artifact;
  - At is a set of typed attributes;
  - $pre, post$: $Ser \rightarrow C(At)$, being $C(At)$ the set of constraints on At, assign to each action its precondition or postcondition, respectively;
- Ω is a set composed of subsets of St representing the end points in the lifecycle of the artifact.

For the motivating example previously mentioned, Fig. 2 depicts the Finances Artifact, represented as artifact graph, where the states are depicted as circles and the services as squares.

The auxiliary functions $inedge, outedge : N \rightarrow \mathcal{P}(E)$, defined in [8], are used to map each node to a set of edges. For a node n ($n \in \{St \cup Ser\}$), $inedge(n)$ is the set of edges entering n, while $outedge(n)$ is the set of edges leaving n. Formally, $inedge(n) = \{(x, y) \in E | y = n\}$ and $outedge(n) = \{(x, y) \in E | x = n\}$. We use subscripts to identify the different elements of $inedge(n)$ and $outedge(n)$. For example, if $inedge(n) = \{e_1, e_2\}$, then $inedge_1(n) = e_1$ and $inedge_2(n) = e_2$.

As mentioned, an artifact evolves through the states in its lifecycle, fulfilling some execution constraints:

- The evolution from a state st_x to a state st_y takes place when the service ser connecting them is executed. That is, when the pre and postcondition of ser are satisfied.
- If a state st_x presents more than one incoming edge, i.e. $|inedge(st_x)| > 1$, the change to st_x in the lifecycle takes place when the execution of one of the services connected to st_x through one of those incoming edges is performed.

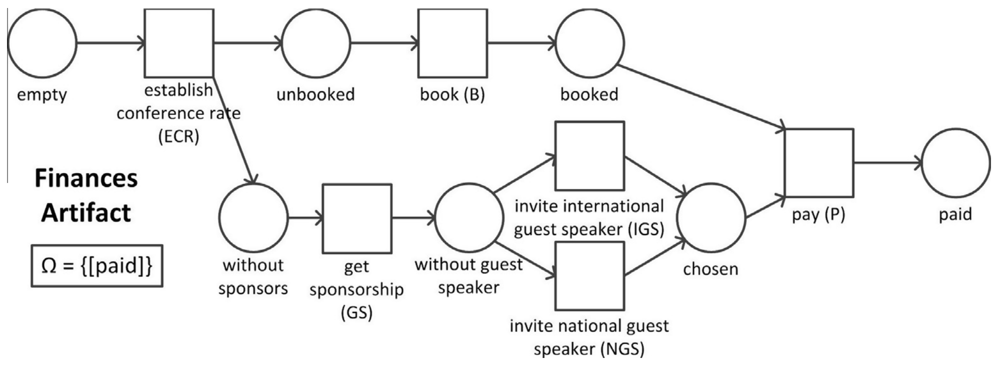

Fig. 2. Artifact finances.

- If a state st_x presents more than one outgoing edge, i.e. $|outedge(st_x)| > 1$, the evolution of the lifecycle can continue only through one of those outgoing edges, specifically the one connecting to the service that will be executed next.
- If a service *ser* presents more than one incoming edge, i.e. $|inedge(ser)| > 1$, *ser* can be executed only when all the states connected to *ser* through one of those incoming edges have been reached, and when the precondition of *ser* is satisfied.
- If a service *ser* presents more than one outgoing edge, i.e. $|outedge(ser)| > 1$, when the execution of *ser* is finished, and its postcondition is satisfied, all the states connected to *ser* through those outgoing edges are activated.

The constraints that should be satisfied during the execution of the services (that is, preconditions, postconditions and some policies) are linear or polynomial equations or inequations over artifact instances and the attributes in *At*.

Let $at \in At$ be an attribute, let *id* be an artifact identifier, let *int_val* be an integer value, let *nat_val* be a natural value, and let *float_val* be a float value. The set of constraints on *At* and *ID*, is generated by the following grammar in BNF:

$$\begin{aligned}
Constraint &::= Constraint\ BOOL\_OP\ Constraint\\
&\quad |Atomic\_Constraint|\neg Constraint\\
&\quad |Constraint \rightarrow Constraint\\
BOOL\_OP &::= \ \wedge|\vee\\
Atomic\_Constraint &::= Function\ RELATIONAL\_OP\ Function\\
Function &::= Function\ ARITHMETIC\_OP\ Function\\
&\quad |\#\ id\\
&\quad |AGGREGATION\_OP\ Set\\
&\quad |id.at|int\_val|nat\_val|float\_val\\
Set &::= SET\_OP\ id.at|id.at\\
RELATIONAL\_OP &::= \ <|\leqslant|==|>|\geqslant\\
ARITHMETIC\_OP &::= \ +|-|*\\
AGGREGATION\_OP &::= Card|Min|Max|Avg|\Sigma\\
SET\_OP &::= \bigcup|\bigcap
\end{aligned}$$

This grammar allows constraints that are conjunctions, disjunctions (*BOOL_OP*) and/or implications ($\rightarrow$) of equations or inequations (*RELATIONAL_OP*) containing certain operators:

- #: is the operator that represents the instance count. That is, #*id* is the number of executed instances of the artifact whose identifier is *id*.
- *ARITHMETIC_OP*: arithmetic operators.
- *AGGREGATION_OP*: operators over sets of attributes. Specifically, the cardinality (*Card*), minimum (*Min*), maximum (*Min*), average (*Avg*) and sum (Σ).
- *SET_OP*: union ($\bigcup$) and intersection ($\bigcap$) of sets of attributes.

In order to verify the correctness of a complete artifact-centric business process model, the global model needs to be considered, taking into account all the artifacts composing it at the same time. To this end, we need to define the global model as the union of all artifacts. This union is established by the policies, which limit the coordinated execution of artifacts lifecycles, avoiding the appearance of independent executions which lead to undesired situations, such as incorrect goal states.

Two types of policies can be distinguished: (1) the Structural Policies (cf. Definition 2) expressing constraints on the relation between states and/or services of different artifacts; and (2) Data Policies (cf. Definition 3) expressing invariant conditions over the data (i.e. attributes) managed by the different artifacts in the complete model.

Definition 2 (*Structural Policy*). A Structural Policy is a graph, represented by the tuple $SP = \langle St,\ Ser,\ E\rangle$.

Definition 3 (*Data Policy*). A Data Policy DP is a set of constraints on the attributes in *At*, so $DP \subseteq C(At)$.

Regarding the ten aforementioned policies for the motivating example described in Section 2:

- Policies from P1 to P8 are structural policies, which result in edges between services and/or states from different artifacts.
- Policies P9 and P10 are data policies, which can be formalized by means of the invariants that should be satisfied at all times during the execution of the artifacts:

$$\begin{aligned}
&DP: \{Finances.regFee * \#Registration\\
&\leqslant \sum_i Registration_i.regCost, Card(\bigcup_i Reviewer_i.papersToReview)\\
&== \#Paper\}
\end{aligned}$$

In this way, the global model can be formalized as the union of artifact graphs and policies, giving rise to a new artifact graph.

Definition 4 (*Artifact union graph*). Let $\{A_1,\ldots,A_n\}$ be artifacts with dependencies between them, and let $\{SP_1,\ldots,SP_m\}$ and $\{DP_1,\ldots,DP_p\}$ be structural and data policies expressing those dependencies. An artifact-centric model is defined as the union of artifacts and policies, called artifact union graph, $\{(\bigcup_{i=1}^{n} A_i) \cup (\bigcup_{j=1}^{m} SP_j) \cup (\bigcup_{k=1}^{p} DP_k)\} = \langle G, Data, Inv, \Omega\rangle$, being $G = \langle St, Ser, E\rangle$ and $Data = \langle At, pre, post\rangle$, where:

- $St = (\bigcup_{i=1}^{n} St_i) \cup (\bigcup_{j=1}^{m} St_j)$, i.e. the states of the artifact union graph are the union of the states of all artifacts and structural policies;

- $Ser = (\bigcup_{i=1}^{n} Ser_i) \cup (\bigcup_{j=1}^{m} Ser_j)$, i.e. the services of the artifact union graph are the union of the services of all artifacts and structural policies;
- $E = (\bigcup_{i=1}^{n} E_i) \cup (\bigcup_{j=1}^{m} E_j)$, i.e. the edges of the artifact union graph are the union of the edges of all artifacts and structural policies;
- $At = \bigcup_{i=1}^{n} At_i$, i.e. the attributes of the artifact union graph are the union of the attributes of all artifacts;
- $pre = \bigcup_{i=1}^{n} pre_i$, i.e. the preconditions of the artifact union graph are the union of the preconditions of all artifacts;
- $post = \bigcup_{i=1}^{n} post_i$, i.e. the postconditions of the artifact union graph are the union of the postconditions of all artifacts;
- $Inv = \bigcup_{k=1}^{p} DP_k$, i.e. the invariants of the artifact union graph are the union of the constraints defined in the data policies;
- and, for the goal states in Ω, they are obtained as the composition of states in $\{\Omega_1, \ldots, \Omega_n\}$ which still are final states in the artifact union, i.e. being $\Omega'_i \subseteq \Omega_i \mid \forall st \in \Omega'_i : outedge(st) = 0$, then $\Omega = \{\Omega'_1 \oplus \ldots \oplus \Omega'_n\}$.

For our motivating example, the artifact union graph of the artifacts Finances, Organization, Paper, Reviewer and Registration, presented in Section 2, is shown in Fig. 3. The edges and states drawn with broken lines are the representation of the structural policies (P1 to P8).

For the services in the artifact union graph shown in Fig. 3, the pre and postconditions are listed in Table 2.[1]

For a global artifact-centric business process model to be verified as correct, all the artifacts composing it, including all policies defining dependencies between them, should be verified as correct.

3.2. Analysis of the verification of the correctness

As mentioned, the verification of the correctness of artifact-centric business process models at design time implies to check two different aspects: (1) the reachability of each state in an artifact union lifecycle; and (2) the weak-termination of the global model.

The reachability of each state depends on the pre and postconditions of the services executed before it in a partial execution of the business process in which the state is reached. And the weak-termination of a global model implies the verification of complete executions traversing trough each state, checking the satisfiability of the pre and postconditions of the services executed in them. To define it formally, we adapt the notion of partial instance subgraph and complete instance subgraph, introduced in [8], which refers to a particular instance of a workflow graph used for the analysis of the business process correctness. In this paper, it is a subset of states and services that are visited in a particular execution of an artifact union graph. In the case of the partial instance subgraph, it is obtained by traversing the graph from the initial state, using the following rules: (1) if one incoming edge of a state is visited, then the state is visited too; (2) if all incoming edges of a service are visited, then the service is visited too; and (3) if a service is visited, all its outgoing edges are visited too.

In the case of the complete instance subgraph, it is obtained by traversing the graph from the initial node using the rules for partial instance subgraphs plus the next rule: (4) if a state is visited, then one and only one of their outgoing edges is visited too.

The correctness of a partial or complete instance subgraph implies the satisfiability of the pre and postconditions of the visited services. Therefore, to check the reachability of a state *st*, it is necessary to find a valuation of attributes satisfying those pre and postconditions for a partial instance subgraph reaching *st*. Likewise, for a global model to be weak-terminating, it is necessary to check complete instance subgraphs visiting each possible state in the artifact union graph, finding a valuation of attributes which satisfies the pre and postconditions of the visited services. To define it formally, we adapt the definition of data instance subgraph from [8] in order to include the valuation of attributes to the instance subgraph (partial or complete).

[1] Only the services with pre and/or postconditions different to *true* are shown.

Definition 5 (*Data instance subgraph*). A data instance subgraph of an artifact union graph $\langle G, Data, Inv, \Omega \rangle$ is a tuple $\langle G', Data', Inv', \Omega', v \rangle$ where

- $\langle G', Data', Inv', \Omega' \rangle$ is an instance subgraph with $St' \subseteq St, Ser' \subseteq Ser, E' \subseteq E, At' \subseteq At, pre' \subseteq pre, post' \subseteq post, Inv' \subseteq Inv, \Omega' \subseteq \Omega$, and
- v is an assignment of values to attributes in At', so that each attribute $at \in At'$ is instantiated with a value $v(at)$. Those values should satisfy the pre and postconditions in pre' and $post'$ respectively (i.e. the pre and postconditions of the services in Ser').

Only considering the reading/writing of attributes by services, we can identify two types of error that can occur in an instance subgraph:

- Data is missing if is read by a service, but not written in any preceding service in the instance subgraph.
- Data is conflicting when an attribute is written in services which can be executed in parallel in an instance subgraph. In that case, the last executed service overwrites the value previously written by the other service.

The detection of these types of errors should be carried out in a similar way as it was performed in [8].

However, taking into account the pre and postconditions defining the behavior of the services, other types of errors can occur, such as the pre and/or postcondition of a service cannot be satisfied for any valuation of attributes, not being possible to reach the subsequent states. To formalize these errors, we define the reachability and weak-termination of a global model (Definitions 8 and 9) by introducing first the notions of reachable and weak-terminable states in Definitions 6 and 7.

Definition 6 (*Reachable state*). A state *st* is reachable if exists a partial instance subgraph including *st* whose valuation of attributes satisfies all pre and postconditions of the visited services.

Definition 7 (*Weak-terminable state*). A state *st* is weak-terminable if exists a complete data instance subgraph including *st* whose valuation of attributes satisfies all pre and postconditions of the visited services.

Definition 8 (*Reachable artifact union graph*). An artifact union graph is reachable if every state is reachable.

Definition 9 (*Weak-terminable artifact union graph*). An artifact union graph is weak-terminable if every state is weak-terminable.

4. Automatic verification of artifact-centric business process models

In this section, the automatic verification of the reachability and weak-termination of artifact-centric business process models is detailed in a formal way. We propose the combination of Constraint Programming and made-to-measure algorithms to find the correctness of the global model following the definitions of the previous section. Algorithms for the verification of the reachability and weak-termination are introduced.

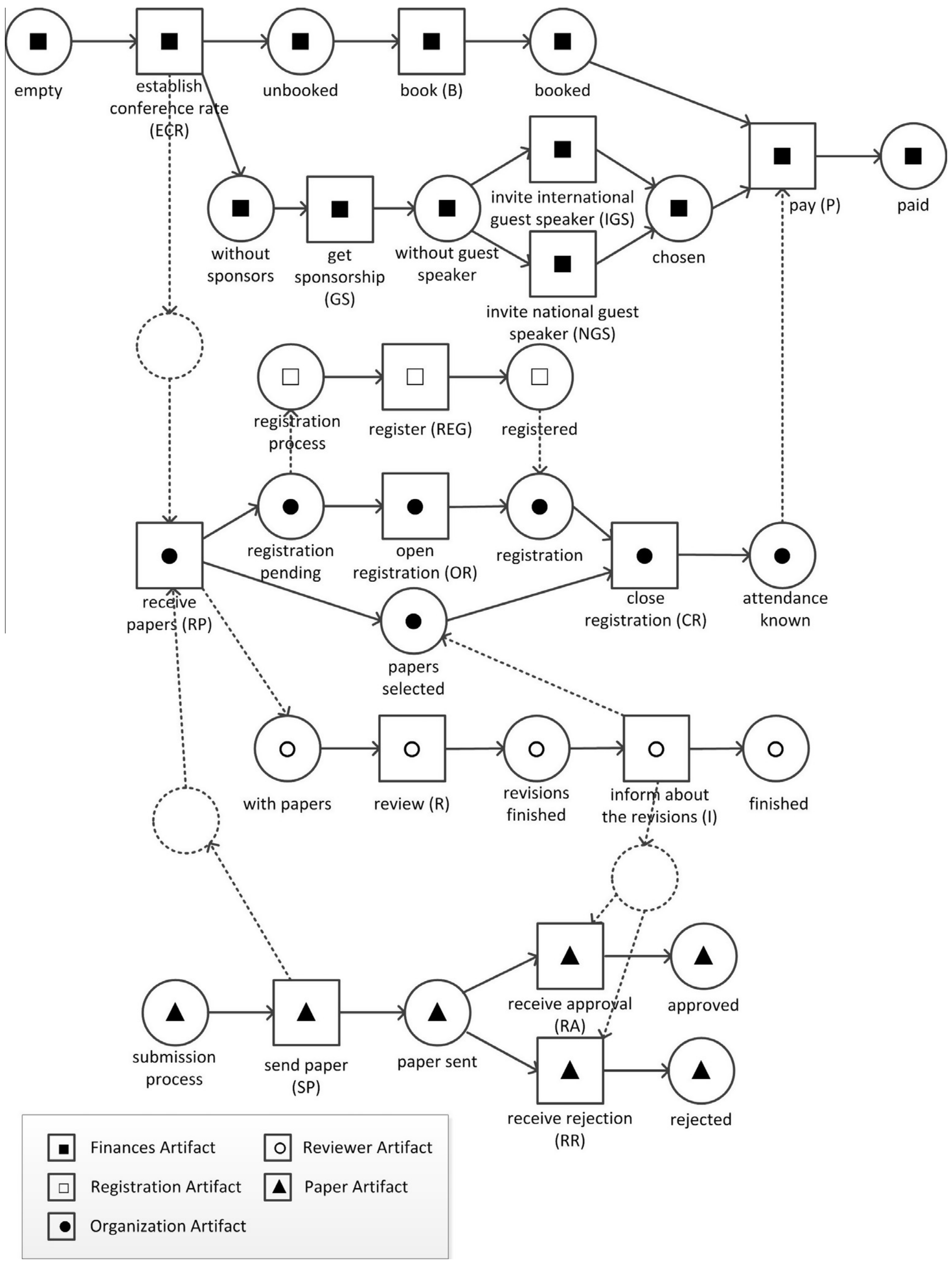

Fig. 3. Artifact union graph.

4.1. Modeling constraint satisfaction problems to verify the correctness

Constraint programming is based on the algorithmic resolution of Constraint Satisfaction Problems (CSP), and is an Artificial Intelligence technique which provides us a way to model the artifact union graph. A CSP [9] consists of the triple $\langle V, D, C\rangle$, where V is a set of n variables $v_1, v_2, \ldots, v_n$ whose values are taken from finite domains $D_{v1}, D_{v2}, \ldots, D_{vn}$ respectively, and C is a set of constraints on their values. The constraint $c_k\ (x_{k1}, \ldots, x_{kn})$ is a predicate that is defined on the Cartesian product $D_{k1} \times \cdots \times D_{kj}$. This predicate is true iff the value assignment of these variables satisfies the constraint c_k.

In order to model the problem as a CSP, we should distinguish between the modeling of the structure or topology of the artifact union graph, and the modeling of the data perspective including the data managed by the services in their pre and postconditions. First, we detail the structural formulation of the artifact union graphs, which is based on the formulation presented in [8], which was adapted from [11].

In the modeling of the structure of the artifact union graph, we need to model complete and partial instance subgraphs. To this end, a variable of the CSP is defined to represent each state, each service and each edge in the artifact union graph. The domain of these variables is $\{0 \ldots 1\}$, taking the variable the value 1 if the element (state, service or edge) is part of the instance subgraph, and 0 otherwise.

Definition 10 (*Structural CSP formulation*). For an artifact union graph $AUG = \langle G,\ Data,\ Inv,\ \Omega\rangle$, the **Structural CSP formulation** assigns the value 1 to the end node subject to the following constraints at each state and service in the artifact graph. For each state, service and edge x of AUG, so $x \in \{St \cup Ser \cup E\}$, a variable in the CSP v_x is created. The constraints are:

Table 2
Services with pre and postconditions.

Service	Precondition and postcondition
B	**post:** $regFee * 0.2 \leqslant (dinner + 3 * lunch) \wedge$ $(dinner + 3 * lunch) \leqslant 0.7 * regFee \wedge$ $regFee * 0.05 \leqslant others \wedge others \leqslant 0.25 * regFee$
IGS	**pre:** $sponsorship > 5000$ **post:** $guestSpeaker \geqslant 0.4 * sponsorship \wedge$ $guestSpeaker \leqslant sponsorship$
NGS	**pre:** $sponsorship \leqslant 5000$ **post:** $guestSpeaker \geqslant 0.2 * sponsorship \wedge$ $guestSpeaker \leqslant sponsorship$
P	**pre:** $\#Registration > 0 \wedge 3 * lunch + dinner + others < regFee$ **post:** $\#Registration * regFee + sponsorship \geqslant$ $\#Registration * (dinner + 3 * lunch + others) + guestSpeaker$
CR	**post:** $Card(selectedPapers) * 1.8 \geqslant \#Registration \wedge$ $Card(selectedPapers) * 0.5 \leqslant \#Registration$
R	**pre:** $Card(papersToReview) \geqslant 4$
RA	**post:** $approval == 1$
RR	**post:** $approval == 0$

C1 For $n \in St$, being $|inedge(n)| = k$: $\sum_{i=1}^{k} \mathtt{v\_inedge_i(n)} - \mathtt{v\_n} == 0$
C2 For $n \in St$, being $|outedge(n)| = k$: $\sum_{i=1}^{k} \mathtt{v\_outedge_i(n)} - \mathtt{v\_n} \leqslant 0$
C3 For $n \in Ser$, being $|inedge(n)| = k$: $\sum_{i=1}^{k} \mathtt{v\_inedge_i(n)} - k \cdot \mathtt{v\_n} == 0$
C4 For $n \in Ser$, being $|outedge(n)| = k$: $\sum_{i=1}^{k} \mathtt{v\_outedge_i(n)} \leqslant k \cdot \mathtt{v\_n}$

In detail, these constraints allow the transformation of the structural part of instance subgraphs as follows:

- The constraint **C1** allows a state n to be visited (i.e. $\mathtt{v\_n} = 1$) if one and only one incoming edge of n has been visited.
- Likewise, the constraint **C2** allows one and only one outgoing edge of the state n is visited if n has been visited too. To model both partial and complete instance subgraphs, this constraint permits $\sum_{i=1}^{k} \mathtt{v\_outedge(n)} - \mathtt{v\_n}$ to be less than or equal to zero in order to allow that a partial instance subgraph stops at n (i.e. $\sum_{i=1}^{k} \mathtt{v\_outedge(n)} == 0$ and $\mathtt{v\_n} == 1$).
- The constraint **C3** forces all the incoming edges of n to be visited to allow the visit of the service n.
- In addition, for **C4**, if a service n is visited, between 1 and $\mathtt{k}$ of its outgoing edges can be visited too. For the example in Fig. 3, we do not need the state *without sponsors* to be visited when we are checking the reachability of *unbooked*. Therefore, we allow only one outgoing edge of the service *establish conference rate* to be visited, so that we do not force *without sponsors* to be visited too.

C1, **C2**, **C3** and **C4** allow the modeling of the structural part of instance subgraphs. However, to obtain the complete model as a CSP, we need to include constraints to model the management of data in the pre and postconditions of the services activated in a particular instance subgraph. That is, the **Data CSP formulation**. To this end, the pre and postconditions of the visited services (that is, whose variable is equal to 1) should participate in the CSP, together with the constraints in *Inv* defining the invariants.

Regarding the translation of these constraints (*Inv*, *pre* and *post*) into constraints of the Data CSP formulation, some rules need to be considered:

- Each attribute *at* in the artifact union graph gives rise to a variable *v_at* within the CSP, whose domain is obtained from previous executions and/or knowledge from experts.
- The #*id* function included in the grammar, which represents the instances count for the artifact *id*, cannot be directly translated into a constraint with the same meaning, since in design time it is not possible to perform a counting of running instances. Therefore, this function is transformed into a variable representing the possible amount of instances, whose domain can also be obtained from previous executions and/or knowledge from experts. For instance, for the motivating example, the function #*Registration* that appears in the data policies and some pre and postconditions gives rise to the variable *registration_instances*, whose domain is {75...170} (collected from previous editions of the conference).
- The operators considered in *RELATIONAL_OP*, *ARITHMETIC_OP*, *BOOL_OP* and *SET_OP* in the aforementioned grammar can be directly translated into their equivalent CSP operators, available at any CSP solver.
- Regarding the operators in *AGGREGATION_OP*, they cannot be directly translated into CSP operators, since the runtime values of the variables are not available at design time. Therefore, they need to be adapted so that they can be useful for a verification based on the domains of the variables. In detail:
  - *Min*, *Max*, *Avg*: The operators $Min(x)$, $Max(x)$ and $Avg(x)$ are used in order to establish the lower bound, upper bound and average value for the attribute x respectively. To be included in the CSP, they must be translated into the variable *v_x*, to study during the propagation phase of the CSP that these values can be taking in any instance, implying the verification of the model. At design time, it is only possible to ensure that the corresponding domain includes values to satisfy them. For instance, the constraint $Min(regCost) \geqslant 250$ would be translated into $v\_regCost \geqslant 250$, so that the instantiated values of *regCost* always fulfill the established lower bound.
  - *Card*: The case of the *Card* operator is similar to the #*id* function, since in design time it is not possible to measure the cardinality (size) of sets. Therefore, analogously, the $Card(x)$ operator is transformed into a variable *x_card* representing the possible size of x, whose domain can be also obtained from previous executions and/or knowledge from experts.
  - Σ: The translation of constraints that include the summation operator depends on the parameter to sum, entailing two different translations:
    - ∗ In the case that the summation operator works on a simple attribute from different artifact instances, becoming a set, the summation is translated into the multiplication of the number of instances by the instantiated value of the corresponding variable. For instance, the expression Σ*Registration.regCost* is translated into *registration_instances* ∗ *v_regCost*.
    - ∗ In a similar way, in the case that the summation operator works on a multiple attribute (for instance, a list), the summation is translated into the multiplication of the cardinality of the multiple attribute by the value of the corresponding variable representing its content.

Following these rules, all the constraints in *Inv* are translated and included into the CSP formulation, and for each service n in the artifact union graph, after translating the $pre(n)$ and $post(n)$, the next constraint is also included:

C5 For $n \in Ser$: $\mathtt{v\_n} == 1 \rightarrow (\mathtt{pre(n)} \wedge \mathtt{post(n)})$

That is, if the service n is part of the instance subgraph, then its pre and postcondition should be satisfied. We need the conjunction stipulating that $\mathtt{n} == 1$ to ensure that the pre and postcondition are only enforced if n is activated, so n is in the instance subgraph.

Note that the pre and postcondition constraints hold for each service in a data instance subgraph. This way, the constraints can be easily encoded in a CSP model. However, this encoding complicates finding a solution to the CSP model, since the postcondition of a service might conflict with the postcondition of an earlier executed service, if both services reference the same variable in the CSP. For instance, an artifact union graph can contain two services *A* and *B* that both write integer variable *i*, where the postcondition of *A* is $i < 10$ and the postcondition of *B* is $i > 10$. A data instance subgraph containing *A* and *B* can assign only one value to *i*, so either the postcondition of *A* or of *B* is violated. Therefore the postconditions of *A* and *B* conflict. To resolve conflicts, the variables and constraints of the CSP model should be converted into Static Single Assignment (SSA) form, as it was detailed in [8].

4.2. Algorithm for reachability verification

To verify the reachability of the states in an artifact union graph, and provide proper feedback in case of incorrectness, we developed an algorithm (Fig. 4) that verifies whether each state is reachable (cf. Definition 6). For each state *st*, the algorithm tries to find a data instance subgraph that activates *st* and whose valuation satisfies the pre and postcondition of the services preceding *st*. The data instance subgraph that is searched for is a solution to the combined Structural and Data CSP formulation defined in Section 4.1 plus additional constraints encoding that *st* is activated and the service or services after *st* are not activated, so that the data instance subgraph stops at *st*.

The algorithm begins with the Structural CSP formulation of the artifact union graph (line 4 in Fig. 4), which states the structural constraints for data instance subgraphs. This Structural CSP formulation is combined with the Data CSP formulation of the artifact union graph (line 5), which states the pre and postconditions for the services in data instance subgraphs, together with the invariants from the policies. This combined Structural and Data CSP model is used in the sequel of the algorithm for every data instance subgraph. Next, the algorithm performs a loop to check if all the states are reachable (line 6). The state being processed in the loop is stored in the variable *current* (line 7). To test whether state *current* is reachable, the CSP model is extended with constraints that are satisfiable if the data instance subgraph activates *current* and stops at it (line 8). If no solution exists, there is no such data instance subgraph for *current*, so *current* is not reachable (line 10). If all states are reachable, the artifact union graph is reachable (line 17).

4.3. Algorithm for weak-termination verification

In order to check the weak-termination of an artifact union graph, we develop an algorithm (Fig. 5) that verifies whether each state in the artifact union graph is weak-terminable (cf. Definition 7). If a state *st* is not weak-terminable, the algorithm provides a counter example in the form of a data instance subgraph that activates *st*, reaches a goal state, and whose valuation violates some of the pre and postconditions of the activated services. If every state is weak-terminable, the artifact union graph is weak-terminable by definition.

First, the Structural and Data CSP formulation is created (line 4 and line 5) as defined in Section 4.1. As in algorithm Reachability-Verification, each data instance subgraph is a solution to this CSP model extended with additional constraints. Next, the algorithm performs a loop that processes each activity of the input artifact union graph (line 6). Variable *current* stores the state processed in the loop. The algorithm extends for *current* the CSP model with a constraint that states that the data instance subgraph activates *current* (line 9), and then, for each goal state in Ω, the CSP model is also extended to encode the reach of a goal state (line 11). If a solution to this extended CSP model exists (line 14) then there is a data instance subgraph that activates *current*, reaches a goal state, and whose valuation satisfies all pre and postconditions of the activated services. Therefore, *current* is weak-terminable (line 15). Otherwise, *current* is not weak-terminable (line 18), so that the variable *error* is set to *true* and the next state is processed. If every state is weak-terminable, the artifact union graph is weak-terminable (line 25).

4.4. Using the verification algorithms for the analysis of the motivating example

Once an artifact-centric business process model is formalized using artifact union graphs, it can be analyzed with our proposed methodology. This section presents the results of applying the algorithms to verify reachability and weak-termination of the motivating example in Fig. 3. This artifact union graph has no missing and no conflicting data.

As mentioned in Section 4.1, the variables in the Data CSP formulation derive from the attributes and instances count that appears in the policies, pre and postconditions. Table 3 shows those variables with their corresponding type and domain. For the variables of type *List of Integer*, the established domain refers to the elements contained in the list. It is possible to notice the declaration of the variables *paper_instances* and *registration_instances*, due to the functions #*Paper* and #*Registration* that appear in the data policies and the pre and postconditions of some services respectively.

First, the artifact-centric business process model is verified for reachability by applying the algorithm Reachability-Verification. The algorithm determines that the artifact union graph is reachable, since for each state *st*, it is always possible to find at least one data instance subgraph that visits *st*. The algorithm also includes that the valuation of the variables is within the determined finite domains (listed in Table 3), such that the pre and postcondition of the services preceding *st* are satisfied.

Then, we verify the weak-termination by applying the algorithm presented in Fig. 5,, to the artifact union graph in Fig. 3. The algorithm finds no error, so that every state *st* is weak-terminable. That is, it is possible to find at least one data instance subgraph that visits *st* and reach a goal state, with a valuation of the variables satisfying the pre and postconditions of the visited services.

However, some changes in the motivating example can lead us to a no reachable and/or weak-terminable model. For instance, if the domains of the variables *regFee* and *lunch* are changed to $\{200\ldots250\}$ and $\{25\ldots30\}$ respectively, and the postcondition of

```
1:  procedure REACHABILITY-VERIFICATION($G, Data, Inv, \Omega$)
2:      $error = false$
3:      $unmarked = G.St$
4:      $structCSP$ = make Structural CSP formulation for $(G.St, G.Ser, G.E)$
5:      $CSP = structCSP$ + Data CSP formulation for $(Data, Inv)$
6:      while $unmarked \neq \varnothing \wedge error = false$ do
7:          $current$ = a state from $unmarked$
8:          $CSP' = CSP \;\&\&\; current = 1 \;\&\&\; \sum_{i=1}^{k} v\_outedge(current) = 0$
9:          $sol$ = solve $CSP'$
10:         if $sol$ is null then // $CSP'$ is unsatisfiable
11:             Print "State $current$ is not reachable"
12:             $error = true$
13:         end if
14:         $unmarked = unmarked \setminus \{\ current\ \}$
15:     end while
16:     if $error = false$ then
17:         Print "The artifact union graph is reachable"
18:     end if
19: end procedure
```

Fig. 4. Algorithm for reachability verification.

```
procedure WEAK-TERMINATION-VERIFICATION($G, Data, Inv, \Omega$)
    $error = false$
    $unmarked = G.St$
    $structCSP$ = make Structural CSP formulation for ($G.St, G.Ser, G.E$)
    $CSP = structCSP$ + Data CSP formulation for ($Data, Inv$)
    while $unmarked \neq \emptyset$ do
        $current$ = a state from $unmarked$
        for each goal state $gs$ in $\Omega$ do
            $CSP' = CSP$ && $current = 1$
            for each state $st$ in $gs$ do
                $CSP' = CSP'$ && $st = 1$
            end for
            $sol$ = solve $CSP'$
            if $sol$ is not null then // $CSP'$ is satisfiable
                Print "State $current$ is weak-terminable"
                break
            else
                Print "State $current$ is not weak-terminable"
                $error = true$
            end if
        end for
        $unmarked = unmarked \setminus \{ current \}$
    end while
    if $error = false$ then
        Print "The artifact union graph is weak-terminable"
    end if
end procedure
```

Fig. 5. Algorithm for weak-termination verification.

the service *book* is set to $\{regFee * 0.2 \leqslant (dinner + 3 * lunch) \wedge (dinner + 3 * lunch) \leqslant 0.4 * regFee \wedge regFee * 0.05 \leqslant others \wedge others \leqslant 0.25 * regFee\}$, the artifact union graph is not reachable (since the state *booked* is not reachable), and either is weak-terminable (due to the state *unbooked* is not weak-terminable). The cause is that is not possible to find a valuation of variables so that the postcondition of the service *book* is satisfied.[2]

4.5. Tractability

Regarding the tractability of our approach, since the verification of artifact-centric business process models has been mapped onto a CSP, the verification time is linked to the complexity of the resolution of the CSP, as it was discussed in [12]. This has been analyzed in great depth over recent decades [13], and depends on two parameters: the width of the graph and the order parameter. On one hand, the width of the graph represents the relation between the constraints, where the tractability in CSPs is due to the structure of the constraint network, where the tree-structured CSPs have polynomial complexity (linear with respect to the number of variables, and quadratic with respect to the cardinal of the domain of the variables). On the other hand, the order parameter, defined as the ratio of the number of forbidden valuations of variables to the total number of possible combinations, determines the partition of the problems space into under-constrained, over-constrained and just-constrained problems. In the first two cases, the problems are scalable, but in just-constrained problems, a significant increase of solving cost could occur and the scalability would not be possible [14].

For these reasons, no affirmation about the efficiency or scalability in a generic way can be given by our proposal, since our approach permits any number of constraints defined with numerical variables, and therefore the evaluation time will depend on the specific problem.

In our approach, for both presented algorithms, the execution time depends on the CSP characteristics: (i) it only requires finding a solution; (ii) the constraints are limited by the aforementioned grammar, where the only possible polynomial constraints are the included by means of the preconditions, postconditions and data policies, since the CSP structure only contains linear constraints [15,9]; and (iii) the short number of handled variables. Therefore, as mentioned, the scalability of our approach could be affected by a large increase in the number of constraints and/or variables wrt the number of states. However, this is not usual in real life artifact-centric business processes [16]. Furthermore, owing to the search methods used by CSP solvers, and to the constraints limited by the grammar previously mentioned, the increase in the number of constraints and/or variables could not affect the execution time. Specifically, the default method used by CSP solver is dynamic and based on the first-fail principle [17,18], which orders the variables by increasing set cardinality, breaking ties by choosing the variable with the smallest domain size, and reducing the average depth of branches in the search tree. Therefore, in theory, the time it would take to verify even large artifact-centric business process models with the presented algorithms is expected to be acceptable.

As a concrete example, the verification of the reachability of the motivating example[3] takes less than 4 s.[4] Since both algorithms for checking reachability and weak-termination have a similar structure (a while-loop which processes every state by solving a CSP model for the state) they are equally tractable.

Table 3
Variables and domains in the Data CSP formulation of the example.

Variable	Type	Domain
regFee	Integer	{200…390}
sponsorship	Integer	{0…15,000}
dinner	Integer	{60…100}
lunch	Integer	{10…30}
others	Integer	{30…185}
guestSpeaker	Integer	{0…10,000}
selectedPapers	List of integer	{1…150}
paper_instances	Integer	{50…150}
oid	Integer	{1…150}
approval	Integer	{0…1}
papersToReview	List of integer	{1…150}
registration_instances	Integer	{75…170}
regCost	Integer	{200…390}

5. Related work

Most of the previous works in the literature do not take into account numerical data verification in the artifact oriented model, or they study the data-aware verification in the activity-centric business process model. In the following paragraphs we analyze these two types of approaches.

Regarding the verification of activity-centric business process models, citation [19] considers verification of semantic business processes, in which activities are annotated with pre and postconditions. They detect conflicts between preconditions and postconditions of parallel activities and next study the reachability and executability of the activities, but only if the activities are conflict free. In their contribution, pre and postconditions are considered as CNF formulas with only boolean variables. Therefore, the approach in [19] cannot diagnose the correctness of workflows whose activities count on pre and postconditions involving other kinds of data. This is performed in contribution [8], where an approach to verify workflow models that integrates both process and numerical data

[2] In order to show the steps for the verification of the example, a video of our tool is available at: http://estigia.lsi.us.es/diana/Artifact-centric_BP_model_verifier.mp4.

[3] The XML file containing the motivating example detailed in this paper is available at: http://estigia.lsi.us.es/diana/ExampleConference.xml.

[4] The test case is measured using a Windows 7 machine, with an Intel Core I7 processor, 3.4 GHz and 8.0 GB RAM.

Table 4
Related works regarding verification at design time.

Source	Closed problem	Problem approach	Solving method
[21]	Static verification of properties	Rules in LTL. Declarative specification	Logic and model checking
[22]	Verification of feedback-free business artifacts	Temporal properties LTL-FO	Specific decision procedure for satisfaction of LTL-FO properties
[23]	Verification of properties of models	GSM models	Simbolic model checking. GSM Checker
[24]	Verification of GSM models	Finite abstractions	Model checking
[25]	Generate complaint and operational process model. Diagnosis of non-compliant BP model	Policies and compliance rules. Petri Nets	Tool Wendy and the Petri net synthesis tool Petrify
[26]	Conformance between process models and data objects	Definition of weak conformance	Conformance checking
[31]	Verification of artifact behaviors	Artifact Behavior Specification Language (ABSL)	Computational Tree Logic

verification is presented. In [12], a runtime compliance and diagnosis of data is proposed. The compliance and diagnosis are performed according to the constrains that represent the possible correct values, but supposing that the model is correct.

On the other hand, as regards the verification of artifact-centric business process models, contribution [20] performs a formal analysis of artifact-centric business processes by identifying certain properties and verifying their fulfillment, such as persistence (once an artifact is created, does it persist or can it disappear?) and uniqueness (can an artifact appear in more than one places at once?).

Citation [21] provides a static verification of whether all runs of an artifact system satisfy desirable correctness properties, expressed as rules in linear-time temporal logic. The services are specified in a declarative manner, including their pre and postconditions. However, they fail in the presence of even very simple data dependencies or arithmetic, both crucial to real-life business processes. This problem is addressed and solved in [22], where data dependencies (integrity constraints on the database) and arithmetic operations performed by services are considered.

To verify the behavior of an artifact system, contribution [23] transforms the GSM model into a finite-state machine and systematically examines all possible behaviors of the new model against specifications. It presents a novel methodology to verify the behavior of artifacts in terms of their possible sequences of B-steps. Likewise, the approach in [24] observes two deficiencies in the GSM approach, and resolves them. They also observe that GSM programs generate infinite models, so that they isolate a large class of amenable systems, which admit finite abstractions and are therefore verifiable through model checking.

The field of compliance for artifact-centric processes has been addressed in [25]. They extend the artifact-centric framework by including the modeling of compliance rules, and obtain a model that is compliance by design. This way, the runtime verification of compliance is not required.

The contribution [26] checks for conformance between process models and data objects at design time. They propose a notion of weak conformance, which is used to verify that the correct execution of a process model corresponds to a correct evolution of states of the data objects.

These most relevant contributions in verification of artifact-centric business process models at design time are collected in Table 4.

Finally, regarding the modeling of the approach, several works in the literature use Petri-net-based models [27–30]. Petri nets are mainly aimed to control problems about the structural perspective of the processes (such as concurrency problems or deadlocks), resource availability, or loss of data. Therefore, the role played by the transitions is very important. In the case of the verification of artifact-centric process models presented in this paper: (1) data is never lost, (2) the resources assignment is not taken into account, (3) the errors in the structural perspective are not considered, and thus (4) transitions are not relevant. Consequently, a more general modeling has been developed, based on graph theory, which is aware of the proper characteristics of the models under study: pre and postconditions in the services, structural and data policies, and different cardinalities between artifacts.

In short, most of the previous approaches in the literature perform verification of artifact-centric business process models through model checking or the static confirmation of whether all runs of an artifact system satisfy desirable properties expressed in (an extension of) linear-time temporal logic. On the contrary, the approach in this paper exhaustively verifies the correctness (reachability and weak-termination) of all states in an artifact-centric system at design time, without needing to define any extra specification. Likewise, the models considered in the approach in the literature either do not define any kind of pre and postconditions of the services or the pre and postconditions and only defined through data objects or by existential first-order sentences. To the best of our knowledge, this paper presents the first verification approach for artifact-centric business process models at design time that integrates pre and postconditions defining the behavior of the services and numerical data verification when the model is formed by more than one artifact, handling 1-to-N and N-to-M associations between artifacts. The approach can detect errors not detectable with other approaches.

6. Conclusions

To ensure the correctness of artifact-centric business process models, their verification is of utmost importance in order to avoid errors at latter phases of the business process management (i.e. at runtime). To that end, we have proposed artifact union graphs as a formalization of artifact-centric business process models together with two correctness notions, reachability and weak-termination, that can be verified for artifact union graphs. Artifact union graphs model artifact-centric business process models with more than one artifact, handling 1-to-N and N-to-M associations between artifacts. This analysis includes structural and data dependencies, with pre and postconditions defining the behavior of the services, and considering the valuations of the managed numerical data.

Our proposal consists of various phases: (a) preprocessing is applied to detect basic data anomalies; (b) the artifact union graph is translated into a CSP formulation in order to automatize the verification, avoiding its manual performance which is time-consuming and error-prone; (c) the CSP formulation models the lifecycle of the artifacts with pre and postconditions of the services and the numerical data managed, analyzing the possible interaction among the different artifacts; and (d) in case of an error, feedback is provided by determining the states which are nor reachable and/or weak-terminable. The approach is complete, so it always generates accurate feedback in case of an error.

As future work, we plan to offer additional feedback in case of a violation, making easier the job of fixing the problem causing the error. Likewise, we would also like to extend the verification by checking the reachability of the services. This way, services which can never be executed would be detected, avoiding the existing of dead services in the lifecycle.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Science and Education (under Grant TIN2009-13714), and by FEDER (under the ERDF Program).

References

[1] M. Weske, Business Process Management: Concepts, Languages, Architectures, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[2] A. Nigam, N.S. Caswell, Business artifacts: an approach to operational specification, IBM Syst. J. 42 (2003) 428–445.

[3] D. Cohn, R. Hull, Business artifacts: a data-centric approach to modeling business operations and processes, IEEE Data Eng. Bull. 32 (2009) 3–9.

[4] R. Eshuis, R. Hull, Y. Sun, R. Vaculín, Splitting gsm schemas: a framework for outsourcing of declarative artifact systems, in: Business Process Management (BPM 2013), Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2013, pp. 259–274.

[5] OMG, Object Management Group, Business Process Model and Notation (BPMN) Version 2.0, OMG Standard, 2011.

[6] A. Meyer, L. Pufahl, D. Fahland, M. Weske, Modeling and enacting complex data dependencies in business processes, in: Proceedings of the 11th International Conference on Business Process Management, Springer-Verlag, 2013, pp. 171–186.

[7] M.T. Gómez-López, D. Borrego, R.M. Gasca, Data state description for the migration to activity-centric business process model maintaining legacy databases, in: Business Information Systems, Springer, 2014, pp. 86–97.

[8] D. Borrego, R. Eshuis, M.T. Gómez-López, R.M. Gasca, Diagnosing correctness of semantic workflow models, Data Knowl. Eng. 87 (2013) 167–184.

[9] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006.

[10] R. Hull, Artifact-centric business process models: brief survey of research results and challenges, in: Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part II On the Move to Meaningful Internet Systems, OTM '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 1152–1163.

[11] R. Eshuis, A. Kumar, An integer programming based approach for verification and diagnosis of workflows, Data Knowl. Eng. 69 (2010) 816–835.

[12] M.T. Gómez-López, R.M. Gasca, J.M. Pérez-Álvarez, Compliance validation and diagnosis of business data constraints in business processes at runtime, Inf. Syst. 48 (2015) 26–43.

[13] P. Cheeseman, B. Kanefsky, W.M. Taylor, Where the really hard problems are, in: Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI'91, vol. 1, Morgan Kaufman Publishers Inc.,, San Francisco, CA, USA, 1991, pp. 331–337.

[14] K. Apt, Principles of Constraint Programming, Cambridge University Press, New York, NY, USA, 2003.

[15] A. Bulatov, P. Jeavons, A. Krokhin, Classifying the complexity of constraints using finite algebras, SIAM J. Comput. 34 (2005) 720–742.

[16] M. Chinosi, A. Trombetta, Bpmn: an introduction to the standard, Comput. Stand. Interf. 34 (2012) 124–134.

[17] R.M. Haralick, G.L. Elliott, Increasing tree search efficiency for constraint satisfaction problems, in: Proceedings of the 6th International Joint Conference on Artificial Intelligence, IJCAI'79, 1, Morgan Kaufman Publishers Inc., San Francisco, CA, USA, 1979, pp. 356–364.

[18] P. Van Hentenryck, Constraint Satisfaction in Logic Programming, MIT Press, 1989.

[19] I. Weber, J. Hoffmann, J. Mendling, Beyond soundness: on the verification of semantic business process models, Distrib. Parallel Databases 27 (2010) 271–343.

[20] C.E. Gerede, K. Bhattacharya, J. Su, Static analysis of business artifact-centric operational models, in: Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications, SOCA '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 133–140.

[21] A. Deutsch, R. Hull, F. Patrizi, V. Vianu, Automatic verification of data-centric business processes, in: Proceedings of the 12th International Conference on Database Theory, ICDT '09, ACM, New York, NY, USA, 2009, pp. 252–267.

[22] E. Damaggio, A. Deutsch, V. Vianu, Artifact systems with data dependencies and arithmetic, ACM Trans. Database Syst. 37 (2012) 1–36.

[23] P. Gonzalez, A. Griesmayer, A. Lomuscio, Verifying gsm-based business artifacts, in: ICWS, IEEE, 2012, pp. 25–32.

[24] F. Belardinelli, A. Lomuscio, F. Patrizi, Verification of gsm-based artifact-centric systems through finite abstraction, in: Proceedings of the 10th International Conference on Service-Oriented Computing, ICSOC'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 17–31.

[25] N. Lohmann, Compliance by design for artifact-centric business processes, Inform. Syst. 38 (2013) 606–618.

[26] A. Meyer, A. Polyvyanyy, M. Weske, Weak conformance of process models with respect to data objects, in: Central-European Workshop on Services and their Composition (ZEUS), 2012, pp. 74–80.

[27] W.M.P. van der Aalst, The application of petri nets to workflow management, J. Circ. Syst. Comput. 8 (1998) 21–66.

[28] W.M.P. van der Aalst, Workflow verification: finding control-flow errors using petri-net-based techniques, in: Business Process Management, Models, Techniques, and Empirical Studies, Springer-Verlag, London, UK, 2000, pp. 161–183.

[29] A. Awad, G. Decker, N. Lohmann, Diagnosing and repairing data anomalies in process models, in: Business Process Management Workshops, Lecture Notes in Business Information Processing, vol. 43, Springer, 2009, pp. 5–16.

[30] D. Fahland, M. Leoni, B. Dongen, W. Aalst, Conformance Checking of Interacting Processes with Overlapping Instances, in: Business Process Management (BPM 2011), Lecture Notes in Computer Science, vol. 6896, Springer-Verlag, Berlin, 2011, pp. 345–361.

[31] C.E. Gerede, J. Su, Specification and verification of artifact behaviors in business process models, in: International Conference on Service Oriented Computing, ICSOC'07, 2007, pp. 181–192.