Knowledge and Information Systems (2020) 62:2653–2683
https://doi.org/10.1007/s10115-019-01431-5

REGULAR PAPER

Verifying the manipulation of data objects according to business process and data models

José Miguel Pérez-Álvarez[1] · María Teresa Gómez-López[1] · Rik Eshuis[2] · Marco Montali[3] · Rafael M. Gasca[1]

Received: 26 July 2017 / Revised: 4 December 2019 / Accepted: 8 December 2019 /
Published online: 3 January 2020

Abstract
Business processes read and write data objects, usually stored in databases. Although data models and activity-oriented business process models originate from different paradigms, they need to work together properly. The data object states are transformed during each process instance by the activities of the process model. It is therefore necessary to verify whether the states of the data objects are correct according to the process model, and to discover the states of the stored data objects. This implies determining the relation between the data objects stored in the database, the data objects involved in the process, and the activities that within the business process that create the data objects and modify their states. In order to verify the business process annotated with data states and to reduce the existing gap between data model and business process model, we propose a methodology that includes enlarging the capability to describe data states in business processes; verifying the completeness and consistency of the data states described in accordance with their relation to the business process model; and discovering the states of the data objects stored in the database according to the business process model where they are managed. The methodology is supported by a framework that enables a natural-like language to be employed to describe the states, to apply the necessary algorithms to verify the consistency and completeness of the model, and to determine the states of the stored data objects according to the model described. To validate our proposal, an extension of Activiti^{TM} has been implemented and applied to a real example as an illustration of its applicability.

Keywords Business processes · Integration of data and processes · Data object state · Object-relational mapping · Data state verification

1 Introduction

Business processes (BPs) and their continuous improvement are fundamental to the operation of companies. For a wide range of enterprises, business process analytics, validation, and verification are a key endeavour [1]. The automation of processes offers an opportunity to

✉ José Miguel Pérez-Álvarez
josemi@us.es

Extended author information available on the last page of the article

gain visibility and control over both the execution of processes and the analysis of their underlying data. Process data in general are stored and manipulated through various systems, applications, and services, and sometimes shared among several processes and information systems. In this light, it becomes crucial to simultaneously tackle both the business processes and their related data [2]. This gives rise to integrated models for processes and data which in turn opens up the possibility for their analysis and verification [3,4]. Notably, these integrated models can also be distributed in Big Data environments [5].

Intensive research on data-aware processes has brought forward non-conventional, data- and artefact-centric models of business processes [6]. At the same time, in spite of the lack of data awareness of conventional, activity-centric process notations such as BPMN [7], conventional business process management systems (BPMS) all support the enrichment of such notations with various types of volatile and persistent data. In fact, when the complexity and quantity of such data are high, contemporary organisations tend to incorporate a commercial BPMS to support the operation of processes and the management of their corresponding data. The benefits of employing a BPMS are several [8], and include: (1) the possibility of choreographing various processes a variety of technologies; (2) the elicitation of domain knowledge using graphical languages; (3) the support by a business modeller to reduce overheads, resulting in less time and/or fewer resources being used to achieve an end result; (4) the possibility of more effectively detecting process inconsistencies, and handling their optimisation; and (5) the possibility to use several tools and technologies that enable business intelligence on top of the process data.

Virtually every contemporary BPMS provides a way to consistently store the data produced and manipulated during the execution of processes, typically relying on relational databases. In this context, a database can be understood as a repository of business data objects that might be created and modified during process executions. A number of the stored data objects represent the values involved in past and current business process executions, which had been included in the database previous to the incorporation of the BPMS. The values of the stored data represent the states of the business objects obtained from the process execution, regardless of whether this process was performed by means of the activities of a BPMS, third-party applications, or direct human updates performed through the information system. Consequently, there are two main challenges arising from the necessity of combining data objects and business processes. On the one hand, data models and activity-centric business process models are usually modelled separately, or integrated within contemporary BPMS using ad hoc mechanisms [9]. On the other hand, the BPMS may use data that is not created and manipulated by its enacted processes, but that was created before and exists independently from the BPMS itself [10]. This brings about the necessity to explicitly incorporate data objects and their states into the business process model, thereby reducing the existing gap between the states of the stored business data objects, and how they are modelled, accessed, and modified in the process model. The capacity to annotate the business processes with the states of the data object was included in BPMN 2.0 [11], but a substantial linkage is required between the data and process dimensions going far beyond that which is supported by the standard. A conceptual alignment between persistently stored data objects and business processes can also help detect deviations between the expected and realised behaviours, and to understand whether the enacted processes are aligned with the intended processes.

In this work, we tackle the problem of verifying the overall correctness of data objects and their lifecycle, according to how process models evolve the states of such business objects. To verify the correct evolution of the data objects through the workflow, a (DSL) domain-specific language has been defined to describe the data states in various parts of the process. The consistency and completeness of the described data states and the business process model

are analysed to ascertain whether the possible evolution of a data object during an instance can create or update an object erroneously.

In order to support the business process model annotated with data objects and its automatic verification, our proposal provides a set of definitions of properties related to consistency and completeness in the context. Furthermore, a methodology has been defined, supported by a framework that defines the steps to model the annotated business process, incorporate the data stored from a legacy database, and to ensure the correctness of the model (i.e. data model and process model). The framework incorporates a constraint programming solver to evaluate the consistency and completeness of the model that is being integrated into a commercial BPMS tool.

In order to facilitate the understanding of our proposal, the rest of the paper is organised as follows: Sect. 2 introduces the definitions regarding process models and data states; Sect. 3 introduces the proposed definitions regarding consistency, completeness, and data object correctness; Sect. 4 depicts the algorithms needed to verify the data states in the process model and the validation of the stored object in a relational repository; Sect. 5 explains the methodology and the proposed framework for the verification of the correctness according to the presented definitions, and the details about a domain-specific language (DSL) that facilitates the description of the data states; Sect. 6 analyses an overview of related work found in the literature; Sect. 7 marks the limitations and the scope of our proposal; and finally, conclusions are drawn and future work is depicted in Sect. 8.

2 Data state model and verification

The understanding of the relationships existing between business processes and business data remains deficient [3], since these two models are disconnected and described using different languages and software tools. Since these two perspectives are not well-integrated in the existing process management systems, the verification of their combination remains a challenge. The necessity to combine data perspective and business process workflow during a verification process is not a new issue and has been analysed by Sun et al. [12].

This decoupling necessitates verification of the correctness of the combinations of the two models, because the problem of data verification cannot be reduced to the simple analysis how single and isolated values traverse a workflow that the existing isolation between stored data and the business process models that manage them is a recognised problem in the research community [4]. Significant proposals have been published in recent years on this issue, including those involving the use of data as the centre of the process model, known as artefact-centric business process models [13–15]. However the problem to be addressed is that activities and data have to coexist in the same business process, while sharing and using information produced in other activities, or even in other processes [16], and the data states must be consistent with the workflow process that manages them.

Various modelling paradigms are combined: *Business Process Modelling*, *Conceptual Data Model*, and *Data Object Life cycle Modelling*. In order to understand the difficulty in verifying the correctness between business processes annotated with the states of the data objects combined together, the following subsections formalise the different parts of the problem using a motivating example. The example given to clarify the proposal is related to the paperwork of the presentation of a thesis described in the current regulations of the University of Seville as published in [17]. The university uses made-to-measure software that stores the information about students in a relational database, but no description is provided

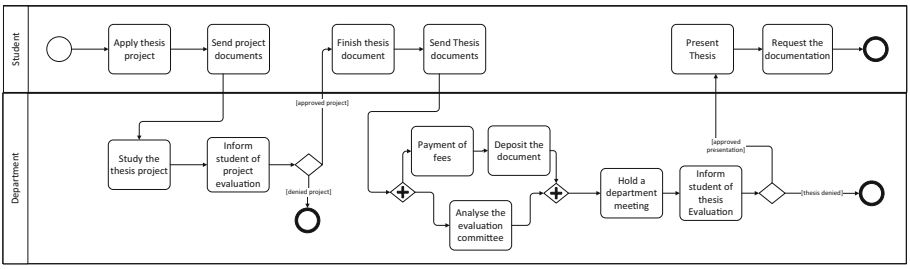

Fig. 1 Thesis paperwork process example

regarding how the object evolves during the process and the relation with the state of thesis development. Since different paradigms must be combined, various formal languages are analysed in this paper.

2.1 Business process modelling

BPMN 2.0 [11] is a standard notation that provides an understandable language for all business users (i.e. business analysts, technical developers, and business people who manage and monitor those processes). Thus, BPMN creates a standardised bridge for the gap between the design and implementation of the business process.

The process described using BPMN, shown in Fig. 1, is a simplification of the real process regarding the presentation of a thesis, but the example includes all the components necessary for the difficulties to be understood. The business process model example includes the activities developed by *Students* and *Departments*. The process begins with the application of a thesis project by a *Student*, after which, the project documents are sent to the *Department*, which studies the documentation, and informs the *Student* about the evaluation. If the thesis project is rejected, then the *Student* should start the process again from the beginning, otherwise the *Student* should finish the documents and send them to the *Department*. In a parallel way, the evaluation committee evaluates the documentation, while the fees are paid and the deposit is completed. After having completed both activities, a department meeting is held to evaluate all the theses which have been prepared in the department since the last department meeting. The Student is then informed about the result of the evaluation and, if the presentation of the thesis is approved, the Student may present the thesis. In order to complete the process, the Student formalises the necessary documentation to obtain the Ph.D. title.

In order to facilitate explanations regarding data states and model verifications, we use the definition of a process model in BPMN 2.0 as a process graph. This facilitates the explanation and development of how to traverse business processes and the data states associated with these processes in order to verify the model correctness. The construction of the process graph is based on the annotated graph presented in [18], and includes certain differences as explained in [19], and is enlarged with the data states associated as input and/or output of each activity node.

Definition 1 (Process Graph) A process graph is a labelled directed graph $G = \langle N, E \rangle$, composed of nodes (N) and edges (E). N is the disjoint union of $\{n_0\}$ (start node), N_+ (end nodes), N_T (task nodes), N_{PS} (parallel splits), N_{PJ} (parallel joins), N_{ORS} (or splits), N_{ORJ}

(or joins), N_{XS} (xor splits), and N_{XJ} (xor joins). For $n \in N$, IN(n)/OUT(n) denotes the set of incoming and outgoing edges of n, respectively.

Each task node n can have an associated data state that represents the states of the data object before and after this task is executed, $S(n)_{IN}$ and $S(n)_{OUT}$ (States IN and States OUT).

In order to determine that a business process workflow model described by a process graph is correct, it is required that:

1. For each split node n, $|\text{IN}(n)| = 1$ and $|\text{OUT}(n)| > 1$;
2. For each join node n, $|\text{IN}(n)| > 1$ and $|\text{OUT}(n)| = 1$;
3. For each $n \in N_T$, $|\text{IN}(n)| = 1$ and $|\text{OUT}(n)| = 1$;
4. For n_0, $|\text{IN}(n)| = 0$ and $|\text{OUT}(n)| = 1$, and vice versa for $n \in N_+$;
5. Each node $n \in N$ is on a path from the start to an end node;
6. If $|\text{IN}(n)| = 1$, then IN(n) is identified with its single element, and similarly for OUT(n);
7. Every split is closed (joined) in a join or an end node;
8. The outgoing edges of $n \in \{N_{XS} \cup N_{ORS}\}$ have to be labelled with a condition to describe when this branch is executed;
9. One and only one of the labels for the outgoing edges of a node $\in N_{XS}$ can be labelled as *default*, or none of them;
10. For each $n \in N_T$, $|S(n)_{IN}| \geq 0$ and $|S(n)_{OUT}| \geq 0$;
11. For each n_j and $n_{j+1} \in N_T$, if $|\text{OUT}(n_j))| = 1$ and $|\text{IN}(n_{j+1})| = 1$ then $S(n_j)_{OUT} = S(n_{j+1})_{IN}$;

2.2 Conceptual data model

Various models can be used for the description of data models. Since relational models contain several details about data types and the primary and foreign key relations, we propose the use of conceptual modelling [20] using unified modelling language (UML) [21] (depicted in Fig. 2) to enclose the implementation details. The conceptual model presented in the example describes information regarding the thesis projects, theses, students, Ph.D.s (i.e. supervisors of each thesis, the members of the evaluation committee for each thesis, the substitution committee for the presentation), and the meetings held by the department. The thesis project of a student is the first part of a possible thesis, which is approved for its presentation in a department meeting. Both thesis and thesis project are evaluated and supervised by Ph.D. personnel of the University.

2.3 Data object life cycle modelling

BPMN was not originally designed for data modelling, although BPMN 2.0 does include elements that enable the annotation of the process with the involved *Data Object* elements. The *Data Objects* can include the State of Data Objects at various points in a process [11]. An activity in a process is able to *Consume* a Data Object, which implies that the activity might be executed when the Data Object is in a particular state (working as a pre-condition). When the activity is executed, the object might transit to a new state (an object in a new state is *Produced*), and thereby work as a post-condition. Figure 3 represents an example of accessing the data objects related to *Activity A*. An edge from a data object to an activity describes a read access to an instance of the data object (e.g. Data object *X*), when the state of the pre-condition is fulfilled. Likewise, an edge from an activity to a data object describes a write access (e.g. Data object *W*), which either creates a data object instance if it does not

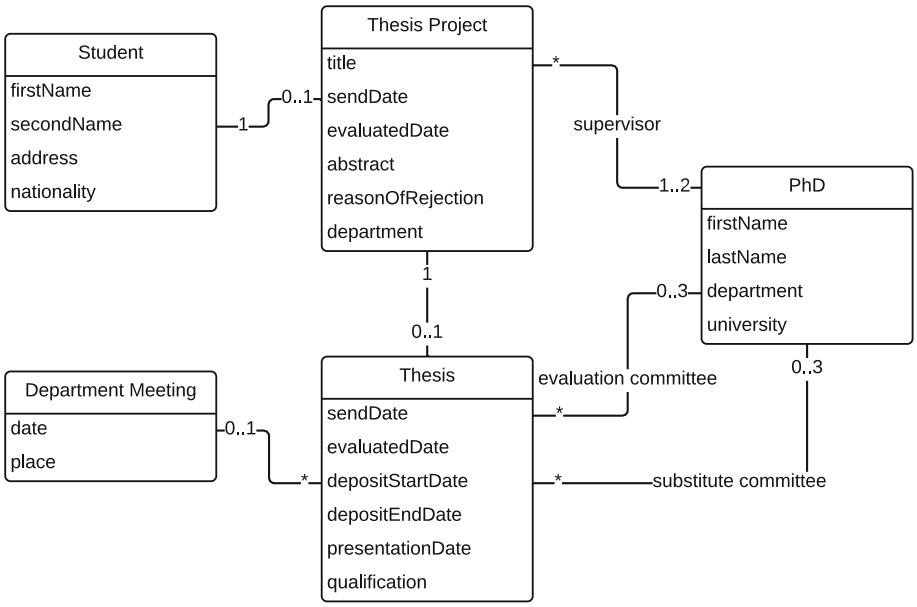

Fig. 2 Conceptual model of the example

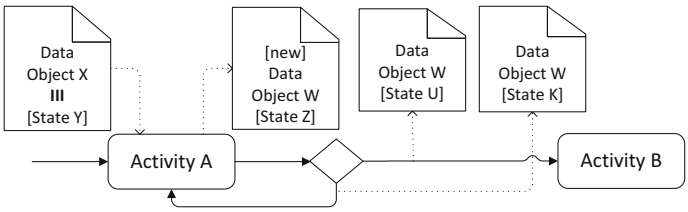

Fig. 3 Example of annotation of business process model

already exist (labelled with [new] as proposed in [22]), or updates the instance if it already exists. A data flow edge connecting a data object with a sequence flow indicates that the data object is flowing through that connection, and gives the state of the flowing data object (e.g. Data Object *W* in State *U* in Fig. 3). Data objects can be modelled as a single object or as a collection of objects (marked by three parallel bars, |||). Only the execution of an activity can imply the creation of a new object, although it is possible to represent different states of an object depending on the executed branch. For example, after the execution of the activity *A*, an XOR-split is executed, where the data object *W* in state *U* flows through the upper branch, or it follows through the lower branch in state *K*.

In order to include the evolution of data states during the process execution, it is necessary to annotate the BPMN model with the object data states, which are supported by BPMN 2.0. The process annotated with the data objects (*Thesis* and *Thesis Project*) and their states is shown in Fig. 4.

In order to verify the annotated BP Model and the conceptual model, it is necessary to enrich the description capacity of the data object states, and not only use a label that determines the state as analysed in [23]. Each class belonging to the conceptual model is formed of a set of attributes, and a set of allowed values for each attribute. The states are

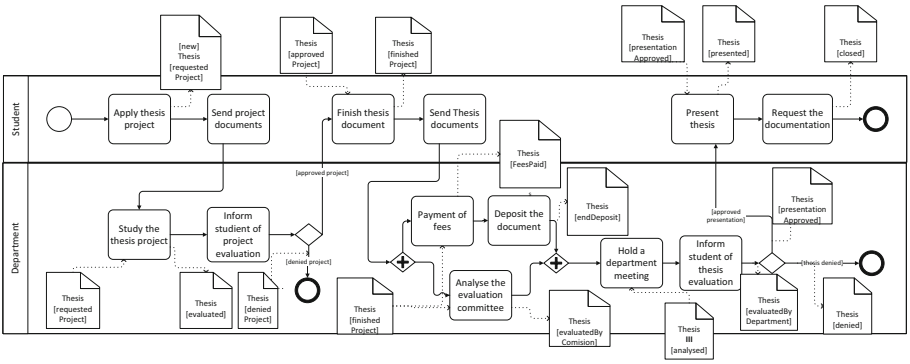

Fig. 4 Thesis process with data description

possible combinations of these values at in different moments of the process instance that must be described.

Definition 2 (Class State) Let C be a class of the conceptual model with its attributes A_C, where each attribute $a \in A_C$ has a domain D_a. A *state* s of C is defined as a set of tuples of values formed by the attribute of C (A_C), in the domain D_a. Every possible tuple of s can be sometimes described in a more compact way by means of a constraint $Const^s$. This constraint describes the possible values of the tuples related to the attributes involved in the state description (A_{Const}), where $A_{Const} \subseteq A_C$. Constraints in the representation of the states can to be stored in constraint databases [24,25].

The grammar to express constraints corresponds with a propositional formula that consists of atoms and, in addition, the logical operators AND, OR, and NOT [26]. These operators can create a Boolean combination of constraints that relate numerical variables [27], as defined below.

Constraint : Atomic_Constraint ***BOOL_OP*** Constraint
| Atomic_Constraint | **'NOT'** Constraint

BOOL_OP : **'AND'** | **'OR'**

Atomic_Constraint : Function ***PREDICATE*** Function

Function : Variable ***FUNCTION_SYMBOL*** Function | Variable

Variable : Attribute | Constant | null

PREDICATE : '=' | '<' | '≤' | '>' | '≥'

FUNCTION_SYMBOL : '+' | '-' | '*' | '/'

In order to illustrate the semantics of the constraints, the constraints that describe the states "requested project" and "approved project" are shown below. However, as shown in Sect. 5.1.2, in order to prevent the users from directly defining constraints, a natural-like language has been also included, to facilitate the description of the data object states.

```
Requested project:
    NOT title = null AND
    NOT sendDate = null AND
    NOT abstract = null AND
    evaluationDate = null AND
    reasonOfRejection = null AND
    supervisor = null
```

```
Approved project:
    NOT title = null AND
    NOT sendDate = null AND
    NOT abstract = null AND
    NOT evaluationDate = null AND
    reasonOfRejection = null
    numSupervisors > 0 AND numSupervisors < 3
```

3 Verification of annotated business process models with data states

The states of the business data objects can be associated with various activities depending on the workflow of the process model, as explained in Sect. 2.3. In order to verify the correctness of the business process control flow and the data states, we define two properties (i.e. consistency and completeness), analysed in Sects. 3.1 and 3.2. From the point of view of the data objects, it is necessary to analyse the data correctness and ascertain both the state of each data object and whether they satisfy any of the states annotated in the process model, as explained in Sect. 3.3.

3.1 Verification of model correctness

Before explaining the aforementioned definitions, certain other definitions must be introduced regarding the possible relations allowed between the data states and the activities of the model. We use the convention that each state is related with only one class, and delegate the possibility of modelling the attributes of different classes in one single state to future work.

Depending on the association of the data state with the activities, and the association between the activities themselves, we differentiate between the following relationships between each pair of states:

- *Two states are exclusive* if the constraints that represent each state are not consistent with each other, and therefore an object cannot satisfy the constraints of both states at the same time. This situation occurs if both states are associated with input or output data of activities that have a sequential or exclusive relation (associated with different branches of an XOR gateway). This means that these activities or branches cannot be executed in parallel. For instance, in the example of Fig. 4, [*requested*] and [*evaluated*] of *Thesis Project* and [*approved*] and [*denied*] of *Thesis* are mutually exclusive.
- *Two states are parallel* if the constraints that represent each state are consistent with each other, and therefore an object can satisfy the constraints of both states at the same time. It implies that they are associated with input or output data of the activities that can be executed at the same time. For instance, in the example, [*evaluatedByCommission*] and [*endDeposit*] of *Thesis* are parallel.
- *One state is a substate of another state (super-state)* if an object that satisfies the constraints of the super-state also satisfies the constraint of the substate. The substates are associated with output data of activities, while the super-state is associated with input data of the subsequent activity, and there are no activities between the substates and the super-states. For instance, [*analysed*] of *Thesis* is a super-state of both substates [*endDeposit*] and [*evaluatedByCommission*].

Based on the concepts above, the definitions for their formalisation are:

Definition 3 (Exclusive, Parallel, Substate, Super-State) Let s_1 and s_2 be two states that belong to the same class C.

State s_1 is *exclusive* to s_2 if there is no tuple t of values of the attributes of class C that satisfies the constraint $Const^{s_1} \wedge Const^{s_2}$.

State s_1 is *parallel* to s_2 if there is a tuple t of values of the attributes of class C that satisfies $Const^{s_1} \wedge Const^{s_2}$.

State s_1 is a *substate* of s_2 if the set of tuples that satisfy s_1 are included in the set of tuples that satisfy s_2.

If s_1 is a substate of s_2, then s_2 is called the *super-state* of s_1.

Definition 4 (Consistent) Let $S(C)$ be a set of states defined for the class C. Set $S(C)$ is *consistent* with respect to a process graph $\langle N, E \rangle$ (Definition 1), if and only if:

- For every pair of task nodes $n_a, n_b \in N$ of the process graph, where n_a, n_b are either executed sequentially or exclusively (but not in parallel), $\forall_{st_1,st_2} \in \{S(n_a)_{IN} \cup S(n_a)_{OUT} \cup S(n_b)_{IN} \cup S(n_b)_{OUT}\}$, st_1 and st_2 are exclusive states.
- For every pair of nodes n_a, $n_b \in N$ of the process graph, where $OUT(n_a) = N_{PJ_i}$ and $OUT(n_b) = N_{PJ_i}$ (the activities located just before an AND-join), $S(n_a)_{OUT}$ and $S(n_b)_{OUT}$ are parallel states. In an equivalent way, where $IN(n_a) = N_{PS_i}$ and $IN(n_b) = N_{PS_i}$ (the activities located just after an AND-split), $S(n_a)_{IN}$ and $S(n_b)_{IN}$ are parallel states.
- For every pair of nodes n_a, $n_b \in N$ of the process graph, where $OUT(n_a) = N_{P_i}$ and $IN(n_b) = N_{P_i}$ (the AND-join or AND-Split is located between two activities), $S(n_a)_{OUT}$ is a substate of $S(n_b)_{IN}$ and $S(n_b)_{IN}$ is a super-state of $S(n_a)_{OUT}$.

3.2 Verification of the model completeness

A data object of the database can take any possible combination of values, which means that the values of their attributes may be very different. If not every combination of attributes is satisfied by a states, then a data object could be in an unknown state. To ascertain if the annotated business process can produce this situation, it is necessary to analyse the property of completeness, as explained below.

Definition 5 (Complete) Let $S(C)$ be a set of states defined on class C. Set $S(C)$ is *complete* if and only if, for each possible tuple of values for the attributes A_C, there is at least one state whose constraint is satisfied.

3.3 Verification of data object correctness

If an annotated business process is *Complete*, then it is possible to guarantee that every stored data object satisfies a state. In the opposite way, it is necessary to analyse every data object to ascertain whether it is correct or not.

Definition 6 (Correct Data Object) Let c be an instantiated object of the class C, whereby c is *correct* if and only if c satisfies at least one state constraint. If c satisfies more than one state, then all of these states must be parallel.

When the constraint associated with a state is satisfied for the values of the attributes of a data object, the state of the data object is known.

4 Algorithms to verify the annotated process models with data object states

In order to verify the annotated business model, it is necessary to verify the consistency (Definition 4) and the completeness (Definition 3.2) of the model, and the correctness of the data objects according to the constraints that describe the states. We tackle the achievement of these objectives using constraint programming combined with certain made-to-measure algorithms. In Sect. 4.1, the constraint programming aspects necessary to understand the solution are presented. The BPMN model represented as a graph (Definition 1) is employed to discover the state relationships, as given in Sect. 4.2. Sections 4.3 and 4.4 analyse the consistency and the completeness of the states according to the business process model. Finally in Sect. 4.5, the correctness of the stored objects is studied according to the data states and the business process model.

4.1 Introduction to constraint satisfaction problems

A constraint satisfaction problem (CSP) represents a reasoning framework consisting of variables, domains, and constraints. Formally, it is defined as a tuple $\langle X, D, C\rangle$, where $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of variables, $D = \{d(x_1), d(x_2), \ldots, d(x_n)\}$ is a set of domains of the values of the variables, and $C = \{C_1, C_2, \ldots, C_m\}$ is a set of constraints. Each constraint C_i is defined as a relation R on a subset of variables $V = \{x_i, x_j, \ldots, x_l\}$, called the *constraint scope*. The relation R may be represented as a subset of the Cartesian product $d(x_i) \times d(x_j) \times \ldots \times d(x_l)$. A constraint $C_i = (V_i, R_i)$ simultaneously specifies the possible values of the variables in V that satisfy R. Let $V_k = \{x_{k_1}, x_{k_2}, \ldots, x_{k_l}\}$ be a subset of X, and an l-tuple $(x_{k_1}, x_{k_2}, \ldots, x_{k_l})$ from $d(x_{k_1}), d(x_{k_2}), \ldots, d(x_{k_l})$ can therefore be called an *instantiation* of the variables in V_k. An instantiation is a solution if and only if it satisfies the constraints C.

In order to solve a CSP, a combination of search and consistency techniques is commonly used [28], and depending on whether a solution for a CSP can be found, certain deductions can be made. For example, in order to ascertain whether a pair of states are exclusive, it is necessary to determine the existence of a tuple of values that satisfy the constraints of both states. If a tuple of values is found when the CSP is solved, it means that these states are not exclusive.

4.2 Structure to analyse the business process model and data state descriptions

In order to determine the *Exclusive*, *Sub*, *Super* and *Parallel* relationships, it is necessary to traverse the *Process Graph* to detect the relationships between the activities and the states of the objects that are read and written during the process execution.

By using the introduced definition of *Process Graph*, and supposing that the business process workflow model is correct, a number of methods have been developed in the *Process Graph* to facilitate the implementation used in the following algorithms. The methods are:

- *Node getStartEvent()* returns the start event of the graph.
- *List⟨Node⟩ getNeighbours(Node n)* returns the list of neighbour nodes of n, whereby there is a directed edge from *n* to these nodes.
- *List⟨Node⟩ getAllNodes()* returns the list with every node of the graph.

- *Constraint getInState(Node n)* returns the constraint that represents the state of an object before the node n is executed.
- *Constraint getOutState(Node n)* returns the constraint that represents the state of an object after the node n is executed.
- *Boolean isActivity(Node n)* returns $true$ if n represents an activity.
- *Boolean isSplitControlFlow(Node n)* returns $true$ if n represents a split control flow.
- *Boolean isJoinControlFlow(Node n)* returns $true$ if n represents a join control flow.
- *List⟨Node⟩ getAllAndSplitControlFlows()* returns a list with all the AND-split control flows of the process graph.
- *List⟨Node⟩ getAllANDJoinControlFlows()* returns a list with all the AND-join control flows of the process graph.
- *List⟨Node⟩ getPreviousOf(Node n)* returns a list with all the nodes $n_1 \ldots n_m$, if there exists a directed edge from each $n_i \in \{n_1 \ldots n_m\}$ to n.

As explained in the definitions above, the consistency and completeness are related to the solution that constraints of the states share with other states. To ascertain this solution, we propose the creation of a constraint satisfaction problem that provides us with information about the properties of the states. To create and solve the CSP, a *Solver* class is defined, which contains the following methods:

- *solver (Constraint c)* is the Constructor to create a *Solver* variable that represents a Constraint Satisfaction Problem with the constraint c.
- *Constraint createOrConstraint (List⟨Constraint⟩ l)* returns a constraint formed by an OR combination of the constraints that form the list l: $\{l_1 \vee \ldots \vee l_n\}$.
- *Constraint createGreaterThanOneConstraint (List⟨Constraint⟩ l)* returns a constraint to indicate whether more than one of the constraints of the list l can be satisfiable at the same time ($l_1 + \ldots + l_n > 1$). The assignation of a numerical value of a constraint involves indicating whether this constraint is satisfiable or not. If there exists a tuple of values that satisfies this constraint, then the value associated with the constraint is 1 (true), 0 otherwise.
- *void solve()* is a method of the class *Solver* that resolves the Constraint Satisfaction Problem.
- *Boolean hasSolution()* indicates the existence of a tuple of values that satisfies the CSP.

4.3 Verification of consistent property

The CSPs created in each case explained in Definition 4 are presented below.

4.3.1 Exclusive state property

The consistency of a model implies that every pair of input and output data states of activities that are either executed sequentially or exclusively (but not in parallel) must be exclusive. This demonstrates that the constraints of two exclusive states cannot be satisfied at the same time. The idea of the CSP is to model a combination of constraints in order to ascertain the existence of a tuple of values that satisfies more than one exclusive *Class State*. For this reason, the CSP is formed of a list of OR ($\vee$) constraints: $C_1 \vee C_2 \vee \ldots \vee C_n$, where each C_j represents a summation of the constraints of the exclusive states ($l_1 + \ldots + l_n > 1$ using *createGreaterThenOneConstraint* method). If the solver (using the *solver* method) finds a tuple of values that makes it possible that more than one constraint is true ($l_1 + \ldots + l_n > 1$), then two exclusive states can be satisfiable at the same time, and the consistency fails.

For example, $[requestedProject]$, $[evaluatedProject]$, $[approvedProject]$ and $[deniedProject]$ are exclusive states for the workflow data relation, but if two of these states could be satisfiable at the same time (whereby the sum of their true values are greater than 1), then the workflow model is not consistent according to the constraint description of the data states. For the example of Fig. 4, the following CSP is created to verify the consistency concerning exclusive property:

$$
\begin{aligned}
&(\mathrm{Const}^{requestedProject} + \mathrm{Const}^{evaluated} + \mathrm{Const}^{approvedProject} + \mathrm{Const}^{deniedProject} \\
&+\mathrm{Const}^{finishedProject} + (\mathrm{Const}^{FeesPaid} \vee \mathrm{Const}^{endDeposit} \vee \mathrm{Const}^{evaluatedByCommission}) \\
&+\mathrm{Const}^{analysed} + \mathrm{Const}^{evaluatedByDepartment} + \mathrm{Const}^{presentationApproved} \\
&+\mathrm{Const}^{evaluationDenied} + \mathrm{Const}^{presented} + \mathrm{Const}^{close} > 1) \\
&\vee \\
&(\mathrm{Const}^{FeesPaid} + \mathrm{Const}^{endDeposit}) > 1
\end{aligned}
$$

In order to create the CSPs automatically, we have developed a recursive algorithm for exclusive state verification (Algorithm 2), whose complexity is linear for the number of nodes, since each node is analysed only once. The input parameters of Algorithm 2 specify the graph to traverse following the structure introduced in Sect. 4.2, the node employed to start the algorithm, and the input/output data states obtained from the execution of the algorithm. The output of the recursive function consists of the subsequent node analysed in the process of traversing.

Starting with the whole graph, Algorithm 1 is employed to initiate the recursive process with the start event. The list is used for the collection of the $C_1 \vee C_2 \vee \ldots \vee C_n$ constraints for the main process (*mainStates*), and of the OR relations found in the nested control-flow structures of the BPMN graph (*nestedStates*). Algorithm 1 initialises the call to the recursive Algorithm 2, which returns the two lists (*mainStates* and *nestedStates*) with the exclusive relations of the states. The CSP is created with an OR relation between the constraints of the lists, as explained above ($C_1 \vee C_2 \vee \ldots \vee C_n$). When a solution is found (with the method *hasSolution()*) for the various OR relations, it means that certain exclusive states can be satisfied at the same time. If no solution is found, then the exclusive states are verified according to the workflow of the process model.

Algorithm 1 Algorithm for Exclusive State Verification

```
1: function ExclusiveStateVerification(Graph g)
2:     List<Constraint> mainStates = new List<Constraint>();
3:     List<Constraint> nestedStates = new List<Constraint>();
4:     Node n = g.getNeighbours(g.getStartEvent());
5:     ExclusiveStateVerification(g, n, mainStates, nestedStates);
6:     Solver s = new Solver(createOrConstraint(nestedStates) OR createGreaterThanOneCon-
   straint(mainStates));
7:     s.solve();
8:     if s.hasSolution() then
9:         Print("Error in the verification: Some exclusive states can be satisfied at the same time.");
10:    else
11:        Print("Exclusive states are correct and they cannot be satisfiable at the same time.");
12:    end if
13: end function
```

Algorithm 2 traverses the process graph g from the node n, to find the exclusive relation between the states, and to collect the list of constraints. The algorithm includes the following:

– *From line 2 to 6* The states of the sequential activities are included in the *mainStates* list in order to compose the set of constraints whose summatory (true-value) is compared with 1. For this reason, when no branches are found in the traverse of the graph, the input and output data states are included, and a recursive call is performed in order to continue with the subsequent neighbour node of the activities.
– *From line 8 to 25* If a split control flow is found, it means that the exclusive relations between the activities of each branch can exist. For this reason, two lists are created (*localMain* and *localNested*) to recursively call to each branch as a smaller recursive problem. For each branch (line 12), the two lists obtained are incorporated into the general lists of lists (*allLocalMain* and *allLocalNested*) that will be incorporated to the general lists (line 23 and 24). The traverse of the algorithm follows on with the neighbour of the join control flow of the split process (line 17 and 18).
– *From line 26 to 28* When an end event or a join gateway is found, it indicates that a nested call has finished, and the execution then returns to the point where the recursive call was made.

Algorithm 2 Recursive Algorithm for Exclusive State Verification

```
1: function ExclusiveStateVerification(Graph g, Node n, List<Constraint> mainStates,
   List<Constraint> nestedStates)
2:     if g.isActivity(n) then
3:                                                  ▷ An activity has only one neighbour
4:         Node neighbour = g.getNeighbours(n).get(0);
5:         mainStates.add(g.getInConstraint(n));
6:         mainStates.add(g.getOutConstraint(n));
7:         return (ExclusiveStateVerification(g, neighbour, mainStates, nestedStates))
8:     else if g.isSplitControlFlow() then
9:         List<Node> neighbours = g.getNeighbours(n);
10:        List<List<Constraint>> allLocalMain = new List<List<Constraint>>();
11:        List<List<Constraint>> allLocalNested = new List<List<Constraint>>();
12:        Node final_nodel = null;
13:        for neighbour in neighbours do
14:            List<Constraint> localMain = new List<Constraint>();
15:            List<Constraint> localNested = new List<Constraint>();
16:            Nodel local_end = ExclusiveStateVerification(g, neighbour, localMain, localNested);
17:            if g.isJoinControlFlow(local_end) then
18:                final_nodel = local_end
19:            end if
20:            allLocalMain.add(createOrConstraint(localMain));
21:            allLocalNested.add(createGreaterThanOneConstraint(localNested));
22:        end for
23:        mainState.add(createOrConstraint(allLocalMain));
24:        nestedState.add(createOrConstraint(allLocalNested));
25:        Node neighbour = g.getNeighbours(final_nodel).get(0);
26:        return ExclusiveStateVerification(g,neighbour,mainState,nestedState)
27:    else
28:        return n
29:    end if
30: end function
```

4.3.2 Parallel state property

Following Definition 4, the consistency of a model also implies that every pair of states of data output of the activities located immediately before an AND-join must be parallel, if they exist. This means that both states can be satisfiable at the same time, since an object can belong to both states at the same time. Formally, state s_i is *parallel* to s_j if $\forall$ tuple t that satisfies $Const^{s_i}$, $\exists$ another tuple t' that satisfies $Const^{s_j}$, where $\forall$ attributes $a \in A^{s_i}$, the values in the tuples t and t' are equal, and $\forall$ a $\in A^{s_j}$, the values in the tuples t and t' are equal.

In order to ascertain whether this property is satisfiable or not, for each possible pair of states $Const^{s_i}$ and $Const^{s_j}$, a CSP is created as: $\{\text{Const}^{Si} \wedge \text{Const}^{Sj}\}$. If a solution cannot be found, then the states cannot be parallel.

The CSP created for the example is:

$$\boxed{\text{Const}^{endDeposit} \wedge \text{Const}^{evaluatedByCommision}}$$

We have developed the *ParallelStateVerification* Algorithm (Algorithm 3) that creates and solves every CSP needed in order to ascertain whether the parallel states are correct. The algorithm studies every And-Join gateway (line 2). For each of these control flows, a CSP is created with every constraint of the output data of the activities that are joined in each control flow (lines 5 to 10). These constraints are related in the CSP with an AND Boolean operator. If a solution is found, then these constraints can be satisfiable in parallel, and the parallel objects are verified. If a solution is not found, then the data model is not consistent with the workflow according to the parallel property.

Algorithm 3 Algorithm to Analyse the parallel states

```
1: function ParallelStateVerification(Graph g)
2:     List<Node> andControlFlowList = g.getAllANDJoinControlFlows();      ▷ List of And Joing nodes
3:     for n in andControlFlowList do
4:         Constraint c = new Constraint();
5:         List<Node> parallelList = g.getPreviousOf(n);
6:         for p in parallelList do
7:             c.add(g.getOutConstraint(p));
8:         end for
9:         Solver s = new Solver(c);
10:         s.solve();
11:         if s.hasSolution then
12:             Print("The output states before the AND node " + n + " are parallel");
13:         else
14:             Print("The output states before the AND node " + n + " are NOT parallel, and hence the consistent
    property is not satisfiable");
15:
16:         end if
17:     end for
18: end function
```

4.3.3 Substate and super-state property

Following Definition 4, the consistency of a model also implies that every state of data output of the activities located immediately before an AND-join are substates of the input data state (super-state) of the first activity after the AND-join, if this subsequent activity exists.

If A is a subclass of B, S_i is the state of data output of A, and S_j is the state of the data input of B, then both states are consistent if $\nexists$ a tuple t that satisfies S_i and not S_j. We propose building a CSP to ascertain whether the opposite exists:

$\exists$ a tuple $t \vDash \text{Const}^{S_i} \wedge \neg\text{Const}^{S_j}$.

The CSP created for the example is:

$$\text{Const}^{evaluatedByCommission} \wedge \neg\text{Const}^{analysed} \vee \text{Const}^{endDeposit} \wedge \neg\text{Const}^{analysed}$$

If a solution is found for the CSP, then the model is not consistent in terms of the substate and super-state property.

4.3.4 Algorithm to obtain the CSP for the verification of substate and super-state relations in the consistent property

Sub&Super-StateVerification Algorithm (4) creates and solves the CSP needed to verify the consistency of the substate and super-state relations. The algorithm analyses every join control flow (AND, OR, or XOR) (line 2), and includes a set of constraints related by means of OR Boolean operator ($\vee$) in the CSP. Each of these constraints constitute an AND combination between the constraints of the output data of the substates, and the input data of the super-state ($\text{Const}^{substate} \wedge \neg\text{Const}^{super-state}$) (lines 5 to 10). If a solution is found for the CSP (line 11), then the model is not correct according to the substate and super-state consistency property. Otherwise, the model is consistent for this property.

Algorithm 4 Algorithm to Analyse the Substate and Super-states

```
1: function SUB&SUPERSTATESTATEVERIFICATION(Graph g)
2:     List<Node> andControlFlowList = g.getAllJoinControlFlows();        ▷ List of And Joing nodes
3:     for n in andControlFlowList do
4:         Constraint c = new Constraint();
5:         List<Node> parallelList = g.getPreviousOf(n);
6:         for p in parallelList do
7:             c.or(g.getOutState(p) ∧ ¬ g.getInState(n));
8:         end for
9:         Solver s = new Solver(c);
10:         s.solve();
11:         if s.hasSolution() then
12:             Print("The model is NOT correct according to Substate and Super-State Consistency Property");
13:         else
14:             Print("The model is correct according to Substate and Super-state Consistency Property");
15:         end if
16:     end for
17: end function
```

4.4 Verification of completeness property

The completeness of a data object model is satisfied if every possible value of the attributes of a Class corresponds with at least one modelled state. This prevents the existence of a business data object in the database that does not correspond to a modelled state, or to satisfy a constraint that describes an incorrect behaviour. It implies that $\forall$ tuple t of values of a class C, $\exists$ a state of C whose constraint is satisfied by t.

The CSP created to verify the completeness incorporates the negation ($\neg$) of the modelled state constraints. If a solution is found for the CSP, it means that the model is not complete since there exists a tuple where no state is satisfiable. For the example, the CSP has the form:

$$\neg\text{Const}^{requestedProject} \wedge \neg\text{Const}^{evaluated} \wedge \neg\text{Const}^{deniedProject} \wedge \neg\text{Const}^{approvedProject} \wedge \neg\text{Const}^{finishedProject} \wedge \neg\text{Const}^{evaluatedByCommision} \wedge \neg\text{Const}^{endDeposit} \wedge \neg\text{Const}^{analysed} \wedge \neg\text{Const}^{evaluated} \wedge \neg\text{Const}^{feesPaid} \wedge \neg\text{Const}^{approvedPresentation} \wedge \neg\text{Const}^{deniedPresentation} \wedge \neg\text{Const}^{presented} \wedge \neg\text{Const}^{close}$$

If a solution is found for the CSP, then the model is not consistent for the completeness property. *CompletenessStateVerification* Algorithm (Algorithm 5) creates and solves the CSP needed to ascertain whether the data model description is complete. The algorithm creates a CSP with an AND Boolean relation ($\wedge$) of every constraint of the states of the model for each class (line 2). The input and output constraints are included with a negation ($\neg$) (line 5). If a solution is found, then the model is not complete, according to the completeness consistency property.

Algorithm 5 Algorithm to Analyse the property of completeness

```
1: function CompletenessStateVerification(Graph g)
2:     List<Node> allnodes = g.getAllNodes();          ▷ List of every node
3:     Constraint c = new Constraint();
4:     for n in allnodes do
5:         c.and(¬g.getInState(n) ∧ ¬g.getOutState(n));
6:     end for
7:     Solver s = new Solver(c);
8:     s.solve();
9:     if s.hasSolution() then
10:        Print("The model is NOT correct according to the Completeness Consistency Property");
11:    else
12:        Print("The model is correct according to the Completeness Consistency Property");
13:    end if
14: end function
```

4.5 Data object correctness: discovering the States of stored data objects

Although the model of a business process is correct according to its business data states, it is necessary to analyse whether the objects stored in the database are correct with respect to the designed process model. This implies ascertaining whether every stored object is in one of the described states, or, in an equivalent way, whether there is an object that is not in any state. This situation could arise since the objects can follow legacy models or business rules of the past, which are not supported by the current business process model. This analysis ensures that

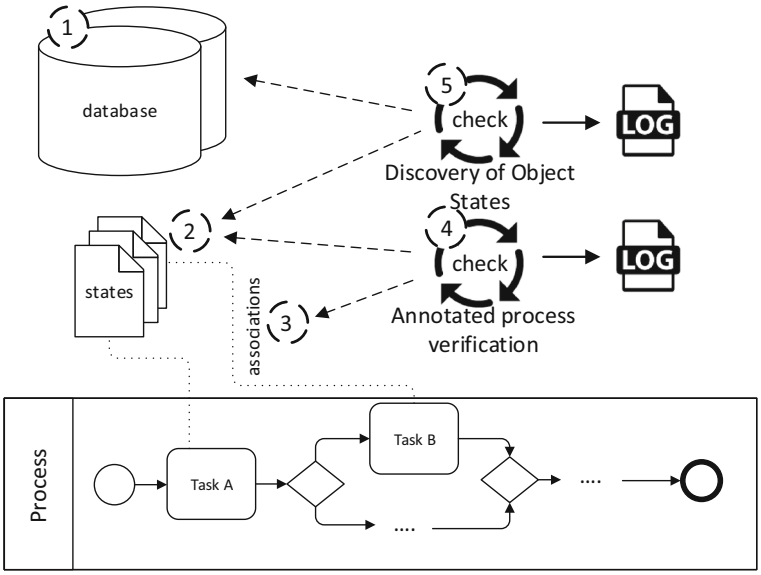

Fig. 5 Steps of the methodology

every object is correct or detects the incorrect objects. As mentioned earlier, an object can be in more than one state (parallel states), and hence Algorithm *CorrectnessObjectVerification* (6) ascertains whether the objects are in at least one state, and either reports the states of the object, or returns *NONE* otherwise (lines 6 and 7).

Algorithm 6 Algorithm to Analyse the property of correctness of the stored Objects

```
1: function CorrectnessObjectVerification(<Type> DataObject)
2:     Iterator iter = session.createQuery("from" + DataObject).iterate();
3:     for iter.hasNext() do
4:         <Type> t = (<Type>) iter.next();
5:         t.calculateState();                          ▷ Method of Figure 6
6:         if t.getState()==NONE then
7:             Print("The object " + t + " is not in a correct state.");
8:         else
9:             Print("The object " + t + " has the State " + t.getState());
10:        end if
11:    end for
12: end function
```

5 Methodology and framework for annotated business Process verification and discovery of object states

The proposed methodology has been developed with a framework whose implementation details are explained in the following subsections. The methodology devises the steps shown in Fig. 5, which include:

1. *To describe the business data objects involved in the business process in accordance with the conceptual model of the database* Since the relational model is highly detailed and is difficult to understand and query by non-expert users, we propose the use of a conceptual model managed by means of an object-relational mapping (ORM) to facilitate the description and management of the business data objects. This conceptual model is also employed to describe the data states according to the attributes and associations of the model. This step must be developed by an IT expert since technical knowledge is necessary.
2. *To describe the state of each business data object in terms of the value of its attributes* In order to bring the experts and the state descriptions closer, it is necessary to provide a natural language to describe why one business data object is in one state or another. Knowledge of the different states of the business object is held by the business experts. Therefore, we have implemented a domain-specific language (sold out in following Subsections) within the business process modeller (Activiti^{TM} in our proposal) to facilitate the business expert task concerning the data state description.
3. *To allocate the data state description in the business process model* The importance of our proposal lies in its capacity to combine the data state and the workflow of the model. The third step is based on the location of the data states related to the activities executed at each moment. This enables the specification of which data objects are read, and which states are obtained after an activity execution to be incorporated into the workflow. This step can be developed by a business expert since no technical knowledge is necessary.
4. *To verify the model correctness* The states of the objects associated with the activities of the model need to be consistent with respect to the workflow. The possible states that an object can satisfy, and where it is read or written must be analysed. Algorithms 4 and 5 are used in this step to analyse the process model correctness in an automatic way.
5. *To discover the stored business objects* The model is created to work with the stored business objects. Algorithm 6 is used to verify whether every business object satisfies any state. This is an automatic task developed by our system using the ORM framework to be iterated in the objects of the database.

5.1 Framework to support the methodology

In order to validate our proposal, a tool that supports the methodology has been developed by using a set of mature technologies. Note that this is just one of many possible implementations and others with different technologies are also possible. The parts that must be supported include: the data store that contains the involved data objects; the editor to describe the states in a friendly language; the connection between the commercial business process and the stored objects; the implementation of the algorithms to verify the correctness of the model; and the setting for the analysis of every stored object to ascertain the state of each object. An Eclipse (RCP) Rich Client Platform application has been developed to include all these elements in the same environment.

5.1.1 Business data model description: object-relational mapping to represent business data objects

Facilitating the state description also implies managing the information represented by means of the conceptual model. It is the object-relational mapping (ORM) that is employed to manage the relational model by means of the conceptual model that represents it.

Table 1 DSL transformations for the class state grammar

`{ClassName}{`	public class {ClassName}{}
	`static final int {State1}=0`
	…
	`@PostLoad`
	`@PostUpdate`
	`private void calculateState()`

`♯[{State1}]`
//Boolean combinations of comparisons
…
`♯[{StateN}]`
//Boolean combinations of comparisons
…
	`if(java code transformation)`
	state = {State1};
	…
	`if(java code transformation)`
	state = {StateN};
	…

}	}

Although most software applications use relational databases to store their data, object-oriented language is employed to manage the information. This implies the mapping from the primary and foreign key relations to an object-oriented paradigm and vice versa. In our case, this introduced the needed for the use of a conceptual model instead of the relational model, which tends to be solved using data access object (DAO) patterns. Common data access object (DAO) implementations are provided by object-relational mapping (ORM). ORM [29] is a programming technique for the conversion of data between systems of incompatible types in object-oriented programming languages. The use of ORM brings major benefits, such as database independence, low coupling between business and persistence, and fast software development. How ORM can be used in this context and how the example should be annotated is detailed in [23].

5.1.2 Business data state description: a natural-like language to describe the data object states

In order to describe the *Class State* following the grammar presented above, we have created a domain-specific language (DSL) to help the business expert in the description of the data states. The proposed DSL follows the grammar below:

```
list_Of_Classes := Class list_Of_Classes
 | Class
Class := '{'ClassName'}'{'list_Of_States'}'
list_Of_States := State list_Of_States
 | State
State := '#'[StateName] Constraint
```

Several examples of how the sentences of the language are transformed automatically into Java code are shown in Table 1. The `Constraint` term of the grammar corresponds to the Constraint grammar described in Definition 2 (Class State). However, in order to provide a natural-like language, a set of natural expressions are created to refer to the tokens `PREDICATE` and `FUNCTION_SYMBOL`, as described in Table 2.

Table 2 DSL pattern transformation for the Constraint Grammar

Token	DSL pattern	Java code pattern	Example
= null	{attribute} IS EMPTY	get{Attribute}()==null	presentationDate is empty
<> null	{attribute} IS NOT EMPTY	get{Attribute}()!=null	presentationDate is not empty
Count =	THE NUMBER OF {attributeSet} IS {number}	get{AttributeSet}().size() ==\{number\}	the number of evaluation comittee is 5
Count <>	THE NUMBER OF {attributeSet} IS DIFFERENT TO {number}	get{AttributeSet}().size() ! =\{number\}	the number of evaluation committee is different to 5
=	{attribute} IS EQUAL TO {value}	get{Attribute}() ==\{value\}	Mark is equal to 'FAIL'
<>	{attribute} IS DIFFERENT TO {value}	get{Attribute}()!={value}	Qualification is different to 'PASS'
<	{attribute} IS LESS THAN {value}	get{Attribute}()<{value}	DepositDate is less than currentDay
≤	{attribute} IS LESS THAN OR EQUAL TO {value}	get{Attribute}()<={value}	DepositDate is less than or equal to currentDay
>	{attribute} IS GREATER THAN {value}	get{Attribute}()>{value}	EvaluationDate is greater than sendDay
≥	{attribute} IS GREATER THAN OR EQUAL TO {value}	get{Attribute}()>={value}	EvaluationDate is greater than or equal to sendDay

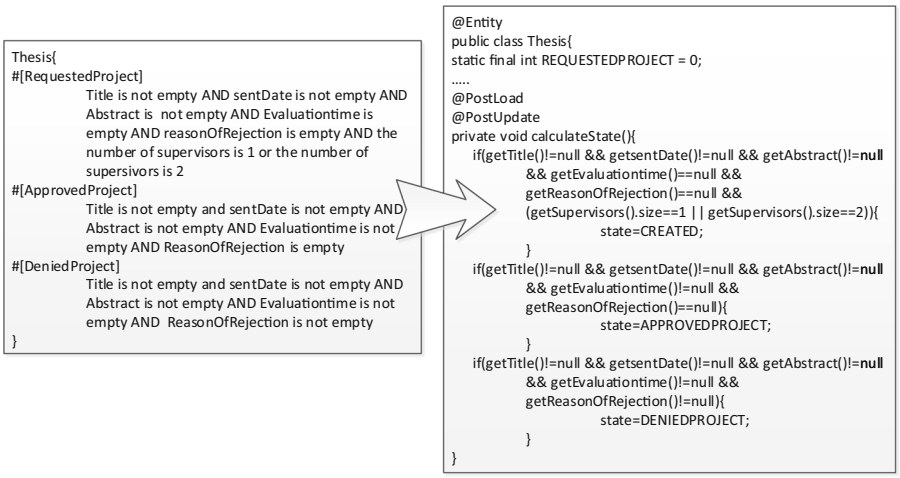

Fig. 6 Example of mapping between the DSL and Java code

As mentioned earlier, it is not always possible to link the state of the data object as a string within a column of the corresponding table in the legacy database. Moreover, the state is a characteristic of the object, and it is not mandatory to be stored as one of the stored attributes. For this reason, the state is presented as the attribute `@Transient` of the mapping classes. The attributes annotated with `@Transient` represent those that are part of the entity but are not required to be persistent, since they do not belong to the database. When the object is loaded with the information of the database or updated by means of the setting methods, the state needs to be updated in terms of the values of the remaining attributes. In Hibernate, there exist call-back methods that should be prefixed by annotations that dictate when these methods have to be executed. Since the change of the state occurs when the object is loaded or updated, `@PostUpdate` and `@PostLoad` annotations are used. Those two annotations represent, respectively, that the method *calculateState()* must be invoked for an entity after a constructor or that the update operation is executed. How the states are represented in the Java code and how the annotation the classes of the ORM framework are annotated are detailed in [23].

5.1.3 DSL transformation into Java code

The transformation of this description into executable code is also necessary to enable the automatic evaluation of the model and the validation of every object of the database to find the states at each moment.

In Fig. 6, a transformation is shown from a description of the states of the *Thesis* class (RequestedProject, ApprovedProject, DeniedProject...), to the Java code included automatically into the ORM classes. When an object is loaded or updated, the state of the data object is updated, and therefore the activity that is being executed for each data object can be ascertained. The DSL patterns can be combined by means of Boolean connectors (AND, OR, NOT) to create states with greater complexity.

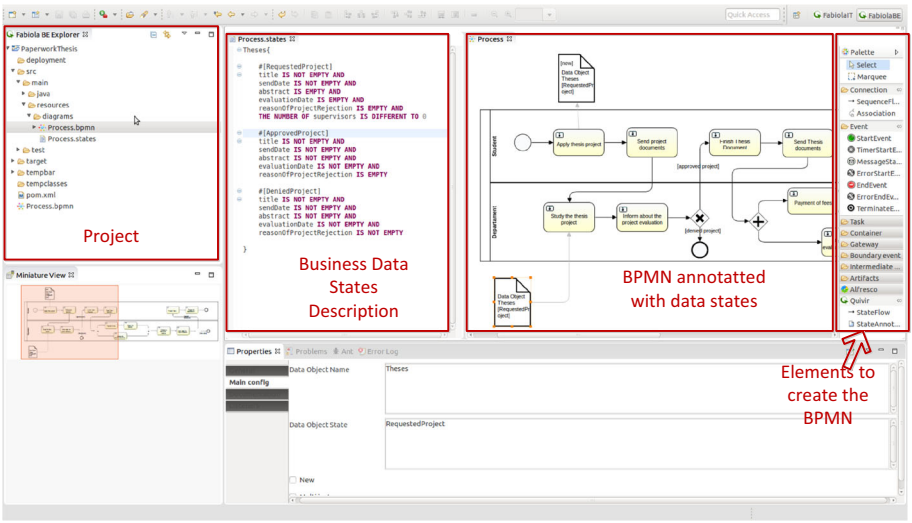

Fig. 7 Activiti^{TM} extension view for BP annotation

5.2 Annotate the business process model with data states: implementation of a tool to support the proposal

In order to integrate every part of the model into a same environment, an implementation has been developed as an extension of Activiti^{TM} [30]. Activiti^{TM} is a light-weight workflow and Business Process Management Platform targeted at business people, developers, and system administrations. This is an open-source distribution that offers the business expert a friendly interface to model the process using BPMN elements. Since the data state description and management is not included in Activiti, an extension of Eclipse IDE-based Activiti Designer has been developed to support new graphical elements.

Our implemented Activiti^{TM} framework supports the development of the five steps presented in the methodology. This extension enables modellers to work in two different views: for Information Technology (IT) experts, and for Business Experts (BE). IT experts can include and describe the database connection and the ORM classes to carry out Step 1. The view for the BEs enables, Steps 2 and 3 to be carried out easily. The ORM classes annotated with @BusinessData are involved in the verification and validation processes (Steps 4 and 5).

As mentioned earlier, in order to facilitate the description of the states by the BE, we have developed a DSL using *xText* [31], which is an open-source framework for the development of programming languages and domain-specific languages. To develop this analysis, Java reflection API has been used. Figure 7 shows the components involved in the workflow creation, data states description, and the Hibernate project to incorporate the relational database.

Once the states are defined and managed entities have been annotated, *xTend* is used to derive an aspect for the calculation of the state of managed entities, according to the states defined by the BE with the grammar.

Having completed the process modelling, the BE must include the data object states in the model by using the new graphical elements developed to annotate the BPMN (as shown

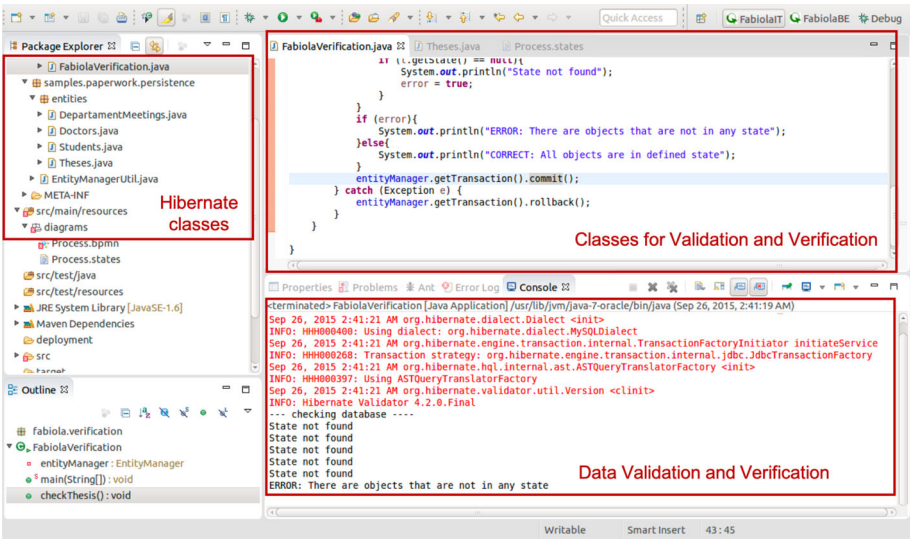

Fig. 8 Activiti^{TM} extension view for process validation

in Fig. 7). Since the Activiti^{TM} tool is developed by using Graphiti (https://eclipse.org/graphiti/), then the new elements for the annotation and validation have also been extended by using this framework.

5.3 Verification of the model and data state discovery

The verification algorithms presented in previous sections create the aforementioned CSPs, and they are solved by using Choco^{TM} solver [32], although it is able to be adapted to another solver [33]. The code of Algorithm 6, included automatically in the Hibernate^{TM} class for each modelled business data object, analyses every object with the iterator and informs whether any object has no state. This extension includes the classes that contain the code for the verification of model and the data states, as is envisioned in Fig. 8.

A set of videos, where the five steps are presented for the example, can be found in http://www.idea.us.es/data-object-verification/.

In summary, the advantages of the implemented tool include:

1. The data object description has been included in Activiti^{TM} commercial tool, as a BPMN 2.0 extension.
2. The ORM code derived from the data object, described in the first step, has been automatically created to facilitate the tasks of the Information Technology experts.
3. The DSL has been included in the framework using aspect-oriented programming to validate the syntax and semantics of the data states during the business process design.
4. The necessary formalisations for the verification have been included. The automatic verification has been designed by means of a set of algorithms that use the constraint programming paradigm. All these algorithms have been included in the extension of Activiti^{TM}.
5. The automatic validation has been implemented and included in the extension of Activiti^{TM}.

Table 3 Evaluation times

Model correctness		Number of constraints	Number of variables	Evaluation time (ms)
Consistency	Exclusive	379	620	234
	Parallel	224	372	3
	Substate	141	239	11

5.4 Evaluation time

The analysis of the completeness and correctness of the data stored, according to the business process model, can present major difficulties. This involves an off-line analysis, since a swift evaluation is not essential. In this section, the evaluation times of the completeness and consistency of the process model and data, and the state discovery of the data objects are shown. We have used the real example presented in Fig. 4, which has been described for the personnel of the University of Seville according to the normative procedures. The relational database employed to store the objects is that of MySQL.

The analysis times for correctness (completeness and consistency) are shown in Table 3. As can be observed, three types of analysis haven been carried out: model consistency (exclusive, parallel, and substate analysis), model completeness and objects correctness. Regarding the model correctness, we have considered it interesting to include the size is of the constraint satisfaction problems created and solved. This size is described by means of the number of constraints that form the CSP and the number of variables.

In the case of the analysis of the correctness of the stored objects (Thesis and Thesis Project), the average evaluation time for Thesis objects is $1.58ms$, while the average of evaluation time for Thesis Project objects in $1.67\,ms$. All the measures are presented in milliseconds and have been obtained by using a 2.2 GHz Intel Core i7 processor and 8 GB 1600 MHz DDR3 RAM.

6 Related work

Much research in BPM has been devoted to the verification of control flow in process models [34–36] in order to detect logical errors before the models are deployed in a workflow engine. This research has been extended to deal with the analysis of process models that contain both control flow and data flow [12,37,38]. The aim of such analysis is to detect logical errors resulting from inconsistencies between control flow and data flow. The errors identified are at a conceptual level; implementation issues are disregarded in these studies. Moreover, the process models use variables rather than data object states. Consequently, the data is represented implicitly, using pre- and post-conditions.

Process models expressed in industrial languages, such as BPMN [7] and UML activity diagrams [39], can use object states. Such process models can contain different states for the same object. These different states implicitly specify the life cycle of that object. The life cycle can be explicitly specified with state machines [39]. Reggio et al. [40] define a more precise version of UML activity diagrams with object states and they empirically validate the level of comprehension. However, these authors fail to consider data persistence, and hence no integration with databases is discussed.

Several papers study how to derive process models from existing object life cycles [41,42], while other papers study how to derive object life cycles from such process models [43–

45]. Meyer and Weske [16,46] discuss the relations between object life cycles and process models. None of these papers consider verification aspects. Instead, they study consistency and transformation between a process model and an object life cycle model that both specify behaviour of the same object. Furthermore, these papers remain at the conceptual level and fail to discuss data persistence and, therefore, the relation with databases.

Cruz et al. [47] define an approach to derive a data model from a BPMN model. The data model does not address different states of the same objects. Moreover, data persistence is not addressed.

While none of these papers considers data persistence, a recent paper [22] proposes an extension of BPMN data objects to facilitate an integration with SQL databases. These authors add annotations to data object states to manage data dependencies and differentiate between instances. These annotations are sometimes very low-level representations (e.g. primary key and foreign key) that have significance only at the database level, not at the conceptual business level. Therefore these annotations are difficult to understand for business stakeholders. Moreover, each state of a data object in the BPMN model is represented by a simple string, which has limited semantic value. In order to increase the semantic value in our proposal, the states are represented by constraints. More importantly, Meyer et al. [22] advocate a top-down design approach, where data persistence is handled in the last stage. Our approach is, in fact, bottom-up, since the data object definitions are derived from the legacy database, which already exists before the process model has been specified.

In a similar top-down fashion, [48] proposes a specific view as a solution for the problem of accessing data of business processes. The data access object (DAO) repository view offers a fast and efficient management of DAOs. In follow-up work [49], the authors have proposed using data access services (DAS) instead of DAOs for the data access. Although these studies are very interesting, their objectives are related to the design of an efficient solution for managing data into processes, whereas we study how to integrate legacy databases in processes; therefore data management is already handled by the legacy databases. In [17], the necessity of including the data stored in the persistent layer is detected and modelled, but the solution is not object-oriented, and it does not support the model and data verification. The stored data verification is studied in [27], and the detection of possible faults and the diagnosis of the incorrect data are developed at run-time. The main difference with the current proposal is that the model and data verification is not performed, and the necessity to include the legacy databases in the validation process is omitted.

Since the incorporation of data aspects in process models has drawn great interest in research, several advanced modelling techniques have been proposed to describe processes and data in an integrated fashion [50]. Important examples include artefacts-centric BPM [51,52] and data-driven BPM [53]. The verification of artefacts-centric models has been studied in the previous work [14,54,55]. The orientations towards the artefacts-centric proposal fail in the disconnection between the activities of the BPMN and the change of states described in the artefacts, since these proposals advocate different modelling techniques to that of BPMN. However, these modelling approaches have yet to be adopted widely in industry, in contrast to BPMN. Our proposal leverages the use of BPMN for the specification of processes supported by legacy databases.

Sun et al. [56] propose a formal approach for the integration of artefacts-centric process models with databases. These authors focus on specifying data mappings between models at both levels and on analysing different properties. The main difference with our approach is that we verify the data state according to the workflow structure, while their approach is based on the comparison between different instances.

To sum up, the main contribution of this paper is the verification of the business process model where business data states and workflow design are combined, and the analysis of the consistency of the stored object read and written during the process execution is included. Moreover, the combined description of the business process annotated with data has been improved to facilitate the task.

7 Discussion of the proposal

Despite the advantageous findings presented in this paper, we have to discuss the results from the perspective of limitations and the scope of application of our proposal should also be discussed.

- *Grammar of the state description* We use a specific grammar and the equivalent DSL to represent the data states. If the stored objects and the states that need to be considered could not be described with the grammar, our proposal could not be used. The limitation of the data domain and of the operations that can be used is established by the solver used for the constraint programming problems. Most of the commercial solvers maintain the capacity to include float, integer, sets, and Boolean variables in the model, thereby making it possible to cover a significant number of problems and their data object states.
- *Non-informative relational database* Our approach is based on the idea that the legacy database has sufficient information stored on the data obtained in the past, and that this data will be used after the incorporation of the new BPMS into the company's daily work. If the legacy database does not have the same data objects that correspond to the annotated BPMN, our proposal holds no interest for the business experts.
- *Data objects used in different business process models* The algorithms and methodology of our proposal work with a single business process, where the whole data life cycle evolves. If the data objects are modified in different business processes, then our proposal needs to be adapted and enlarged to support it. New adaptations will also be necessary if collections of objects are included in the model.
- *Data Objects affected by external processes* The relational databases can be affected by others tools outside of the business process model under verification. External modifications do not contradict both the model and data object correctness proposed in the paper. Our methodology determines that the model is correct (Complete and Consistent), where the life cycle of the object in the process model is correct. Regarding the stored objects, it is possible to ascertain which objects are or are not in a known state according to the process model. However, our methodology is unable to determine whether there are not more data object states or incorrect modifications thereof carried out by other processes.
- *CSP complexity* The complexity of the algorithms developed to analyse the process correctness are linear with respect to the number of nodes of the BPMN Graph, therefore the verification time of the model is linked to the complexity of the resolution of the CSPs. The CSPs complexity has been analysed in great depth over recent decades [57], and depends on two parameters: the width of the graph that relates variables and constraints, and the order parameter.
  In our case, the created CSPs depend on: (1) the type of data state constraints; (2) the number of data states; and (3) the number of data objects involved. Concerning the types of BDCs, it is possible to carry out an analysis on the complexity of the NP-complete problem resolution, since it is determined by the CSPs. The identification of tractable classes of CSPs is convenient for a scalability analysis. The complexity

of resolution of CSPs depends on the number of possible solutions of the problem, and whether it is neither under-constrained nor over-constrained. For these reasons, no affirmation regarding the efficiency or scalability can be given in a generic way by our proposal, since our framework permits any type or number of data constraint states with numerical and Boolean variables, and therefore the evaluation time will depends on the specific problem.

8 Conclusions and future work

The business process combination with data objects is a recognised problem in the research community. In this paper, we go one step further towards the verification of the integration of the two models. The verification consists of the completeness and compliance of the data states associated with the process workflow, and of the discovery the states of the stored data objects. Furthermore, we provide a methodology to facilitate the automatic analysis, which is fully implemented by an extension of a commercial BPMS. In order to bring the methodology closer to the business expert, the proposed DSL provides a natural language to define the state of each class, and this description can be employed to evaluate the stored object according to the ORM that defines the database. The proposed methodology has been implemented with a set of technologies applied to a real example.

Related to this work, there are significant lines of research that can be analysed in further depth. These include: extension of the DSL to enrich the capacities to describe the data object states; simulation of the execution of the instances for the business process migration; analysis of the correct data transition in the business process model; and the inclusion of an analysis of the relation between various classes with different cardinalities in the same verification process.

Acknowledgements This work has been partially funded by the Ministry of Science and Technology of Spain (ECLIPSE project), the European Regional Development Fund (METAMORFOSIS project), and the Free University of Bozen-Bolzano (through the CRC projects KENDO and REKAP).

References

1. El-Qurna J, Yahyaoui H, Almulla M (2017) A new framework for the verification of service trust behaviors. Knowl Based Syst 121:7–22. https://doi.org/10.1016/j.knosys.2017.01.011
2. Pérez-Álvarez JM, Maté A, López MTG, Trujillo J (2018) Tactical business-process-decision support based on KPIs monitoring and validation. Comput Ind 102:23–39
3. Reichert M (2012) Process and data: two sides of the same coin? In: On the Move to Meaningful Internet Systems: OTM 2012, confederated international conferences: CoopIS, DOA–SVI, and ODBASE 2012, Rome, 10–14 September 2012. Proceedings, Part I, pp 2–19. https://doi.org/10.1007/978-3-642-33606-5_2
4. Calvanese D, De Giacomo G, Montali M (2013) Foundations of data-aware process analysis: a database theory perspective. In: Proceedings of the 32nd symposium on principles of database systems, PODS '13, ACM, New York, pp 1–12. https://doi.org/10.1145/2463664.2467796
5. Beheshti SMR, Benatallah B, Sakr S, Grigori D, Motahari-Nezhad HR, Barukh MC, Gater A, Ryu SH (2016) Process Analytics—Concepts and Techniques for Querying and Analyzing Process Data. Springer, New York. https://doi.org/10.1007/978-3-319-25037-3
6. Meyer A, Smirnov S, Weske M (2011) Data in business processes. EMISA. Forum 31(3):5–31
7. BPMN Task Force (2011) Business Process Model and Notation (BPMN) Version 2.0. Object Management Group. OMG Document Number formal, 03 January 2011
8. Weske M (2007) Business Process Management: Concepts, Languages, Architectures. Springer, New York

9. De Masellis R, Di Francescomarino C, Ghidini C, Montali M, Tessaris S (2017) Add data into business process verification: bridging the gap between theory and practice. In: Proceedings of the 31st AAAI conference on artificial intelligence (AAAI 2017). AAAI Press, San Francisco
10. Parody L, Gómez-López MT, Varela-Vaca ÁJ, Gasca RM (2018) Business process configuration according to data dependency specification. Appl Sci 8:10. https://doi.org/10.3390/app8102008
11. OMG: Object Management Group, Business Process Model and Notation (BPMN) Version 2.0. OMG Standard (2011)
12. Sun SX, Zhao JL, Nunamaker JF, Sheng ORL (2006) Formulating the data-flow perspective for business process management. Inf Syst Res 17(4):374–391
13. Hull, R.: 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, pp 1152–1163. Springer, Monterrey (2008). https://doi.org/10.1007/978-3-540-88873-4_17
14. Borrego D, Gasca RM, Gómez-López MT (2015) Automating correctness verification of artifact-centric business process models. Inf Softw Technol 62:187–197. https://doi.org/10.1016/j.infsof.2015.02.010
15. Calvanese D, Montali M, Estañol M, Teniente E (2014) Verifiable UML artifact-centric business process models. In: Proceedings of the 23rd ACM international conference on conference on information and knowledge management, CIKM 2014, Shanghai 3–7 November 2014, pp 1289–1298. https://doi.org/10.1145/2661829.2662050
16. Meyer A, Pufahl L, Batoulis K, Fahland D, Weske M (2015) Automating data exchange in process choreographies. Inf Syst 53:296–329. https://doi.org/10.1016/j.is.2015.03.008
17. Gómez-López MT, Gasca RM (2010) Run-time monitoring and auditing for business processes data using constraints. International Workshop on Business Process Intelligence, BPI 2010. Springer, Hoboken, pp 15–25
18. Weber I, Hoffmann J, Mendling J (2008) Semantic business process validation. In: 3rd international workshop on Semantic Business Process Management
19. Gómez-López MT, Gasca RM, Pérez-Álvarez JM (2014) Decision-making support for the correctness of input data at runtime in business processes. Int J Coop Inf Syst 23(2):29
20. Cabot J, Gómez C, Sancho M, Teniente E (2017) 30 years of contributions to conceptual modeling. Conceptual Modeling Perspectives. Springer, New York, pp 7–23
21. Group OM (2015) Unified modeling language reference manual, Version 2.5. OMG Standard
22. Meyer A, Pufahl L, Fahland D, Weske M (2013) Modeling and enacting complex data dependencies in business processes. In: BPM, pp 171–186
23. Gómez-López MT, Borrego D, Gasca RM (2014) Data state description for the migration to activity-centric business process model maintaining legacy databases. In: Business information systems–17th international conference, BIS 2014, Larnaca, 22–23 May 2014. Proceedings, pp 86–97. https://doi.org/10.1007/978-3-319-06695-0_8
24. Revesz P (2010) Introduction to databases
25. Gómez-López MT, Gasca RM (2014) Using constraint programming in selection operators for constraint databases. Expert Syst Appl 41(15):6773–6785. https://doi.org/10.1016/j.eswa.2014.04.047
26. Rossi F, Pv Beek, Walsh T (2006) Handbook of Constraint Programming. Foundations of Artificial Intelligence. Elsevier Science Inc., New York
27. Gómez-López MT, Gasca RM, Pérez-Álvarez JM (2015) Compliance validation and diagnosis of business data constraints in business processes at runtime. Inf Syst 48:26–43. https://doi.org/10.1016/j.is.2014.07.007
28. Dechter R (2003) Constraint Processing. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, San Francisco
29. Vennam S, Dezhgosha K (2009) Application development with object relational mapping framework-hibernate. In: International conference on internet computing, pp 166–169. CSREA Press, Athens
30. Rademakers T (2015) Activiti Documentation http://activiti.org/
31. xText Documentation (2015). https://www.eclipse.org/Xtext/
32. Prud'homme C, Fages JG, Lorca X (2014) Choco3 Documentation. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. http://www.choco-solver.org
33. Gómez-López MT, Reina-Quintero A, Gasca R (2011) Model-driven engineering for constraint database query evaluation. In: First workshop model-driven engineering, Logic and Optimization: Friends or Foes
34. Sadiq W, Orlowska M (2000) Analyzing process models using graph reduction techniques. Inf Syst 25(2):117–134
35. van der Aalst WMP, van Hee KM, ter Hofstede AHM, Sidorova N, Verbeek HMW, Voorhoeve M, Wynn MT (2011) Soundness of workflow nets: classification, decidability, and analysis. Form Asp Comput 23(3):333–363. https://doi.org/10.1007/s00165-010-0161-4

36. Eshuis R, Kumar A (2010) An integer programming based approach for verification and diagnosis of workflows. Data Knowl Eng 69:816–835
37. Sidorova N, Stahl C, Trcka N (2011) Soundness verification for conceptual workflow nets with data: early detection of errors with the most precision possible. Inf Syst 36(7):1026–1043. https://doi.org/10.1016/j.is.2011.04.004
38. Borrego D, Eshuis R, Gómez-López MT, Gasca RM (2013) Diagnosing correctness of semantic workflow models. Data Knowl Eng 87:167–184
39. UML Revision Taskforce: OMG Unified Modeling Language. Object Management Group
40. Reggio G, Ricca F, Scanniello G, Cerbo FD, Dodero G (2015) On the comprehension of workflows modeled with a precise style: results from a family of controlled experiments. Softw Syst Model 14(4):1481–1504. https://doi.org/10.1007/s10270-013-0386-9
41. Küster JM, Ryndina K, Gall H (2007) Generation of business process models for object life cycle compliance. In: Proc. BPM, pp 165–181
42. Redding G, Dumas M, ter Hofstede AHM, Iordachescu A (2008) Generating business process models from object behavior models. IS Manag 25(4):319–331. https://doi.org/10.1080/10580530802384324
43. Eshuis R, Gorp PV (2016) Synthesizing object life cycles from business process models. Softw Syst Model 15(1):281–302. https://doi.org/10.1007/s10270-014-0406-4
44. Kunchala J, Yu J, Sheng QZ, Han Y, Yongchareon S (2015) Synthesis of artifact lifecycles from activity-centric process models. In: Hallé S, Mayer W, Ghose AK, Grossmann G (eds.) 19th IEEE international enterprise distributed object computing conference, EDOC 2015, Adelaide, 21–25 September 2015, pp 29–37. IEEE Computer Society
45. Liu R, Wu FY, Kumaran S (2010) Transforming activity-centric business process models into information-centric models for soa solutions. J Database Manag 21(4):14–34
46. Meyer A, Weske M (2014) Activity-centric and artifact-centric process model roundtrip. In: Lohmann N, Song M, Wohed P (eds.) Proceedings business process management workshops 2013, Revised Papers, Lecture Notes in Business Information Processing, vol 171, pp 167–181. Springer, Berlin
47. Cruz EF, Machado RJ, Santos MY (2012) From business process modeling to data model: a systematic approach. In: Faria JP, da Silva AR, Machado RJ (eds.) 8th International conference on the quality of information and communications technology, QUATIC 2012, Lisbon, Portugal, 2–6 September 2012, Proceedings, pp 205–210. https://doi.org/10.1109/QUATIC.2012.31
48. Mayr C, Zdun U, Dustdar S (2008) Model-driven integration and management of data access objects in process-driven soas. In: Mähönen P, Pohl K, Priol T (eds) ServiceWave, vol 5377. Lecture Notes in Computer Science. Springer, Berlin, pp 62–73
49. Mayr C, Zdun U, Dustdar S (2011) View-based model-driven architecture for enhancing maintainability of data access services. Data Knowl Eng 70(9):794–819
50. Valencia-Parra A, Varela-Vaca ÁJ, López MTG, Ceravolo P (2019) Chamaleon: Framework to improve data wrangling with complex data. In: International conference on information systems ICIS 2019, Munich, 15–18 December 2019
51. Nigam A, Caswell NS (2003) Business artifacts: an approach to operational specification. IBM Syst J 42(3):428–445. https://doi.org/10.1147/sj.423.0428
52. Cohn D, Hull R (2009) Business artifacts: a data-centric approach to modeling business operations and processes. IEEE Data Eng Bull 32(3):3–9
53. Künzle V, Reichert M (2011) Philharmonicflows: towards a framework for object-aware process management. J Softw Maint 23(4):205–244
54. N, L (2011) Compliance by design for artifact-centric business processes. In: BPM 2011 LNCS vol 6896 Springer, Berlin, pp 99–115
55. Estañol M, Sancho M, Teniente E (2015) Verification and validation of UML artifact-centric business process models. CAiSE 2015:434–449
56. Sun Y, Su J, Wu B, Yang J (2014) Modeling data for business processes. In: Cruz IF, Ferrari E, Tao Y, Bertino E, Trajcevski G (eds.) IEEE 30th international conference on data engineering, Chicago, ICDE 2014, March 31–April 4, 2014, pp 1048–1059. https://doi.org/10.1109/ICDE.2014.6816722
57. Cheeseman P, Kanefsky B, Taylor WM (1991) Where the really hard problems are. In: Proceedings of the 12th international joint conference on Artificial intelligence-Vol 1, IJCAI'91, pp 331–337. Morgan Kaufmann Publishers Inc., San Francisco

José Miguel Pérez-Álvarez received the degree in computer engineering minoring in systems engineering from the Universidad de Sevilla, Spain, in 2010, the M.Sc. degree in software engineering and technology in 2011, and the Ph.D. degree from the Universidad de Sevilla in 2018. He also received a Master of Business Administration (MBA) from EOI Business School in 2013. He is Research Scientist with the Systemic AI group of Naver Labs Europe. Currently, he is also a collaborator of the IDEA Research Group. He has participated in several private and public research projects. He has published several high-impact papers.

María Teresa Gómez-López is a Lecturer at the University of Seville and the head of the IDEA Research Group. Her research areas include business processes and data management, improving the business process models including better decisions and enriching the model with data perspectives. She has led several private and public research projects and has published more than 25 papers in citation index journals and relevant conferences. She was nominated as a member of several program committees and as a reviewer of international journals. She has given keynotes or was an invited speaker at the IV Workshop on Data & Artifact Centric BPM, 5th International Workshop on Decision Mining & Modelling for Business Processes BPM, the X National Conference of BPM, the 28th IBIMA Conference, the biannual International Summer School on Fault Diagnosis of Complex Systems or Summer Program in Cybersecurity and Entrepreneurship at the University of Virginia, WISE.

Rik Eshuis works as an Assistant Professor of Information Systems at Eindhoven University of Technology, The Netherlands. He received an M.Sc. degree with distinction (1998) and a Ph.D. degree (2002) in Computer Science from the University of Twente. He has been a visiting researcher at IBM Thomas J. Watson Research Centre in New York and at CRP Henri Tudor in Luxembourg. He is and has been involved in several national and EU research projects together with industry focusing on developing advanced, flexible business process support. He was the General Chair of the IEEE European Conference on Web Services (ECOWS) in 2009 and Programme Co-chair of IEEE ECOWS 2008 and IEEE EDOC 2020. His main research interest is in business process management for knowledge-intensive processes. He is a member of the IEEE, the IEEE Computer Society, and the ACM.

Marco Montali is an Associate Professor in the Faculty of Computer Science, Free University of Bozen-Bolzano, Italy. He coordinates the PRISM research group, which devises foundational and applied techniques grounded in artificial intelligence and formal methods for the intelligent management of dynamic systems operating over data, with a specific focus on business process management and multiagent systems. On these topics, he authored more than 170 publications, many of which appeared in top-tier journals and conferences in the areas of artificial intelligence, business process management, and information systems. He is recipient of seven best paper awards. In 2015, he received the "Marco Somalvico" 2015 Prize from the Italian Association for Artificial Intelligence, as the best under-35 Italian researcher who autonomously contributed to advance the state of the art in Artificial Intelligence. He is a steering committee member of the IEEE Task Force on Process Mining.

Rafael M. Gasca received the Ph.D. degree in computer science from the Universidad de Sevilla, Spain. Since 2000, he has been leading the Quivir Research Group. Since 2015, he has been a member of the IDEA Research Group, Universidad de Sevilla. Since 2018, he has been a Full Professor. He is the leader of different public and private research projects and has directed 12 Ph.D. dissertations. He has published ten dozens of papers in high-impact-factor journals, including the IEEE COMPUTING, IEEE Communications Magazine, Information and Software Technology, Journal System and Software, Information Systems, Information and Software Technology, and Data and Knowledge Engineering. He is a reviewer for relevant security conferences and journals. He is an organizer of artificial intelligence conferences and an International Summer School on Fault Diagnosis of Complex Systems.

Affiliations

José Miguel Pérez-Álvarez[1] · María Teresa Gómez-López[1] · Rik Eshuis[2] · Marco Montali[3] · Rafael M. Gasca[1]

María Teresa Gómez-López
maytegomez@us.es

Rik Eshuis
h.eshuis@tue.nl

Marco Montali
montali@inf.unibz.it

Rafael M. Gasca
gasca@us.es

1 Universidad de Sevilla, Sevilla, Spain

2 School of Industrial Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands

3 Free University of Bozen-Bolzano, Bolzano, Italy