Integrated Exploration of Data-Intensive Business Processes

Carlo Combi, Barbara Oliboni, and Francesca Zerbato

***Abstract*—Modeling and reasoning over business processes require enterprises to manage and integrate large amounts of information. Despite process designers and engineers may benefit from a unified view of process and data models, integrating these two perspectives is challenging, especially when considering conceptual models. In this paper, we provide a uniform formal representation of a process model, the schema of a related database, and the data operations connecting them. Then, we show how we can use such a formal representation to identify interesting information during the integrated conceptual modeling and analysis of processes and related databases, from a process (re-)design and improvement perspective. Finally, we discuss the evaluation of the proposed approach through a controlled experiment and a proof-of-concept implementation that considers both relational and XML database technologies.**

***Index Terms*—Process Modeling, BPMN, Process Analysis, Conceptual Database Design, Relational Database, SQL, XML.**

I. Introduction

Business processes are widely used in industry to organize and document procedures aimed to deliver services or products to customers [1]. Modeling and executing business processes requires handling different kinds of information generated and used by process activities and, at the same time, stored and managed by (enterprise) databases [2].

Modeling is an important phase of the business process life-cycle [1]. Process models entail different inter-related perspectives, such as control-flow, data, and resources. The data perspective represents the informational entities produced or manipulated by a process and their relationships, being crucial to capture business requirements and, in turn, improve both the understanding and streamlining of process activities [3], [4].

The Business Process Model and Notation (BPMN) [5] is the standard for business process modeling, widely used for the so-called "activity-centric" processes. BPMN supports the representation of processes at different levels of abstraction [1], spanning from high-level process models needed during process design and analysis to detailed models for supporting enactment and automation. However, BPMN provides limited support for the data perspective [3], and, in practice, BPMN data objects are often used inconsistently [6].

Indeed, linking business processes and data is still an open challenge [7], [8], especially when considering conceptual models [2], [7], [9]–[11]. Connecting process models and data at the conceptual level brings many advantages: it improves the understanding of the overall process in a general, tool-agnostic way [7] and the modeling of data sources supporting process execution, and supports the collaboration among process designers and data engineers [3], as well as the detection [12], [13] and repair [14] of data flaws at design time.

C. Combi, B. Oliboni are with the Department of Computer Science, University of Verona, Italy, e-mail: {carlo.combi | barbara.oliboni@univr.it}; F. Zerbato is with the Institute of Computer Science, University of St. Gallen, Switzerland, e-mail: francesca.zerbato@unisg.ch.

Such a research issue can be seen as a facet of the more general theme of conceptually modeling both data and the related "behavioral" aspects in a holistic way. For example, such behaviors could be related to the communication needs within an information system [15]. In this case, processes and related data are observed from the viewpoint of supporting communication within complex organizations. Another behavioral aspect concerns the seamless conceptual modeling of data-intensive web applications, where process, data, and hypertext design, have to be integrated [16]. In this case, organizational processes are considered according to the web pages, data, and services supporting them. Another behavioral aspect is related to the specification of software systems by conceptually designing both the data and the related operations, and their usage within a complex organizational software system [17].

Compared to the briefly sketched research scenario, this paper is motivated by (and focuses on) the need to model and analyze activity-centric business processes and related data at design time. We focus on explicitly highlighting the connections between a process model and the conceptual schema(ta) of one or more databases storing the data relevant for its execution with the final aim to enable the human-driven exploration of the overall "connected system". To this end, we combine the BPMN [5] and UML class diagram [18] notations with formal methods, whose suitability for reasoning over processes and data in a general, tool-agnostic manner has been acknowledged by literature, e.g., in the case of logic-based conceptual description languages for the design of processes [19], [20] and data [21], [22]. We choose BPMN and UML class diagrams as they are both well-known, widely used, and sound notations for the conceptual modeling of processes and data, respectively, and their specifications are (integrated) parts of the OMG technology standards, adopted by ISO [5], [18]. Besides, we ground in a set-theoretic formal model that supports the uniform representation of processes, data, and related operations and the querying over them. In detail, we use a complex value model [23], also known as non-first normal form (NFNF) model, which allows specifying complex values for attributes, including atomic values, sets of values, or nested relations. We choose such a model since it serves as a formal reference for querying over processes and connected data without being tied to specific database technology but supports the mapping to many logical models, e.g., the object-oriented, relational, and semi-structured ones.

The contributions of this paper can be outlined as follows.

- Building upon the notion of Activity View [10], we provide a uniform, formal representation of an activity-centric process model, a conceptual database schema, and the operations performed by the process on the database.

- We show how such a formal representation supports the integrated conceptual modeling and design-time analysis of process models and the data operations performed by the processes on databases. In detail, after introducing an algorithm that captures the structure of a process model, we formalize queries that aim to support designers and analysts in reasoning over process activities and data operations and gaining useful insights for process (re-) design and improvement.
- Finally, we discuss the evaluation of our approach, consisting of (i) a controlled experiment that investigates the effects of Activity Views on process and data understanding and (ii) a proof-of-concept that shows the applicability of the proposed approach to different database management technologies. In detail, we present an implementation based on relational database technology and SQL and, then we provide an example of process representation in XML and a related XQuery expression.

The paper unfolds as follows. Section II motivates our approach with an example. Section III introduces foundational concepts. Then, Section IV presents the core contribution of our approach, which is evaluated in Section V. Section VI discusses related work and Section VII concludes the paper.

II. Motivating Example

In this section, walking through a simple order-to-cash process, we introduce some examples of information needs that capture interesting aspects of integrated process models and conceptual database schemata, and support their analysis.

Order-to-cash processes are enacted in many organizations to fulfill customer orders for goods or services [1] and encompass activities such as order verification, shipment, invoicing, and payment. These activities are often performed with the help of a BPM system that interacts with one or more database systems to manage the relevant information.

Figure 1 (left) shows a simplified BPMN process model for managing a purchase order placed by a customer and (right) the UML class diagram representing the conceptual schema of the database supporting the process. The process starts when a purchase order is received from a customer (start event s). The first task Manage order (A) represents order reception and verification, and determines whether the order will be "accepted" or "declined". The data needed by task A are listed in the related text annotation. Then, exclusive gateway G1, enclosing symbol × and labeled Are order details correct?, splits the control flow into two alternative paths. If errors are detected or the ordered items are out of stock, the vendor must Decline order (B). Otherwise, the order is accepted. Upon checking customer details (task C), the vendor runs multiple activities in parallel, which are enclosed by gateways G2 and G5 depicted with symbol +. First, items are procured (task D) and the order status is updated to "in process". Then, items are packaged (task E) with the help of a warehouse checklist, shown as a data object. Meanwhile, the vendor may decide to Offer discount (F). Then, the invoice is issued and sent to the customer (task H). The vendor waits to Verify payment (I) before shipping the products (task J). Finally, new offers are sent to the customer (task K) based on the purchase history, if available, and the process ends (end event e).

To be properly executed, the process in Figure 1 (left) needs to access the purchase database (data store DB) whose conceptual schema is outlined in Figure 1 (right) and contains classes corresponding to invoices, customers, orders, items, and so on, together with the required associations. Figure 1 remarks the lack of a conceptual integration between activity-centric process models and conceptual database schemata representing the information needed for process execution [2], [9], [10], which, in turn, undermines the capability of jointly analyzing and re-designing them to keep up with business and market changes [24]. Indeed, especially when dealing with large and complex process models, it is not easy to grasp which information entities are needed to support the process and how they are used along the process flow [4].

For example, by looking at Figure 1, we can see that some process activities need to access the same kind of information (e.g., both tasks B and F concern communication with the customer, i.e., information represented by classes Customer, Order and Communication), and that some activities do not access the database (e.g., task E). However, it is not possible to understand how the process accesses these data, nor can we see whether they are needed at later stages. Indeed, activities may require data that are generated or obtained in previous steps of the process, e.g., the kind and quantity of ordered items obtained during task A are necessary to execute task D.

Starting from the exemplified scenario, we focus on some information needs that are quite common in software design when considering both processes and data. Such information needs (I1–I4) were elicited by considering both the structure of process models (i.e., alternative and parallel flows) and the kinds of data access operations (e.g., CRUD operations) that are typically performed by process activities on a database. These are just some possible examples of the needs that can be managed through our proposal in a real-world software and requirements engineering context [4], [24]–[26].

Information Need I1 – Identical data operation signatures. Understanding which activities are characterized by the same kind of data operations is particularly relevant for enhancing re-use of process fragments. Re-use can entail the definition of reusable elements during process (re-)design (e.g., global process elements such as *call activities* [5]) but can also concern the definition of data access permissions to users. Indeed, resources may be associated with roles based on their authorization to read or write specific data or use associated client applications [4]. Moreover, identifying activities with the same signature in terms of data operations can help understand possible data-related interactions among activities.

As an example, let us consider tasks B and F. Despite being completely different activities from a process perspective, from a database point of view they both involve communication with the customer, which is realized through the same operations on the database: to communicate with a certain customer, the vendor must access the customer's email address and the order

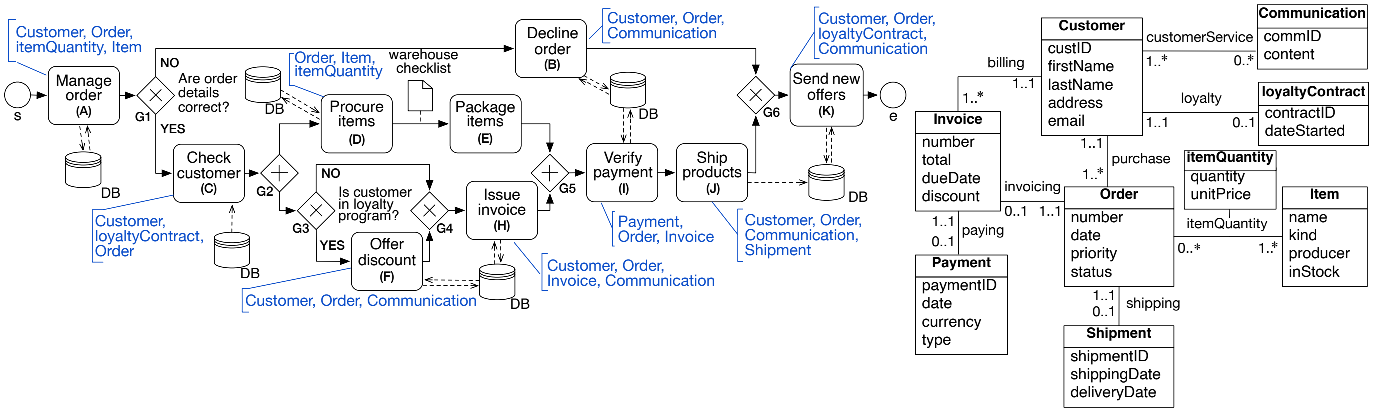


Figure 1. (Left) BPMN process model showing the main steps of a generic purchase order process. Text annotations, depicted in blue, describe the data needed by each activity, whose short name is enclosed in parentheses. (Right) UML class diagram showing the conceptual schema of a purchase order database.

number. Knowing which activities have access to the same data is useful for several reasons, e.g., for (i) improving the compliance of the process with customer communication best practices or (ii) managing database access permissions of the procurement team and staff accountants.

Information Need I2 – Use of data across process paths. I2 evaluates whether activities located on alternative process paths require the same data to be executed. Alternative paths often capture different business settings and executions. However, activities lying on alternative paths may need to access the same data. During process re-engineering, this information may help to re-arrange activities and improve data access (e.g., an activity dedicated solely to information gathering may be moved before the exclusive gateway). From a data perspective, knowing which data are needed by activities across process paths is useful to identify information that is central to the whole process, regardless of which is the preferred flow.

For example, since both tasks B and C require information about customers to communicate that the order is declined or to check the details of an accepted order, persistent instances (hereinafter, objects) of class Customer are used in every process execution. Instead, information about invoicing and shipping is not needed if the order is declined.

Information Need I3 – Use of data along process paths. I3 explores how data is used by activities located along a specific process path. While some activities create data that are used later in the process, other activities may perform operations that are superfluous or may lead to data inconsistencies [12], [13]. Thus, discovering data reading and writing patterns may help designers to get rid of redundant data access operations, to add new ones, or resolve inconsistencies.

For example, designers may be interested in knowing which is the last activity of the process that updates objects of class Order to ensure that the information provided to the customer is up-to-date and not modified afterward. In Figure 1, the last activity to access objects of class Order is either task J if the order is accepted or task A, i.e., the first of the process, if the order is declined.

Information Need I4 – Concurrent data access. I4 checks if activities located on parallel paths require access to the same data. From the point of view of the process, it helps identify activities that do not have concurrent data access and may be parallelized. Instead, from the point of view of the database, it identifies potentially concurrent operations that may require careful transactional control or additional constraints. For example, tasks D and H have concurrent access to instances of class Order and both activities update the status of the order, either to "in process" or to "invoice sent". Besides, we can consider concurrent data access from many different process models to understand which parts of the given database(s) are most commonly accessed by process activities and to optimize and tune, for example, the transaction isolation levels.

We would like to note that the presented list of information needs is not complete and can be extended with other aspects that are interesting for the design-time analysis of integrated processes and data (e.g., discovering data that are never used by a process or data authorization constraints [4]).

III. Foundations

This section introduces the basic concepts of process and data modeling that will be encountered throughout the paper.

Starting from a significant subset of BPMN [5] as done in [3], [13], we give the following definition of process model, focusing on control- and data-flow elements used in this paper.

Definition 1 (**Process Model**)**.** A process model m is a tuple $m = (N, C, DN, F, P_\ell, \mathcal{P})$ consisting of a finite non-empty set of flow nodes N, a finite non-empty set C of control flow edges, a finite set DN of data nodes, a finite set F of data associations, a function P_ℓ associating proposition literals to branching edges, and a set of propositions $\mathcal{P}$. The set $N = Ac \cup G \cup E$ of flow nodes consists of the disjoint sets Ac of activities, G of gateways, and $E = \{s, e\}$ of start and end events. The set $G = G^x_s \cup G^x_m \cup G^a_s \cup G^a_m$ is partitioned according to the routing behavior of its nodes into the disjoint sets G^x_s of *xor split* nodes, G^x_m of *xor merge* nodes, G^a_s of *and split* nodes, and G^a_m of *and merge* nodes, respectively. The control flow $C \subseteq N \times N$ connects the elements of N. Given a flow node $n \in N$, $\bullet n \subseteq N$ ($n \bullet \subseteq N$) denotes the set of direct predecessor (successor) nodes of n. $DN = DO \cup DS$ is the set of data nodes, consisting of the disjoint sets DO of data objects and DS of data stores. $F \subseteq (DN \times Ac) \cup (Ac \times DN)$ is the data flow that connects data nodes with activities. Ultimately, function $P_\ell : (C \cap (G_s^x \times N)) \to \mathcal{P}^*$, where $\mathcal{P}^*$ is the set of literals that we can derive from propositions in $\mathcal{P}$, assigns two mutually exclusive proposition literals to edges outgoing from a xor split node, respectively.

Without loss of generality, we assume to deal with well-structured process models [27], [28]. Also, we consider process models having exactly two flows outgoing (incoming) from (to) xor split (merge) gateways, i.e., $|g\bullet|$=2 ($|\bullet g|$=2) for $g \in G_s^x$ (G_m^x), as this simplifies the encoding of process paths. Indeed, a gateway with x outgoing (incoming) flows may always be expressed as $x{-}1$ cascading binary gateways.

Besides, given a xor split gateway g, the two outgoing edges are labeled with some literals p and $\neg p$, respectively. Thus, we may say that proposition p is implicitly associated to g.

Definition 2 (Database schema). A database schema $\mathcal{DS}$ is a tuple $\mathcal{DS} = (Cl, As, Att, \mathcal{A}, \mathcal{I}, \mathcal{C_A})$ where Cl is a finite non-empty set of classes, As is a finite set of associations between classes of Cl, Att is a finite set of attributes, $\mathcal{A} : Cl \to 2^{Att}$ and $\mathcal{C_A} : Cl \to As$ are functions mapping classes to their attributes and to the corresponding associations (if any), respectively. The set $AsC \subset Cl$ is composed by association classes c, having $\mathcal{C_A}(c) \neq \varnothing$. The set $As \subset AsN \times Cl \times m \times M \times Cl \times m \times M$ represents associations. Each association has a name uniquely identifying it from set AsN, two associated classes[1] and their related multiplicities, respectively. A multiplicity $(min..max)$, where $min \in m$ and $max \in M$, denotes the minimum and maximum number of objects of that class that can participate in the association. The most common multiplicity values are 0, 1, or $*$, where symbol $*$ is used for representing no maximum limit on participation. With the notation $c_j(attr_1, \ldots, attr_n)$, where $c_j \in C_{set_i}$, we specify in a compact way that $\mathcal{A}(c_j) = \{attr_1, \ldots, attr_n\}$. The set $\mathcal{I} \subset Cl \times Cl$ represents the inheritance relationships: $(c_i, c_j) \in \mathcal{I}$ when class c_i inherits from class c_j.

Activity Views have been introduced in [10] as a possible approach for representing the connection between a process model and a conceptual database schema. In short, an Activity View shows which parts of a database schema are related to data that are accessed by a certain process activity and which data operations are performed on it. Below, we introduce the concept of Activity View, by refining the definition given in [10].

Definition 3 (Activity View). Let us consider a process model $m = (N, C, DN, F, P_\ell, \mathcal{P})$, and a database schema $\mathcal{DS} = (Cl, As, Att, \mathcal{A}, \mathcal{I}, \mathcal{C_A})$ corresponding to a data store $ds \in DS \subseteq DN$. Given an activity $ac \in Ac \subseteq N$, its *Activity View* $av_{ac} = \{t_1, \ldots, t_n\}$ is a possibly empty set[2] of tuples $t_1, \ldots, t_n$, where each tuple t_i denotes a particular data access operation performed by ac on data of a given database schema $\mathcal{DS}$.

[1]For the sake of simplicity, we will consider here only binary associations.
[2]We represent data access operations as a set as the same activity can imply the execution of different queries in many possible orders.

Each tuple of the Activity View has the form $t_i = \langle C_{set_i}$, A_{set_i}, *AccessType*$_i$, *AccessTime*$_i$, *NumInstances*$_i\rangle$, where:

- $C_{set_i} = \{c_1, \ldots, c_j\} \subseteq Cl$ is the set of connected classes, having instances accessed by ac. Classes $c_j, c_h \in C_{set_i}$ are connected if they both are at the extremes of an association a_f. If a_f is connected also to association class $c_f \in AsC$ (i.e., $\mathcal{C_A}(c_f) = a_f$), then c_f must also belong to C_{set_i}. If a class $c_j \in C_{set_i}$ specializes class c_l, then it is sufficient that c_l is one end of a_f for c_j to be considered connected to other classes of C_{set_i}. When all the attributes of c_j are involved in the data operation, we write $c_j(*)$, otherwise, we specify them as $c_j(attr_g, \ldots, attr_m)$ with $\{attr_g, \ldots, attr_m\} \subset \mathcal{A}(c_j)$.
- $A_{set_i} = \{a_1, \ldots, a_r\} \subseteq AsN$ is a set of binary associations, identified through their names, that directly link any two classes of C_{set_i}.
- $AccessType_i \in \{$*R, I, D, U*$\}$ defines the type of access to the related information. *R* denotes a *read* of elements of C_{set}, whereas *I*, *D*, and *U* respectively denote an *insertion*, a *deletion*, and an *update* operation.
- $AccessTime_i \in \{$*start, during, end*$\}$ denotes when a data operation is performed w.r.t. activity execution.
- $NumInstances_i = (min, max)$, where $min \in \{0, 1\}$ and $max \in \{1, *\}$, denotes the number of objects the operation focuses on. The value of max is set to 1 if the operation selects at most a single object of (at least) a class or to $*$ if the operation may select many objects of any class. The value of min is set to 1 when the operation will use for sure at least one object of any class, or to 0 if the operation could not use any object of a class.

Below, we show the Activity Views representing the operations performed by the process in Figure 1 on the database.

To check an order, the vendor verifies if the order details are correct and all the requested items are in stock. Then, the status of the order is updated to “accepted” or “declined”. These two data access operations correspond to the two tuples of the following Activity View enclosed in angular brackets.

$$\begin{aligned} av_A = \{&\langle\{Customer(*), Order(*), ItemQuantity(*), Item(*)\},\\ &\{purchase, itemQuantity\}, R, start, (1,1)\rangle,\\ &\langle\{Order(status)\}, \varnothing, U, during, (1,1)\rangle\}\end{aligned}$$

In this case, the number of instances involved in both data operations is $(1, 1)$ as single objects of classes *Customer* and *Order* need to be mandatorily accessed.

If the order is declined, the customer must be immediately informed and all the communications are recorded in the purchase database. When a discount is applied and communicated to the customer the Activity View is the same.

$$\begin{aligned} av_B = \{&\langle\{Customer(*), Order(*)\}, \{purchase\}, R, start, (1,1)\rangle,\\ &\langle\{Communication(*)\}, \varnothing, I, end\ (1,1)\rangle\}\end{aligned}$$

Instead, if the order is accepted, customer details related to loyalty contracts are checked. Only for loyalty customers, purchase history is also reviewed to possibly offer discounts.

$$\begin{aligned} av_C = \{&\langle\{Customer(*), loyaltyContract(*)\}, \{loyalty\}, R,\\ &during, (1,1)\rangle, \langle\{Customer(custID), Order(*)\},\\ &\{purchase\}, R, during, (0,*)\rangle\}\end{aligned}$$

In this case, the number of instances involved in the last data operation is $(0, *)$. Indeed, such operation may not find any object related to the purchase history but, in general, such purchase history can be composed by many objects.

To procure items the warehouse staff needs to access the details about the kind and quantity of the ordered items and, then, updates the status of the order to “in process”.

$av_D = \{\langle\{Order(number, priority), ItemQuantity(*), Item(*)\}, \{itemQuantity\},\ R,\ start\ (1,*)\rangle,\ \langle\{Order(status)\},\ \varnothing,\ U,\ end,\ (1,1)\rangle\}$

To issue the invoice order and delivery data are reviewed prior to creating the invoice and sending it to the customer. Then, the status of the order is updated to “invoiced”.

$av_H = \{\langle\{Customer(*), Order(*)\}, \{purchase\}, R, start, (1,1)\rangle, \langle\{Invoice(*)\}, \varnothing, I, during, (1,1)\rangle, \langle\{Communication(*)\}, \varnothing, I,\ end, (1,1)\rangle, \langle\{Order(status)\}, \varnothing, U, end, (1,1)\rangle\}$

To verify a payment, the vendor compares the received amount with the invoice. Then, payment details are inserted into the database and the order status is updated to “paid”.

$av_I = \{\langle\{Invoice(*)\}, \varnothing, R, start, (1,1)\rangle, \langle\{Payment(*)\}, \varnothing, I, during, (1,1)\rangle,\ \langle\{Order(status)\}, \varnothing, U, end, (1,1)\rangle\}$

Product shipping implies adding shipping information to the database and communicating the tracking details to the customer. The status of the order is then updated to “shipped”.

$av_J = \{\langle\{Shipment(shipmentID, shippingDate)\}, \varnothing, I, start, (1,1)\rangle, \langle\{Communication(*)\}, \varnothing, I, start, (1,1)\rangle, \langle\{Order(status)\}, \varnothing, U, during, (1,1)\rangle\}$

Finally, new offers are sent to the customer, based on the purchase history. Depending on the period of the year, loyalty customers may receive additional offers.

$av_K = \{\langle\{Customer(custID), Order(*)\}, \{purchase\}, R, start, (1,*)\rangle, \langle\{Customer(*), loyaltyContract(*)\}, \{loyalty\}, R, start, (0,1)\rangle, \langle\{Communication(*)\}, \varnothing, I, end, (1,1)\rangle\}$

IV. An Approach to Analyze Connected Models

To encode the information related to process models, database schemata, and their connections realized through Activity Views, we rely on a complex value model [23], [29] and design a complex value schema that collects and integrates such information. The complex value model allows us to specify complex values for attributes, i.e., values with a hierarchical structure, such as sets of values or nested relations. In this work, we use complex values for representing sets of elements, e.g., the class and association sets of Activity Views.

Complex values are defined through the use of set constructors[3]. For example, given a relation Class, the notation {Class} represents the domain of an attribute that can have instances of Class as its values. The proposed set-theoretic complex value schema is shown in Figure 2. Attributes that denote primary keys are underlined, while symbol $*$ is used to denote attributes that can be NULL. For example, avID is NULL when activities do not have an Activity View (e.g., task E in Figure 1). In Figure 2, complex value relation names are in upper camel case, attribute names are in lower camel case and inclusion dependency constraints are listed at the bottom.

[3]The notation for the other constructor *tuple* is the standard one for relational schemata extended to represent attributes with complex values, i.e., RelationName(attribute1, $\cdots$, attributeM:{attribute0} attributeN: {Relation1})

Complex Value Relations

Business Process Models

1) Process(<u>processID</u>, name, documentation)
2) DataNode(<u>dnID</u>, processID)
3) DataStore(<u>dnID</u>, dsName, capacity)
4) DataObject(<u>dnID</u>, name, isCollection)
5) DataFlow(<u>fnID, dnID</u>, direction)
6) FlowNode(<u>fnID</u>, processID)
7) ControlFlow(<u>from, to</u>, hasLiteral, literal$*$)
8) Activity(<u>fnID</u>, activityName, isReusable, isAtomic, type, avID$*$, label:{literal})
9) Gateway(<u>fnID</u>, gatewayName, routingType, splitType, hasCondition, condition$*$)
10) Event(<u>fnID</u>, eventName, type, position, triggerType)
11) Succ(<u>fnID</u>, successors:{Activity})

Database Schemata

12) DomainDB(<u>dbID</u>, dbName)
13) ClassDB(<u>className, dbID</u>, isAssociationClass, assocNameRef*, assocDBref*)
14) AssociationDB(<u>assocName, dbID</u>, leftC, minL, maxL, rightC, minR, maxR)
15) AttributeDB(<u>attrName, className, dbID</u>, type)
16) InheritDB(<u>classNameS</u>, classNameG)

Activity Views

17) ActivityView(<u>avID</u>, operations:{DataOp})
18) DataOp(<u>opID</u>, cSet:{ClassDB}, aSet:{AssociationDB}$*$, accessType, accessTime, minInstances, maxInstances)
19) AttDataOp(<u>opID</u>, cAttSet:{AttributeDB})

Inclusion Dependency Constraints

- DataNode[processID] $\subseteq$ Process[processID];
- DataStore[dnID] $\subseteq$ DataNode[dnID];
- DataObject[dnID] $\subseteq$ DataNode[dnID];
- FlowNode[processID] $\subseteq$ Process[processID];
- ControlFlow[from] $\subseteq$ FlowNode[fnID];
- ControlFlow[to] $\subseteq$ FlowNode[fnID];
- Activity[fnID] $\subseteq$ FlowNode[fnID];
- Activity[avID] $\subseteq$ ActivityView[avID];
- Gateway[fnID] $\subseteq$ FlowNode[fnID];
- Event[fnID] $\subseteq$ FlowNode[fnID];
- Succ[fnID] $\subseteq$ FlowNode[fnID];
- ActivityView[processID] $\subseteq$ Process[processID];
- ActivityView[operations] $\subseteq$ DataOp;
- AssociationDB[dbID] $\subseteq$ DomainDB[dbID];
- ClassDB[dbID] $\subseteq$ DomainDB[dbID];
- AttrDB[className, dbID] $\subseteq$ ClassDB[className, dbID];
- Succ[successors] $\subseteq$ Activity;
- DataOp[cSet] $\subseteq$ ClassDB;
- DataOp[aSet] $\subseteq$ AssociationDB;
- AttDataOp[cAttSet] $\subseteq$ AttributeDB;
- AttrDB[className, dbID] $\subseteq$ ClassDB[className, dbID]
- ClassDB[assocNameRef, assocDBref] $\subseteq$ AssociationDB[assocName, dbID] *if ClassDB[IsAssociationClass] is true*

Figure 2. Complex value schema of the repository storing information about a process model, a conceptual database schema, and related data operations.

Complex value relations 1–11 (hereinafter *relations*) represent the elements of BPMN process models (cf. Definition 1). Since relations Activity, Gateway, and Event specialize FlowNode, and relations DataObject and DataStore specialize DataNode, we used the same primary keys. Relation Succ and complex attribute label of Activity store information related to the order and position of the activities along the process flow. Relations 12–16 represent database schemata (cf. Definition 2). Finally, relations ActivityView and DataOp denote the data operations performed on databases (cf. Definition 3).

Some attributes are represented as complex (set) values: label of relation Activity, as each label can be seen as a set of literals (cf. Section IV-A); successors of relation Succ, as each activity is associated to a (possibly empty) set of successors; and attributes operations, cSet, and aSet, and cAttSet of relations ActivityView, DataOp, and AttDataOp, respectively.

A. *Unravelling Process Paths with a Labeling Approach*

To address information needs I1–I4 it is essential to establish whether any two activities are executed sequentially, in parallel, or alternatively, i.e., on which process path(s) they lie and we need to keep track of their successors along those paths. Since process models have a graph-like structure, information about process paths can be retrieved by exploring flow nodes and edges. However, since not all process executions include the same set of activities and the number of alternative process paths is exponential w.r.t. the number of xor split nodes, obtaining a compact encoding of the process structure is challenging. To tackle this problem, we draw inspiration from [30] and associate *labels* with process activities.

Definition 4 (Label). Given a set $\mathcal{P}$ of propositional letters, a label is a (possibly empty) conjunction of positive or negative literals from $\mathcal{P}^*$. The empty label is notated as $\boxdot$ and is an identity for label conjunction.

As an example, let $\mathcal{P} = \{p, q, r\}$. A label consistent with Definition 4 is $L = p \wedge q \wedge \neg r$. For convenience hereinafter we notate logical conjunctions such as '$p \wedge q \wedge \neg r$' as '$pq\neg r$'. As explained later, labels are associated to activities and encode their position along process paths. As an example, in Figure 3 activities B and C are labeled with $\neg p$ respectively p, since they lie on the alternative paths originating from gateway G1, which is associated to propositional letter p.

Algorithm 1 shows the pseudocode of a recursive procedure that traverses the process as a depth-first search and populates relation Succ and attribute label of relation Activity of the schema in Figure 2. Algorithm 1 takes as input a process model $m = (N, C, DN, F, P_\ell, \mathcal{P})$, a node $n \in N$, and a label L managed as a stack of literals. S is a global associative array that records the set of successor activities of each node.

From the start event, the procedure traverses each process path recursively[4]. At every forward step, the procedure keeps track of the current label L, while at every backward step, it updates the set S of successor activities of each node.

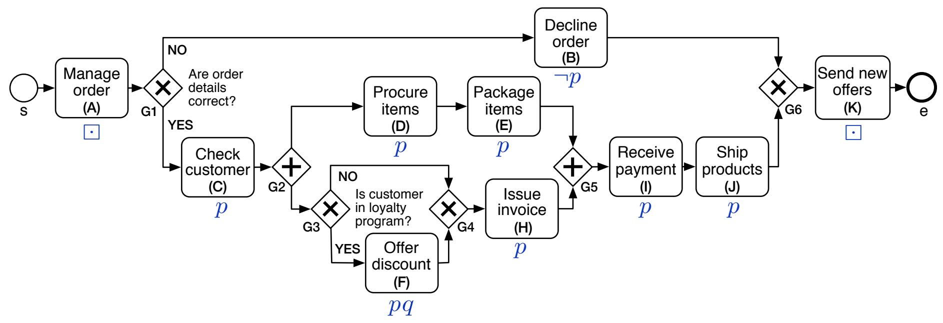
The initial call of Algorithm 1 is traverseProcess($m, s, \{\boxdot\}$). Each conditional statement evaluates the current node n and proceeds as follows, based on the kind of n.

[4]Algorithm 1 does not explicitly deal with (finite) loops. Loops in the process model must first be unfolded in many disjoint paths, each corresponding to the execution of the loop with a different number of iterations.

Algorithm 1: traverseProcess

```
Input: process model $m = (N, C, DN, F, P_\ell, \mathcal{P})$, $n \in N$, label $L$.
Result: populates field label of Activity and relation Succ.
1  if ($n \in Ac$) then // Activities
2      Activity(fnID, label) $\leftarrow (n, L)$
3      $n' \leftarrow n\bullet$
4      if $S(n') ==$ NIL then
5          traverseProcess($m$, $n'$, $L$)
6      Succ(fnID, successors) $\leftarrow (n, S(n'))$
7      $S(n) \leftarrow S(n') \cup \{n\}$
8  else if $n \in G_s^x \cup G_s^a$ then // Split gateways
9      $n' \leftarrow$ FIRST($n\bullet$)
10     $n'' \leftarrow$ SECOND($n\bullet$)
11     if ($S(n') ==$ NIL $\wedge$ $S(n'') ==$ NIL) then
12         if $n \in G_s^x$ then
13             $\ell \leftarrow P_\ell(n, n')$
14             $\bar{\ell} \leftarrow P_\ell(n, n'')$
15             if $L$.PEEK() = '$\boxdot$' then
16                 $L$.POP()
17             traverseProcess($m$, $n'$, $L$.PUSH($\ell$))
18             traverseProcess($m$, $n''$, $L$.PUSH($\bar{\ell}$))
19         if $n \in G_s^a$ then
20             traverseProcess($m$, $n'$, $L$)
21             traverseProcess($m$, $n''$, $L$)
22     $S(n) \leftarrow S(n') \cup S(n'')$
23 else if $n \in G_m^x \cup G_m^a$ then // Merge gateways
24     $n' \leftarrow n\bullet$
25     if $S(n') ==$ NIL then
26         if $n \in G_m^x$ then
27             traverseProcess($m$, $n'$, $L$.POP())
28         if $n \in G_m^a$ then
29             traverseProcess($m$, $n'$, $L$)
30     $S(n) \leftarrow S(n')$
31 else if $n == s$ then // Start event
32     $n' \leftarrow n\bullet$
33     traverseProcess($m$, $n'$, $L$)
34 else
35     $S(n) = \varnothing$
```

- Activities (lines 1–7). The label L of n is stored in the database (line 2). Then, Algorithm 1 finds the direct successor of n, i.e., n', from the singleton $n\bullet$ (line 3). If the successors of n' are unknown (line 4), then a recursive call is made to discover them (line 5). Once all the successors of n have been discovered, the set $S(n')$ of successors is added to the database (line 6) and n is added to $S(n)$ as it is the latest visited activity (line 7).
- Split gateways (lines 8–22). Split gateways require dealing with two successor nodes at once. Functions FIRST() and SECOND() extract the first and second element of $n\bullet$, i.e., n' and n'' (lines 9–10). If the sets of successors $S(n')$ and $S(n'')$ are unknown, then two recursive calls are made to discover them. If the gateway is exclusive, i.e., $n \in G_s^x$ (line 12), the proposition literals of the two outgoing edges are assigned to variables ℓ and $\bar{\ell}$ (lines 13–14) and the two recursive calls are made after updating the label L by appending ℓ and $\bar{\ell}$ (lines 17–18). If the gateway is parallel, i.e., $n \in G_s^a$ (line 19), the two recursive calls on n' and n'' are made considering the original label L (lines 20–21). Then, $S(n)$ is updated with the successors



Activity

fnID	...	label
A	...	$\boxdot$
B	...	$\neg p$
C	...	p
D	...	p
E	...	p
F	...	pq
H	...	p
I	...	p
J	...	p
K	...	$\boxdot$

Succ

fnID	successors
A	{B,C,D,E,F,H,I,J,K}
B	{K}
C	{D,E,F,H,I,J,K}
D	{E,I,J,K}
E	{I,J,K}
F	{H,I,J,K}
H	{I,J,K}
I	{J,K}
J	{K}
K	{ }

Figure 3. The process of Figure 1 labeled by applying Algorithm 1 (left) and updated complex value relations (right). For simplicity, activity short names and database identifiers are assumed to coincide. Activities labeled with the empty label '$\boxdot$' belong to paths that are traversed by every process execution.

found along both outgoing paths (line 22).

- Merge gateways (lines 23–30). Whenever a merge gateway is encountered (line 23), if the set $S(n')$ is undefined then a recursive call is made. If the gateway is exclusive (lines 26–27), the latest added literal is also removed from the current label. Then, $S(n)$ is updated accordingly.
- Start event (lines 31–33). From the start event, the algorithm makes a recursive call to visit the following node.
- End event (lines 34–35). When the end event is reached, then $S(n)$ is set to empty[5].

Figure 3 shows the output of Algorithm 1 for the process in Figure 1. Attribute successors of relation Succ is the set of all the successors of one activity in the process.

The labels recorded in attribute label of Activity can be combined with the information about successors to characterize the relative position of any two activities. Given any two activities a_1 and a_2, respectively labeled L_1 and L_2, then:

(1) if $a_1 \in$ successors of a_2 or $a_2 \in$ successors of a_1, then a_1 and a_2 are in sequential order;
(2) if condition (1) does not hold (i.e., $a_1 \notin$ successors of a_2 and $a_2 \notin$ successors of a_1) and there is no propositional letter ℓ such that $\ell \in L_1$ and $\neg\ell \in L_2$, then a_1 and a_2 belong to parallel paths;
(3) if there exists at least one propositional letter ℓ such that $\ell \in L_1$ and $\neg\ell \in L_2$, then a_1 and a_2 belong to alternative paths.

For example, from Figure 3 we can evince that activities B and F are located on alternative paths since their labels, $\neg p$ and pq, include the opposite literals p and $\neg p$. Instead, tasks D and F are executed in parallel as they are not successors of one another and their labels p and pq do not include literals in opposition. It is worth noting that Algorithm 1 labels with '$\boxdot$' all the activities belonging to paths that are always executed.

The computational complexity of Algorithm 1 is basically the one of a recursive depth-first search algorithm. Each recursive call requires a constant number of steps to be executed, as functions FIRST(), SECOND(), NEWPROPOSITION(), PUSH(), PEEK(), and POP() can be implemented in constant time. Since Algorithm 1 visits all the nodes in m only once, the number of recursive calls needed to traverse the process is linear in the number of nodes in m (i.e., its computational complexity is bound by $\mathcal{O}(|N|)$). Algorithm 1 is complete as all kinds of flow nodes described in Definition 1 are considered and the procedure always deals with all the immediate successors of one node, i.e., at most two in case of split gateways.

[5]In the algorithm and the following queries, we assume opposite literals to be unique to each pair of xor outgoing edges without losing generality. Finding successors of each node in case of repeating literals is straightforward by considering both the literals and activity successors derived by the algorithm.

B. *Querying Connected Process and Database Models*

This section shows how our approach can be used to explore the data perspective of a process by addressing information needs I1–I4. We focus on properties that cannot be easily observed on a process model alone but originate from its connection with a conceptual database schema and can provide useful hints for process re-design and improvement.

To formalize examples of the queries that can be run against the schema of Figure 2, we choose a complex value calculus [23]. This many-sorted calculus extends the relational calculus with complex-valued attributes, thus being suitable to deal with models including complex structures, such as the one proposed in this work. Indeed, this formalism includes set and tuple variables, and supports quantification over them. Besides, it features three binary predicates: '$=$' (equality), '$\in$' (membership), and '$\subseteq$' (set containment) that allow us to express conditions on sets of elements, e.g., "element e belongs to set A". For tuple comparison, we rely on the well-known principle of deep equality [31].

Below, we introduce some shortcuts for formulas that we will use to formalize queries Q1–Q6 that exemplify possible ways of addressing information needs I1–I4 with our approach.

- Given activity a and data operation do, *DataOpOf*(a,do) checks if do belong to the Activity View of a:
  DataOpOf(a,do) $\equiv$ (Activity(a) $\wedge$ DataOp(do) $\wedge$ $\exists$av (ActivityView(av) $\wedge$ av.avID = a.avID $\wedge$ do $\in$ av.operations)).
- Given two activities a and a', *SuccessorOf*(a,a') checks if a' is a successor of a:
  SuccessorOf(a,a') $\equiv$ (Activity(a) $\wedge$ Activity(a') $\wedge$
  $\exists$s (Succ(s) $\wedge$ s.fnID = a.fnID $\wedge$ a' $\in$ s.successors)).
- Given an activity a and a process p, *ActOf*(a,p) checks if a is an activity of process p:
  ActOf(a,p) $\equiv$ (Activity(a) $\wedge$ Process(p) $\wedge$ $\exists$n (FlowNode(n) $\wedge$ n.fnID = a.fnID $\wedge$ n.processID = p.processID)).

Query Q1 focuses on identifying activities lying on alternative process paths but having identical data operation signatures, thus addressing both information needs I1 and I2.

Q1: Which pairs of activities lying on alternative process paths have the same Activity View?

$$\{\mathsf{a, a'} \mid \exists\ \mathsf{p}\ (\mathit{ActOf}(\mathsf{a,p}) \wedge \mathit{ActOf}(\mathsf{a',p})) \wedge \exists\ \mathsf{l}\ (\mathsf{l} \in \mathsf{a.label} \wedge \neg \mathsf{l} \in \mathsf{a'.label}) \wedge \forall \mathsf{do}\ (\mathit{DataOpOf}(\mathsf{a,do}) \rightarrow \exists\ \mathsf{doa}\ (\mathit{DataOpOf}(\mathsf{a',doa}) \wedge \mathsf{do.cSet} = \mathsf{doa.cSet} \wedge \mathsf{do.aSet} = \mathsf{doa.aSet} \wedge \mathsf{do.accessType} = \mathsf{doa.accessType} \wedge \mathsf{do.accessTime} = \mathsf{doa.accessTime} \wedge \mathsf{do.numInstances} = \mathsf{doa.numInstances})) \wedge \forall \mathsf{do'}(\mathit{DataOpOf}(\mathsf{a',do'}) \rightarrow \exists\ \mathsf{dob}\ (\mathit{DataOpOf}(\mathsf{a,dob}) \wedge \mathsf{do'.cSet} = \mathsf{dob.cSet} \wedge \mathsf{do'.aSet} = \mathsf{dob.aSet} \wedge \mathsf{do'.accessType} = \mathsf{dob.accessType} \wedge \mathsf{do'.accessTime} = \mathsf{dob.accessTime} \wedge \mathsf{do'.numInstances} = \mathsf{dob.numInstances}))\}$$

When applied to the process in Figure 1, Q1 returns tasks B and F, as they both read information about purchases and insert the messages sent to the customer into the database.

Query Q1 may be tuned to compare Activity Views, so that (i) all the attributes of all their tuples coincide, (ii) only some tuples coincide, or (iii) only some attributes of some tuples coincide. In the context of well-structured processes, process elements or blocks may be repeated within alternative paths to maintain the correct nesting of Single-Entry-Single-Exit regions [28]. Here, Q1 can be useful to improve the identification of re-usable process activities from a data perspective.

Queries Q2 and Q3 are both examples of addressing I3 as they consider (successive) activities along process paths, focusing on which data operations are performed and when.

Q2: Which are the last activities of a process p reading objects of a certain (data) class x?

$$\{\mathsf{a} \mid \exists\ \mathsf{do, c}\ (\mathit{ActOf}(\mathsf{a,p}) \wedge \mathit{DataOpOf}(\mathsf{a,do}) \wedge \mathsf{ClassDB(c)} \wedge \mathsf{p.name} = `p' \wedge \mathsf{c.className} = `x' \wedge \mathsf{c} \in \mathsf{do.cSet} \wedge \mathsf{do.accessType} = `\mathrm{R}' \wedge \nexists \mathsf{a',do'}\ (\mathit{DataOpOf}(\mathsf{a',do'}) \wedge \mathit{SuccessorOf}(\mathsf{a,a'}) \wedge \mathsf{c} \in \mathsf{do'.cSet} \wedge \mathsf{do'.accessType} = `\mathrm{R}'))\}$$

Below we show Q2 for the process *Purchase order* of Figure 1, considering the last activities that read class *Invoice*.

$$\{\mathsf{a} \mid \exists\ \mathsf{p, do, c}\ (\mathit{ActOf}(\mathsf{a,p}) \wedge \mathit{DataOpOf}(\mathsf{a,do}) \wedge \mathsf{ClassDB(c)} \wedge \mathsf{p.name} = `\mathit{Purchase\ order}' \wedge \mathsf{c.className} = `\mathit{Invoice}' \wedge \mathsf{c} \in \mathsf{do.cSet} \wedge \mathsf{do.accessType} = `\mathrm{R}' \wedge \nexists \mathsf{a',do'}(\mathit{DataOpOf}(\mathsf{a',do'}) \wedge \mathit{SuccessorOf}(\mathsf{a,a'}) \wedge \mathsf{c} \in \mathsf{do'.cSet} \wedge \mathsf{do'.accessType} = `\mathrm{R}'))\}$$

In this case, we obtain task Receive Payment (I) as a result. Q2 can be adapted to find the *first* activity a that accesses objects of a class x: this can be done by ensuring that a is not a successor of any other activity reading objects of x. Also, we can change the *accessType* to consider writing access. For example, to find the last activity updating objects of class *Order*, e.g., to ensure that order details are up-to-date when communicated to the customer, we can set $\mathsf{do'.accessType} = `\mathrm{U}'$ and $\mathsf{c.className} = `\mathrm{Order}'$.

Query Q3 considers the dependencies among data reading/writing operations to check if the data written or modified by an activity *a'* are needed (e.g., read) by any following activities of the same process.

Q3: Which classes have objects that are written by an activity and read by a successive one of the same process?

$$\{\mathsf{c} \mid \mathsf{ClassDB(c)} \wedge \exists\ \mathsf{a},\ \mathsf{do}(\mathit{DataOpOf}(\mathsf{a,do}) \wedge \mathsf{c} \in \mathsf{do.cSet} \wedge (\mathsf{do.accessType} \in \{`\mathrm{I}', `\mathrm{U}'\} \rightarrow \exists\ \mathsf{a',do'}(\mathit{DataOpOf}(\mathsf{a',do'}) \wedge \mathit{SuccessorOf}(\mathsf{a,a'}) \wedge \mathsf{do'.accessType} = `\mathrm{R}' \wedge \mathsf{c} \in \mathsf{do'.cSet})))\}$$

When applying Q3 to the process of Figure 1, we obtain classes Order and Invoice as a result. Indeed, payment details and communication history are inserted in the database without being further modified, and information about customers and loyalty contracts is only read by the process.

Query Q4 retrieves all classes that have objects accessed by a writing operation by concurrent activities of the same process, thus addressing information need I4.

Q4: Which classes have objects that are accessed in writing mode by at least two concurrent activities of the same process?

$$\{\mathsf{c, a, a'} \mid \exists \mathsf{p, do, do'}(\mathit{ActOf}(\mathsf{a,p}) \wedge \mathit{ActOf}(\mathsf{a',p}) \wedge \mathit{DataOpOf}(\mathsf{a,do}) \wedge \mathit{DataOpOf}(\mathsf{a',do'}) \wedge \mathsf{ClassDB(c)} \wedge \mathsf{c} \in \mathsf{do.cSet} \wedge \mathsf{c} \in \mathsf{do'.cSet} \wedge \nexists\ \mathsf{l}\ (\mathsf{l} \in \mathsf{a.label} \wedge \neg \mathsf{l} \in \mathsf{a'.label}) \wedge \neg \mathit{SuccessorOf}(\mathsf{a,a'}) \wedge \neg \mathit{SuccessorOf}(\mathsf{a',a}) \wedge \mathsf{do.AccessType} \in \{`\mathrm{I}', `\mathrm{U}', `\mathrm{D}'\} \wedge \mathsf{do'.AccessType} \in \{`\mathrm{I}', `\mathrm{U}', `\mathrm{D}'\})\}$$

When applied to the process of Figure 1, Q4 returns class Order, which is concurrently updated by activities D and H. Query Q4 can be specialized to retrieve also specific class attributes that are affected by concurrent writing operations.

Query Q5 considers operations that are always performed in a process, i.e., across (I2) and along (I3) *all* process paths.

In detail, Q5 retrieves all the classes having objects that are accessed in all the paths of a process considering the following settings: (i) either a class has objects accessed by an activity labeled with '$\boxdot$', or (ii) it has objects accessed by multiple activities located on paths that together cover all the possible alternative flows, i.e., the logical disjunction of all activity labels is *true*.

Q5: Which data classes are *read* in all process paths and in which processes?

$$\{\mathsf{c, p.name} \mid \mathsf{ClassDB(c)} \wedge \mathsf{Process(p)} \wedge (\exists \mathsf{a, do}\ (\mathit{ActOf}(\mathsf{a,p}) \wedge \mathit{DataOpOf}(\mathsf{a,do}) \wedge \mathsf{c} \in \mathsf{do.cSet} \wedge \mathsf{do.accessType} = `\mathrm{R}' \wedge \mathsf{a.label} = `\boxdot') \vee \forall \mathsf{l,a,do}\ ((\mathit{DataOpOf}(\mathsf{a,do}) \wedge \mathsf{a.label} \neq `\boxdot' \wedge \mathsf{l} \in \mathsf{a.label} \wedge \mathsf{c} \in \mathsf{do.cSet} \wedge \mathsf{do.accessType} = `\mathrm{R}') \rightarrow \exists\ \mathsf{a',do'}\ (\mathit{DataOpOf}(\mathsf{a',do'}) \wedge \neg \mathsf{l} \in \mathsf{a'.label} \wedge \mathsf{c} \in \mathsf{do.cSet} \wedge \mathsf{do.accessType} = `\mathrm{R}')))\}$$

When considering our process Purchase order, Q5 returns classes Order, Customer, loyaltyContract, Item and itemQuantity. The query can be adapted to find classes having objects that are updated, inserted, or deleted in all process paths, thus generalizing Q4.

Q1–Q5 as well as parts of them can be combined to address more complex information needs. For example, by combining and re-arranging the properties expressed by Q3 and Q5, one can easily find the classes having objects modified by a process activity and *always* read by a successive one.

Last but not least, our approach allows formulating quantitative queries. For example, a database engineer may be interested in knowing which data classes are accessed more frequently for database optimization purposes. Access frequency can be quantified by considering the number of (a) data operations performed on the class, (b) distinct activities using it, or (c) distinct processes using it. Although counting cannot be expressed in complex value calculus, we rely on a well-known extension of the relational calculus with aggregate functions [32] and introduce query Q6 using the word *count* to express the homonym aggregate function.

Q6 Which are the classes having objects accessed more frequently?

(a) According to the highest number of data operations.
$\{\mathsf{c} \mid \mathsf{ClassDB(c)} \land \nexists \mathsf{c'}(\mathsf{ClassDB(c')} \land count\{\mathsf{do} \mid \mathsf{DataOp(do)} \land \mathsf{c'} \in \mathsf{do.cSet}\} > count\{\mathsf{do} \mid \mathsf{DataOp(do)} \land \mathsf{c} \in \mathsf{do.cSet}\})\}$

(b) According to the highest number of (distinct) activities.
$\{\mathsf{c} \mid \mathsf{ClassDB(c)} \land \nexists \mathsf{c'}(\mathsf{ClassDB(c')} \land count\{\mathsf{a} \mid \mathsf{Activity(a)} \land \exists\, \mathsf{do}\ (DataOpOf(\mathsf{a,do}) \land \mathsf{c'} \in \mathsf{do.cSet})\} > count\{\mathsf{a} \mid \mathsf{Activity(a)} \land \exists\, \mathsf{do}\ (DataOpOf(\mathsf{a,do}) \land \mathsf{c} \in \mathsf{do.cSet})\})\}$

(c) According to the highest number of (distinct) processes.
$\{\mathsf{c} \mid \mathsf{ClassDB(c)} \land \forall \mathsf{c'}(\mathsf{ClassDB(c')} \rightarrow count\{\mathsf{p} \mid \mathsf{Process(p)} \land \exists\, \mathsf{a, do}\ (ActOf(\mathsf{a,p}) \land DataOpOf(\mathsf{a,do}) \land \mathsf{c'} \in \mathsf{do.cSet})\} \leq count\{\mathsf{p} \mid \mathsf{Process(p)} \land \exists\, \mathsf{a, do}\ (ActOf(\mathsf{a,p}) \land DataOpOf(\mathsf{a,do}) \land \mathsf{c} \in \mathsf{do.cSet})\}$

Q1–Q6 exemplify blueprints of queries that address information needs I1–14.

V. Evaluation

In this section, we describe the controlled experiment devised to evaluate the Activity View (cf. Section V-A) and the proof-of-concept developed to test the feasibility of our approach, considering relational (cf. Section V-B) and XML (cf. Section V-C) data management technologies.

A. Empirical Evaluation through a Controlled Experiment

The integrated exploration of processes and data assumes a good understanding of their connection, which we realized with Activity Views. To assess whether Activity Views improve the understanding of integrated process models and database schemata, we conducted a human-oriented single factor controlled experiment following the guidelines in [33].

The experimental design was informed by previous studies on process model comprehension [34]–[36], and especially by [37], as it considers aspects of integrated modeling. However, while previous research has focused on the effects that intrinsic process model properties, e.g., the structure [34], have on understanding, we explored if the Activity View as an additional artifact can improve the understanding of the interplay between process models and database schemata.

Figure 4 outlines the three main phases of the experiment[6]. PHASE I was organized as a tutorial introducing the Activity View to the subjects. PHASE II was a comprehension task consisting of two supervised runs, while PHASE III consisted of a modeling exercise and a questionnaire-based interview.

[6]More details can be found at https://iris.univr.it/retrieve/handle/11562/976919/97357/RR106%3a2018.pdf

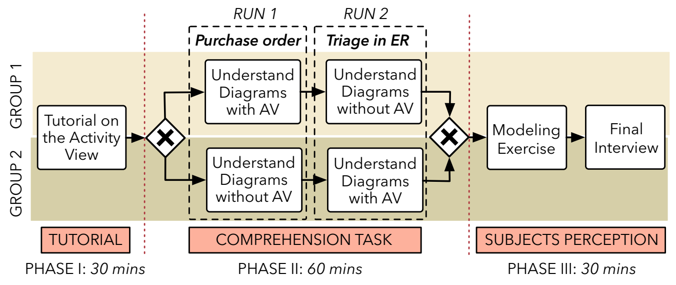


Figure 4. The three phases of the empirical evaluation of the Activity View.

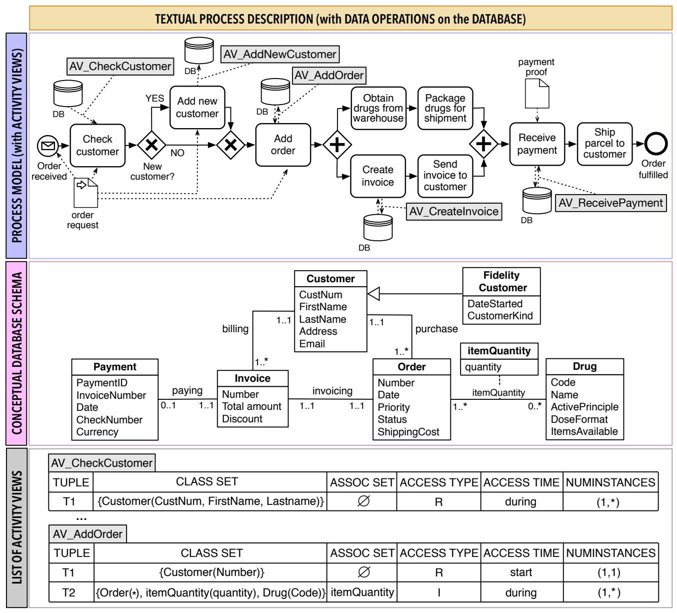
The factor of our experiment is the Activity View with levels *present* or *absent*. The research question we investigated is "Does the Activity View improve the comprehension of integrated processes and data?", which led to alternative hypothesis H_1:*"The Activity View improves the comprehension of integrated processes and data"*, i.e., of the parts of a conceptual database schema needed by one or more process activities to be executed. We measured comprehension task performance considering answering times and answer accuracy.

Subjects were 21 master's students in Computer Science Engineering at our university, 8 master's students in Medical Bioinformatics, and 4 database researchers. All the 33 subjects had attended the same information systems course, where BPMN is explained, and a database design course. Overall, 8 subjects reported having professional experience with database design, whereas no-one worked with BPMN professionally.

Each run of PHASE II was designed as a conceptual analysis task, asking subjects 7 questions regarding the connection between processes and related data [10], e.g., "Which process activities access objects of class Order?" or "Are there classes, whose objects are used only for read operations? If so, which ones?"

We used a within-subjects design and randomly divided subjects into two groups, but keeping the groups balanced w.r.t. their background. During both runs of the comprehension task, all subjects were provided with a BPMN process model, having 9 activities and 4 gateways, the conceptual schema of the database, and a process description including the data operations executed on the database. In each run, one group was also provided with the Activity Views related to the given models (e.g., GROUP 1 in RUN 1). Figure 5 replicates parts of the material provided to the subjects in RUN 1 (colors are used just for presentation). The subjects answered questions on paper and had unlimited time to complete the task. A simplified version of Definition 3 was written on a whiteboard for the whole task duration. For RUN 2, we switched the factor levels (i.e., the Activity Views were given to GROUP 2) and changed the process domain (from a purchase process to a nurse triage).

For accepting H_1, we needed to prove that the subjects with the Activity Views were (i) faster and (ii) more accurate in answering the questions. Both measurements are related to the comprehension of integrated processes and data and, especially, to using the Activity View effectively. Since our experiment was designed to collect *repeated measurements*,



AV_CheckCustomer

TUPLE	CLASS SET	ASSOC SET	ACCESS TYPE	ACCESS TIME	NUMINSTANCES
T1	{Customer(CustNum, FirstName, Lastname)}	∅	R	during	(1,*)

...

AV_AddOrder

TUPLE	CLASS SET	ASSOC SET	ACCESS TYPE	ACCESS TIME	NUMINSTANCES
T1	{Customer(Number)}	∅	R	start	(1,1)
T2	{Order(*), itemQuantity(quantity), Drug(Code)}	itemQuantity	I	during	(1,*)

Figure 5. Example of comprehension task material related to purchase process. The list of Activity Views, here incomplete for space reasons, and the related gray tags were shown only to the treatment group in each run.

i.e., each subject was exposed to all factor levels, we relied on paired analysis [38] and measured, for each subject, the time needed and the number of correct answers for each run.

Figure 6 shows the results of our analysis. In RUN 1, the subjects in the treatment group took an average of 12,45 minutes, and answered 84,03% of the questions correctly. Instead, the group without Activity Views took 21,57 minutes on average and only 39,29% of the answers were correct. The results of RUN 2 are comparable, but revealed a reduction in answering times and correctness for both groups, especially for the one without Activity Views. When interviewed, some subjects reported that they had familiarized with the kind of questions that were asked and, thus, they were faster to answer. Overall, with Activity Views the average comprehension task time decreased by 37,68%, while the number of correct answers increased by 44,15%. The paired t-test and the Wilcoxon signed-rank test [38] applied to both measurements yielded a $p < 0.001$ and, thus, we could accept H_1.

In PHASE III we asked subjects to design a BPMN model starting from a textual process description, and to write the Activity Views connecting their model with a given database schema. Overall, 58,89% of the Activity Views and 83,94% of the designed process models were correct. With this modeling task we aimed to explore how Activity Views are used in practice. When interviewed, subjects were asked to rate how easy the Activity View is to read, understand, use, and write, based on a 5-point Likert scale, with 3 being “neutral” and to suggest potential improvements. Overall, Activity View were rated easy to read (score=4,16), understand (score=4,06), and use (score=4,03), whereas subjects leaned towards “neutral” when asked about writing them (score=3,13). However, we expect such results to improve with proper training and the support of modeling software, as suggested by some subjects.

The results of our evaluation should be read in light of the following limitations. In terms of internal validity, the final results could have been influenced by a potential disparity in the subjects’ modeling expertise and cognitive abilities [36]. Indeed, we did not screen the subjects for familiarity with BPMN and UML class diagrams. However, we provided the same tutorial on process and data modeling to everyone and assigned subjects to groups randomly, but in a balanced way, e.g., ensuring that researchers and students with similar backgrounds were equally split into the two groups. Familiarization is another factor threatening internal validity since subjects could have become familiar with the experimental material and procedure. We considered two different domains for the comprehension task (i.e., purchase and healthcare) to mitigate this risk. Also, we asked a low number of questions addressing various and mostly independent aspects of the provided material. Nevertheless, we could not prevent subjects from familiarizing themselves with the experiment’s format as it emerged when the triangulating the decreased answering times of RUN 2 with the final interviews. However, since answer correctness also decreased, we speculate that familiarization affected comprehension task performance only partially, inducing some people to read the questions less attentively. Probably, changing the order of the questions in each run could have better mitigated this effect.

As for external validity, the relatively small number of subjects and the fact that the majority are students make our results hard to generalize to real organizational environments. However, under certain conditions, there is evidence that graduate students may be proxies for professionals [39]. Besides, since the core contribution of the paper is the analysis approach, the limitations concerning sample selection are less critical. Finally, we acknowledge that the experimental settings may not be realistic as the chosen process models are real but simple, and the questions are centered on the Activity View, i.e., we focus only on some facets of process and data integration that can be addressed during conceptual modeling and analysis. However, this experiment should be seen as an initial study meant to be complemented by the proofs-of-concept presented next and by future studies.

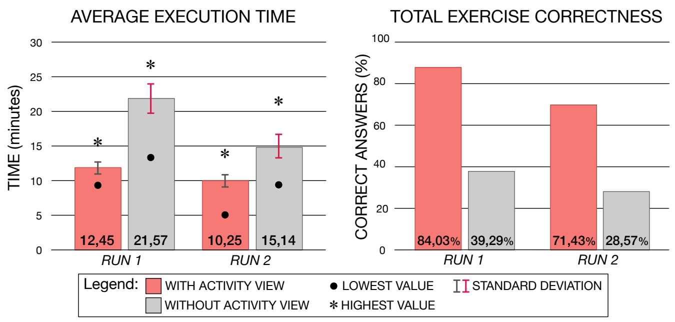


Figure 6. Results of the controlled experiment showing average execution time with standard deviation (left) and percentage of correct answers (right).

B. Proof-of-concept: Implementation with PostgreSQL

As a proof-of-concept, we considered the standard relational database technology and implemented a Java-based tool, using the Camunda BPMN model API[7]. The tool parses and labels a BPMN 2.0 XML process description and is interfaced with a relational database implemented using PostgreSQL[8]. The database was implemented by following the schema in Figure 2, but translating all the fields containing complex set-valued attributes into many-to-many relations. For example, activity successors, originally captured by attribute successors of Succ, were represented as relation NewSucc(<u>activity, successor</u>) which relates each activity to one or more successive ones. Similarly, we added:

- Label(<u>activityID, literal</u>), where literal is an integer, so that value 0 corresponds to $\boxdot$;
- ComposedOf(<u>avID, opID</u>);
- Cset(<u>opid, attrName, className, dbID</u>);
- Aset(<u>opid, assocName, dbID</u>).

Figure 7 outlines the implemented proof-of-concept. The parser reads the XML description of a BPMN process provided in input and creates a Java object containing all the information about the parsed process model. Then, Algorithm 1 labels activities as explained in Section IV-A. Finally, SQL statements are created to populate the database. In detail, relations 1-10 in Figure 2, Label and newSucc are populated automatically by the tool, whereas the remaining relations are populated by directly inserting data in the database through SQL statements.

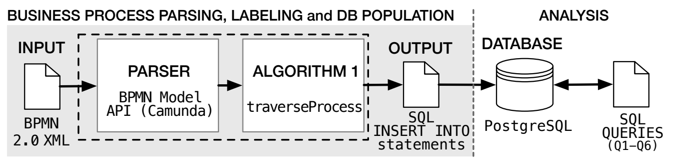


Figure 7. Overview of the proof-of-concept implemented using PostgreSQL.

We translated queries Q1-Q6 into SQL and run them against the described database storing data related to different processes and application domains. As an example, in Figure 8, we report Q4-SQL, omitting the part of the query needed to select the process and the domain database, for simplicity. To establish whether two activities are concurrent, Q4-SQL checks that they are not successors of one another and that their labels do not have opposing literals. Since literals are treated as integers, opposite literals are represented by integers having the same absolute value, different from 0, but opposite sign (e.g., if p corresponds to 1 then $\neg p$ corresponds to -1).

In general, using SQL as query language is beneficial for several reasons. Among others, SQL is the standard language for querying relational database management systems and supports the definition of aggregation operators (e.g., Q6). All the other queries presented in Section IV-B can be translated in SQL as illustrated for Q4.

C. Proof-of-Concept: Querying XML data

Our approach can be implemented using logical (and physical) database models other than the relational ones. As an example, in this paragraph we discuss how an XML database can be used for implementation.

Figure 10 sketches a possible fragment of an XML document storing information about the BPMN process model (based on the 2.0 XML structure) and about Activity Views and data classes related to our motivating example. The structure of the process derives from the standard XML

[7]https://docs.camunda.org/manual/7.9/user-guide/model-api/bpmn-model-api/

[8]https://www.postgresql.org

Q4-SQL Which classes have objects accessed in writing mode by at least two concurrent process activities?

```
SELECT DISTINCT C1.className, A1.fnID, A2.fnID,
  A1.activityName, A2.activityName
FROM Cset C1 JOIN DataOp D1 ON C1.opID = D1.opID
  JOIN ComposedOf CO1 ON D1.opID = CO1.opID JOIN
  Activity A1 ON CO1.avID = A1.avID JOIN FlowNode F1
   ON  A1.fnID = F1.fnID, Cset C2 JOIN dataOp D2 ON
  C2.opID = D2.opID JOIN ComposedOf CO2 ON D2.opID =
   CO2.opID JOIN Activity A2 ON CO2.avID = A2.avID
  JOIN FlowNode F2 ON  A2.fnID = F2.fnID
WHERE C1.className = C2.className AND F1.ProcessID =
   F2.ProcessID AND D1.accessType IN ('U', 'I', 'D')
   AND D2.accessType IN ('U', 'I', 'D') AND  A1.fnID
   < A2.fnID AND A1.fnID NOT IN(
    SELECT activity
    FROM NewSucc S1
    WHERE A1.fnID = S1.activity AND
        S1.successor = A2.fnID) AND A2.fnID NOT IN(
    SELECT activity
    FROM NewSucc S2
    WHERE A2.fnID = S2.activity AND
        S2.successor = A1.fnID)
    AND NOT EXISTS(SELECT 1
        FROM Label L1, Label L2
        WHERE L1.activityID = A1.fnID AND
        L2.activityID = A2.fnID AND
        L1.literal <> 0 AND L2.literal <> 0 AND
        L1.literal = -1*L2.literal);
```

Figure 8. SQL expression corresponding to query Q4.

```
<PairsOfActivities>{
  for $p in doc("ProcessViews.xml")//bpmn:process
  for $a1 in $p/bpmn:task, $a2 in $p/bpmn:task
    let $dos1 := for $d1 in doc("ProcessViews.xml")//
      ActivityView where $d1/@activityRef= $a1/@id
       return $d1//dataOperation
    let $dos2 := for $d2 in doc("ProcessViews")//
      ActivityView where $d2/@activityRef= $a2/@id
       return $d2//dataOperation
  where ($a1/@id < $a2/@id) and (some $l1 in $a1//
    literal satisfies (some $l2 in $a2//literal
    satisfies $l1= concat("not ",$l2)))
    and (every $do1 in $dos1 satisfies (some $do2 in
    $dos2 satisfies
    $do2/cset/@classSet = $do1/cset/@classSet and
    $do2/asSet/@AsSet = $do1/aset/@AsSet and
    $do2/bpmn:accessType = $do1/bpmn:accessType and
    $do2/bpmn:accessTime = $do1/bpmn:accessTime and
    $do2/bpmn:numInstances/@minCardinality =
      $do1/bpmn:numInstances/@minCardinality and
    $do2/bpmn:numInstances/@maxCardinality =
      $do1/bpmn:numInstances/@maxCardinality))
    and (every $do2 in $dos2 satisfies (some $do2 in
      $dos2 satisfies
    (: the symmetrical conditions follow, as for
      $do1 in $dos1 :)
  return <Tpair>{$a1}{$a2}</Tpair>
}</PairsOfActivities>
```

Figure 9. XQuery expression corresponding to query Q1 expressed through the complex value calculus in Section IV-B

representation provided by BPMN, while the elements for Activity Views and data classes are derived directly from the complex value schema, using both element nesting and element references.

```
  <?xml version="1.0" encoding="UTF-8"?>
  <definitions xmlns="http://www.omg.org/spec/BPMN
/20100524/MODEL">

  <process id="Process_1" name="PurchaseManagement">
  <!-- ............. -->
  <task id="TaskA" name="Manage order (A)">
  <incoming>SequenceFlow_01</incoming>
  <outgoing>SequenceFlow_02</outgoing>
  <label><literal value="dotBox"/></label>
  </task>
  <!-- ............. -->
  <task id="TaskF" name="Offer discount (F)">
  <incoming>SequenceFlow_12<incoming>
  <outgoing>SequenceFlow_13</outgoing>
  <label><literal value="p"/> </label>
  <label><literal value="q"/> </label>
  </task>
  <!-- ............. -->
  </process>

  <!-- ............. -->

  <activityView processRef="Process_1"
                activityRef="TaskA">
  <dataOperation>
  <cset classSet="Customer Order ItemQuantity Item"/>
  <aset AsSet="purchase ItemQuantity"/>
  <accessTime> start </accessTime>
  <accessType> R </accessType>
  <numInstances minCardinality="1" maxCardinality="1"/>
  </dataOperation>
  <dataOperation>
  <cset classSet="Order">
  <attributes class="Order"><att>status</att>
  </attributes>
  </cset>
  <accessType> U <accessType>
  <accessTime> during </accessTime>
  <numInstances minCardinality="1" maxCardinality="1"/>
  </dataOperation>
  </activityView>

  <!-- ............. -->

  <dataClass>
  <className> Customer </className>
  <!-- ............. -->
  </dataClass>
  </definitions>
```

Figure 10. Fragment of an XML document containing information about the process and the Acitivity Views of our motivating example. The namespace prefix `bpmn` has been removed for improving readability.

An example of query that can be run against such kinds of XML documents is summarized in Figure 9. In this XQuery expression, the complex value query Q1 (i.e., *Which pairs of activities lying on alternative process paths have the same Activity View?*) is mapped to the XML specification of the database (e.g., the one in Figure 10), and the nesting of elements and the references between element identifiers are used to represent complex data.

VI. Related Work

The relationship between processes and data has been central to several works in the business process management and database fields [2], considering all the phases of the business process life-cycle and different process granularity and data abstraction levels. However, the integrated modeling, verification, enactment, and analysis of these two perspectives still presents relevant research challenges [4], [7], [9], [20]. Among process modeling approaches, we can distinguish between those aiming to enhance activity-centric processes for the modeling and execution of the data perspective [3], [9], [10], [13], [40] and those focusing on defining languages for data- or object-aware process modeling, verification and enactment [4], [7], [8], [14], [41], [42]. Our approach falls within the first research line and, particularly, aims to define a conceptual and unified view of a process model and database schema that can be used to explore the data perspective of a process at design-time. Some recent proposals [9], [24], [40] remark the importance of introducing conceptual modeling frameworks to support the design, analysis and execution of integrated activity-centric processes and persistent data. The framework proposed in [9] links BPMN process models to a UML class diagram representing the domain data of interest, which includes a class "Artifact" containing all the process variables needed for process execution. Data manipulations are defined as OCL (Object Constraint Language) expressions on such variables. The execution relies on relational SQL technology: the UML class diagram is encoded as a relational database, the BPMN diagram is translated into a Petri net, and OCL contracts are encoded as logic rules that support the derivation of SQL statements that can be run against the database. *Db-nets* are proposed in [40] to support data-aware process modeling and verification and are grounded in colored Petri nets and relational databases.

In [3], the authors address the problem of modeling and executing BPMN processes with complex data dependencies. They extend data objects with identifiers, information about their life-cycle, and fields to express complex correlations among multiple objects. These data annotations comprehensively define activity pre- and post-conditions, and are used to derive SQL queries that execute the modeled data dependencies. Both the approaches introduced in [9], [40] focus on reducing the gap between control flow and persistent data aspects by means of an intermediate layer (i.e., a class "Artifact" in [9] and the data logic layer of db-nets in [40]) capturing changes over process data. The Activity View follows this line of thought, as it embraces the idea of keeping the process model and database schema untouched while connecting them. However, the design and analysis goals discussed in this paper are substantially different from those presented in [9], [40]. Indeed, we focus on the human-driven exploration of connected conceptual models at design-time, and do not address the (automatic) execution of the overall "connected system", as done also in [3]. Thereby, our approach abstracts from aspects related to process execution, e.g., the violation of integrity constraints on the database or the detection of undesired data evolution. Indeed, despite our approach can be a starting point for detecting data inconsistencies [12], [13] and for reviewing data access privileges [4], in this paper we abstain from looking into such analyses as they require to consider lower process granularity and data abstraction levels (e.g., execution traces and database instances).

The problem of providing a unified view of processes and data was tackled also in the field of process mining, mostly by proposals focusing on obtaining and transforming data from databases to derive event logs [24], [43], [44]. In [24], the authors propose a conceptual framework that combines information coming from an event log, a transaction log, and a relational database storing the current values of data attributes. The framework enables the in-depth exploration of business process behavior through nine mapping operations defined between the three data sources. In [43], propose an approach grounded in conceptual modeling for supporting the extraction of event logs from legacy information systems. In [44], the authors present a meta-model to support the creation of multi-perspective process logs from data coming from different sources. The meta-model includes three levels of process granularity, i.e., (i) processes, (ii) instances, and (iii) events, and three data abstraction levels, namely (i) data model, (ii) objects, and (i) versions, i.e., the instantiations of an object over a certain amount of time. The process and the data perspectives are connected at the level of events and versions.

Compared to our work, the proposals in [24] and [44] focus on connecting processes and data at a lower level, i.e., considering trace events and database instances. Both the works in [44] and [24] are valuable starting points for extending Activity Views to deal with process instances and logical database models, which is part of our future research agenda. Recently, in [26], the authors consider the role of data in process similarity checking. The way they incorporate the data-flow information into the process control flow by considering data reading and writing semantics and the introduction of different types of similarity measures may be viewed as a last proof of the relevance of the discussed Information Needs. Further recent contributions confirmed the importance of bridging the gap between data and process design and explored some research directions complementary to the one focusing on conceptual modeling proposed in this paper. They deal with: a formal language for modeling process- and information-related concepts and constraints [45]; the support to analyze the impact of unexpected data changes during process executions [46]; an operational framework, mainly based on BPMN and SQL languages, supporting modeling and verification of business process models enriched with data management capabilities [11]. All these recent contributions share a sound theoretical foundation with our approach. However, their focus is not on the conceptual modeling of processes and data, as they consider more specific data dependencies and (runtime) constraints and do not focus on the seamless conceptual modeling of processes and data.

VII. Discussion and Conclusions

In this paper, we proposed an approach to capture and query the data perspective of business processes, to support conceptual modeling and analysis tasks. By allowing designers to explore the data manipulations realized by process activities on a database schema, our approach supports the understanding at design time of how data are used by a process, fostering process (re)-design and improvement.

Compared to data-aware approaches that consider also execution, an obvious limitation of our approach, as it focuses solely on conceptual design and analysis, is that real execution traces are not considered/mined. However, the proposed approach allows designers to analyze and specify the connections between process models and data before facing the implementation of the considered business processes. Both inter- and intra-process data-related features may be analyzed. Another limitation, we plan to face in future work, is that Activity Views are not yet supported by a diagrammatic notation integrating BPMN and UML class diagrams. We preferred to focus on a formal description of our proposal, making it sound and independent from the (many) possible implementations.

As for other future work, we aim to extend the scope of our approach by considering the modeling and analysis of non-persistent data (e.g., process variables), as done by [3], [9], [44], and improve the proof-of-concept discussed in Section V, considering a thorough evaluation with end-users and its integration into open-source process modeling tools.

References

[1] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Springer Publishing Company, Incorporated, 2010.

[2] D. Calvanese, G. De Giacomo, and M. Montali, "Foundations of data-aware process analysis: a database theory perspective," in *32nd ACM SIGMOD Symposium on Principles of Database Systems (PODS)*. ACM, 2013, pp. 1–12.

[3] A. Meyer, L. Pufahl, D. Fahland, and M. Weske, "Modeling and enacting complex data dependencies in business processes," in *International Conference on Business Process Management (BPM)*, ser. LNCS. Springer, 2013, vol. 8094, pp. 171–186.

[4] V. Künzle and M. Reichert, "PHILharmonicFlows: towards a framework for object-aware process management," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 23, no. 4, pp. 205–244, 2011.

[5] Object Management Group, "Business Process Model and Notation (BPMN), v2.0.2," http://www.omg.org/spec/BPMN/2.0.2/.

[6] H. Leopold, J. Mendling, and O. Günther, "Learning from Quality Issues of BPMN Models from Industry," *IEEE Software*, vol. 33, no. 4, pp. 26–33, 2016.

[7] D. Calvanese, M. Montali, F. Patrizi, and A. Rivkin, "Modeling and in-database management of relational, data-aware processes," in *International Conference on Advanced Information Systems Engineering (CAiSE)*, P. Giorgini and B. Weber, Eds. Springer, 2019, pp. 328–345.

[8] A. Artale, A. Kovtunova, M. Montali, and W. M. P. van der Aalst, "Modeling and reasoning over declarative data-aware processes with object-centric behavioral constraints," in *Int. Conf. on Business Process Management (BPM)*. Cham: Springer, 2019, pp. 139–156.

[9] G. De Giacomo, X. Oriol, M. Estañol, and E. Teniente, "Linking Data and BPMN Processes to Achieve Executable Models," in *International Conference on Advanced Information Systems Engineering (CAiSE)*, ser. LNCS, vol. 10253. Springer, 2017, pp. 612–628.

[10] C. Combi, B. Oliboni, M. Weske, and F. Zerbato, "Conceptual modeling of processes and data: Connecting different perspectives," in *International Conference on Conceptual Modeling (ER)*, ser. LNCS, vol. 11157. Springer, 2018, pp. 236–250.

[11] S. Ghilardi, A. Gianola, M. Montali, and A. Rivkin, "Delta-bpmn: A concrete language and verifier for data-aware BPMN," in *Business Process Management - 19th International Conference, BPM 2021, Rome, Italy, September 06-10, 2021, Proceedings*, ser. Lecture Notes in Computer Science, A. Polyvyanyy, M. T. Wynn, A. V. Looy, and M. Reichert, Eds., vol. 12875. Springer, 2021, pp. 179–196. [Online]. Available: https://doi.org/10.1007/978-3-030-85469-0_13

[12] N. Trčka, W. M. Van der Aalst, and N. Sidorova, "Data-flow anti-patterns: Discovering data-flow errors in workflows," in *International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer, 2009, pp. 425–439.

[13] C. Combi, B. Oliboni, M. Weske, and F. Zerbato, "Conceptual modeling of inter-dependencies between processes and data," in *ACM Symposium on Applied Computing (SAC '18)*. ACM, 2018, pp. 110–119.

[14] X. Oriol, G. De Giacomo, M. Estañol, and E. Teniente, "Embedding reactive behavior into artifact-centric business process models," *Future Generation Computer Systems*, vol. 117, pp. 97 – 110, 2021.

[15] S. España, A. González, and O. Pastor, "Communication analysis: A requirements engineering method for information systems," in *International Conference on Advanced Information Systems Engineering (CAiSE)*, ser. LNCS, vol. 5565. Springer, 2009, pp. 530–545.

[16] M. Brambilla, S. Comai, P. Fraternali, and M. Matera, "Designing web applications with webml and webratio," in *Web Engineering*, ser. Human-Computer Interaction Series. Springer, 2008, pp. 221–261.

[17] O. Pastor and J. C. Molina, *Model-driven architecture in practice - a software production environment based on conceptual modeling*. Springer Science & Business Media, 2007.

[18] Object Management Group, "Unified Modeling Language, v2.5," available at: http://www.omg.org/spec/UML/2.5/.

[19] A. Borgida, D. Toman, and G. E. Weddell, "On special description logics for processes and plans," in *Description Logics*, ser. CEUR Workshop Proceedings, vol. 2373. CEUR-WS.org, 2019.

[20] D. Calvanese, S. Ghilardi, A. Gianola, M. Montali, and A. Rivkin, "From model completeness to verification of data aware processes," in *Description Logic, Theory Combination, and All That*, ser. LNCS, vol. 11560. Springer, 2019, pp. 212–239.

[21] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Conceptual modeling for data integration," in *Conceptual Modeling: Foundations and Applications*. Springer, 2009, pp. 173–197.

[22] M. Lenzerini and C. Daraio, "Challenges, approaches and solutions in data integration for research and innovation," in *Springer Handbook of Science and Technology Indicators*, ser. Springer Handbooks. Springer, 2019, pp. 397–420.

[23] S. Abiteboul and C. Beeri, "The power of languages for the manipulation of complex values," *VLDB J.*, vol. 4, no. 4, pp. 727–794, 1995.

[24] A. Tsoury, P. Soffer, and I. Reinhartz-Berger, "A Conceptual Framework for Supporting Deep Exploration of Business Process Behavior," in *International Conference on Conceptual Modeling (ER)*, ser. LNCS, vol. 11157. Springer, 2018, pp. 58–71.

[25] J. L. de la Vara, M. H. Fortuna, J. S. Díaz, C. M. L. Werner, and M. R. S. Borges, "A requirements engineering approach for data modelling of process-aware information systems," in *BIS*, ser. LNBIP, vol. 21. Springer, 2009, pp. 133–144.

[26] C. Liu, Q. Zeng, L. Cheng, H. Duan, and J. Cheng, "Measuring similarity for data-aware business processes," *IEEE Transactions on Automation Science and Engineering*, pp. 1–13, 2021, in press.

[27] C. Combi and M. Gambini, "Flaws in the flow: The weakness of unstructured business process modeling languages dealing with data," in *On the Move to Meaningful Internet Systems: OTM 2009, Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009, Vilamoura, Portugal, November 1-6, 2009, Proceedings, Part I*, ser. Lecture Notes in Computer Science, R. Meersman, T. S. Dillon, and P. Herrero, Eds., vol. 5870. Springer, 2009, pp. 42–59. [Online]. Available: https://doi.org/10.1007/978-3-642-05148-7_6

[28] M. Dumas, L. García-Bañuelos, and A. Polyvyanyy, "Unraveling unstructured process models," in *Business Process Modeling Notation*. Springer, 2010, pp. 1–7.

[29] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[30] L. Hunsberger, R. Posenato, and C. Combi, "A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks," in *22nd Int. Symp. on Temporal Representation and Reasoning (TIME)*. IEEE Press, 2015, pp. 4–18.

[31] S. Abiteboul and J. Van den Bussche, "Deep equality revisited," in *International Conference on Deductive and Object-Oriented Databases*. Springer, 1995, pp. 213–228.

[32] G. Özsoyoğlu, Z. Özsoyoğlu, and V. Matos, "Extending relational algebra and relational calculus with set-valued attributes and aggregate functions," *ACM Transactions on Database Systems (TODS)*, vol. 12, no. 4, pp. 566–592, 1987.

[33] C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer-Verlag, 2012.

[34] J. Melcher, J. Mendling, H. A. Reijers, and D. Seese, "On measuring the understandability of process models," in *Business Process Management Workshops*, ser. LNBIP, vol. 43. Springer, 2010, pp. 465–476.

[35] H. A. Reijers, J. Mendling, and R. M. Dijkman, "Human and automatic modularizations of process models to enhance their comprehension," *Information Systems*, vol. 36, no. 5, pp. 881 – 897, 2011.

[36] J. Recker, H. A. Reijers, and S. G. van de Wouw, "Process model comprehension: the effects of cognitive abilities, learning style, and strategy," *Communications of the association for information systems*, vol. 34, no. 1, p. 9, 2014.

[37] W. Wang, M. Indulska, S. Sadiq, and B. Weber, "Effect of linked rules on business process model understanding," in *International Conference on Business Process Management (BPM)*, ser. LNCS, vol. 10445. Springer, 2017, pp. 200–215.

[38] H. Motulsky, *Intuitive biostatistics: a nonmathematical guide to statistical thinking*. Oxford University Press, USA, 2014.

[39] M. Höst, B. Regnell, and C. Wohlin, "Using students as subjects—a comparative study of students and professionals in lead-time impact assessment," *Empirical Software Engineering*, vol. 5, no. 3, pp. 201–214, 2000.

[40] M. Montali and A. Rivkin, "DB-Nets: On the Marriage of Colored Petri Nets and Relational Databases," in *Transactions on Petri Nets and Other Models of Concurrency XII*. Springer, 2017, pp. 91–118.

[41] K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su, "Towards formal analysis of artifact-centric business process models," in *International Conference on Business Process Management (BPM)*. Springer, 2007, pp. 288–304.

[42] Y. Sun, J. Su, B. Wu, and J. Yang, "Modeling data for business processes," in *IEEE 30th International Conference on Data Engineering (ICDE)*. IEEE Press, 2014, pp. 1048–1059.

[43] D. Calvanese, T. E. Kalayci, M. Montali, and S. Tinella, "Ontology-based data access for extracting event logs from legacy data: the onprom tool and methodology," in *International Conference on Business Information Systems*. Springer, 2017, pp. 220–236.

[44] E. González López de Murillas, H. A. Reijers, and W. M. P. van der Aalst, "Connecting databases with process mining: a meta model and toolset," *Software & Systems Modeling*, vol. 18, no. 2, pp. 1209–1247, 2019.

[45] A. Polyvyanyy, J. M. E. M. van der Werf, S. Overbeek, and R. Brouwers, "Information systems modeling: Language, verification, and tool support," in *Advanced Information Systems Engineering - 31st International Conference, CAiSE 2019, Rome, Italy, June 3-7, 2019, Proceedings*, ser. Lecture Notes in Computer Science, P. Giorgini and B. Weber, Eds., vol. 11483. Springer, 2019, pp. 194–212. [Online]. Available: https://doi.org/10.1007/978-3-030-21290-2_13

[46] A. Tsoury, P. Soffer, and I. Reinhartz-Berger, "Data impact analysis in business processes," *Bus. Inf. Syst. Eng.*, vol. 62, no. 1, pp. 41–60, 2020. [Online]. Available: https://doi.org/10.1007/s12599-019-00611-5

Carlo Combi is full professor of Computer Science at the Dept. of Computer Science, University of Verona. In 1993, he received the Ph.D. degree in biomedical engineering from the Politecnico of Milan. From 2009 to 2013 he was chair of the Artificial Intelligence in Medicine Society (AIME). Since 2017 he is Editor-in-Chief of the journal Artificial Intelligence in Medicine. His main research interests are in the database and information systems field, with an emphasis on clinical data and processes.

Barbara Oliboni is associate professor at the Dept. of Computer Science of the University of Verona. She received the Ph.D. degree in Computer Engineering by the Politecnico of Milan. Her main research interests are in the database field, with an emphasis on semistructured data, temporal information, business processes management, and clinical information management. She is part of the Program Committee of International Conferences, and reviewer for International Journals.

Francesca Zerbato is a post-doc at the Institute of Computer Science, University of St. Gallen, Switzerland. She obtained her Ph.D. degree in Computer Science from the University of Verona in 2019. Her main research interests are in information systems and BPM, with an emphasis on process and data modeling, and healthcare applications. She is a member of the program committee of the BPM and SAC conferences, and managing editor of the journal Artificial Intelligence in Medicine.