Verification of Data-Value-Aware Processes and a Case Study on Spectrum Auctions

Elaheh Ordoni*, Jutta Mülle†, Klemens Böhm‡
Institute for Program Structures and Data Organization
Karlsruhe Institute of Technology
76131 Karlsruhe, Germany
Email: {*elaheh.ordoni, †jutta.muelle, ‡klemens.boehm}@kit.edu

***Abstract*—Verification techniques are fundamental to improve the reliability of process designs in practice. In application domains like auctions, the issue is extremely valuable; the goal of auction designers is to prevent undesirable executions and maximize certain outcome measures. Current verification approaches tend to be confined to control flows, even though data values play a significant role. We address this issue by proposing a new data-value-aware verification approach: We enhance process models with information on data values. Then we transform the data-value-aware process models to Petri Nets, respecting the semantics of data value usages. By employing an off-the-shelf model checker and specifying data-value centered properties, one can now verify data-value-aware process models. A distinctive feature of our approach is the specification of data values and of their modifications during the process. This enables the verification of interesting properties in many domains. We evaluate our approach against the use case of Simultaneous Multi-Round (SMR) spectrum auctions. We verify SMR auction models and compute extreme values of common auction measures such as *revenue*.**

I. Introduction

Industry takes great interest in verification techniques to improve the reliability of process designs. To illustrate the importance of reliable designs, think of spectrum auctions. Spectrum auction revenue has been an important source of governmental income. In Germany and the UK for instance, it earned 50.8 and 37.5 billion Euros in 2000 [1]. Despite much testing of various formats of spectrum auctions with human subjects ("experiments" in what follows), auctions with embarrassing outputs have happened in the past. In the Dutch UMTS Auction in 2000, the low revenue caused a fiasco in public policy [2]. In another example [3], about fifty percent of the items remained unsold. Clearly, discovering auction flaws by means of evaluation methods, e.g., experiments is extremely difficult, because this requires to check all possible courses of the auction in question. For example, the experimental design in [4] results in evaluating more than 13 million possible paths. Obviously, this is beyond the capacity of any laboratory. So those catastrophic results might be the consequence of possible courses of auctions which remain uninspected in experiments.

To deal with this issue, verification methods have been proposed to find undesirable system behavior in a precise and unambiguous manner, see [5] for an overview. However, most of these methods focus on workflows without the data aspect, and, in particular, regardless of *data values*. A data value is a specific characteristic of a *data object* in a process model, e. g., $500 as the price of a product. In many settings, data values can be modified during the process, and the behavior depends on them. In principle, behavioral analysis enables verification of data-value centered properties. For example, to arrive at the lowest revenue in auctions, data values such as "bidder's budget", "price of products", etc., in the process model must be known. However, current verification approaches do not sufficiently support data-value-aware processes and modifications of data values in particular, see Section VI.

To overcome this issue, we propose a new approach to represent and verify data-value-aware processes. We enhance the process model to facilitate specifying the usage of data values and their modifications during the process flow, using so-called Data-Value Enhancement Functions (EF). Next, we propose a new algorithm to transform this data-value-aware process model to Petri Nets, a formal modeling language, for which a comprehensive set of analysis tools is available [6]. This way of generating a Petri-Net-based representation is generic, i. e., does not take any application-specific characteristic into account. Our approach is analogous to [7]. However, they only address the optional and alternative usage of data objects by activities as a whole and not at the level of data values. To our knowledge, we are the first to consider data values and their modifications in the transformation of process models to Petri Nets. Based on the generated Petri Nets, one can specify data-value centered properties of the process model as CTL formulas [8]. By deploying an off-the-shelf model checker [9], one is now able to find, say, undesirable outcomes. We evaluate our approach against the use case of Simultaneous Multi-Round (SMR) spectrum auctions. One can then show the executions yielding undesirable outcomes to auctioneers to let them improve the auction design. We summarize our contributions as follows: (1) We propose an approach to enrich process models with the usage of data values. (2) We propose a general approach to verify data-value centered properties of process models, supporting both constants and value modifications in a process model. (3) We demonstrate the usefulness of our approach by modeling SMR auctions in BPMN 2.0 [10] including data values and implementing a tool to detect undesired executions of SMR auctions.

Our evaluation shows that our approach allows to check important data-value centered properties of SMR auctions. For

example, one can now find the worst possible values of three relevant measures of auction designs [4]: *auctioneer's revenue*, i. e., the sum of the final price of products, *bidder's profit*, i. e., the sum of the unused budgets of bidders who have won a product, and *auction's efficiency*, i. e., selling the products to bidders with the highest budgets. Put differently, we cannot only prove the presence of flaws, but also their absence. Last but not least, when we detect an undesirable outcome, we can provide a counterexample. On the downside, the auctions whose verification has been possible without state-space explosion are smaller than the ones having taken place in reality. Thus, further work is required to control the state-space explosion, for instance by reducing the Petri Nets via domain knowledge. Having said this, our approach has been useful from a practical perspective: Our verification of small auctions has uncovered potential for improvement in existing designs, as we will explain; auction designers may now invent alternative formats and compare them to the settings analyzed here in a rigorous manner.

Paper outline: Section II explains SMR spectrum auctions. Section III provides definitions. Section IV features our approach, with spectrum auctions as illustration. Section V discusses its implementation and evaluation. Section VI and VII discuss related work and conclude.

II. SMR Spectrum Auctions

Simultaneous Multi-Round (SMR) auctions have been the standard format to allocate spectrum licenses to bidders for more than two decades [11]. Such an auction consists of several rounds to award the spectrum to bidders with the highest *bid*. At the beginning of an auction, the auctioneer specifies a *reserve price* for each license, i. e., its lowest acceptable price. In each round, participants can bid on individual licenses simultaneously. The highest *bid* for each license will be its reserve price in the following round. The auction ends when there is no new bid for any license. Then the bidder with the highest bid for a product is its winner. In this auction type, there are several so-called activity rules, which can impact auction performance [12]:

- *Capacity rule*: Each bidder has a *capacity point*, the maximum number of licenses he may win. This rule prevents bidders from winning too many items.
- *Eligibility rule*: Before the auction starts, each bidder has a certain *eligibility point*. It determines how many bids he can make in each round. The eligibility point of a bidder decreases if his number of bids in a round is lower than his current eligibility point. A bidder cannot bid any more if his eligibility point is zero.

Figure 1 represents the BPMN model of the SMR spectrum auction. with capacity and eligibility rules. This model consists of three subprocesses. The first one checks *availability of bidders*, i.e., whether the bidder can afford a certain license which has not won yet. The auction continues if there is at least one qualified bidder to specify their bids in Subprocess *bidders offering bids*. Activity *record requested products* marks all requested licenses to consider them in the *winner determination* phase. Subprocess *winner specification* outputs the new reserve prices and the winners. Based on these results, the auction process updates the bidders' capacity points in two activities: *increase capacity* and *decrease capacity*. These three subprocesses are repeated until no bids remain.

III. Data-Value Specific Definitions

To verify process models we rely on model checking of Petri Nets [6]. To specify the properties, we use *Computation Tree Logic* (CTL) [8]. See the conventional definitions in Section III-A. Verifying data-value-aware processes requires representations of process models from a specific domain. See the new data-value specific definitions in Section III-B.

A. Conventional Definitions

Definition 1. *(Petri Nets [6]) A Petri Net is a triple* (S, T, W) *where:*

- S *is a finite set of places, represented by circles*
- T *is a finite set of transitions, represented by rectangles.* $S \cap T = \varnothing$
- $W \subseteq (S \times T) \cup (T \times S)$ *is a set of directed arcs (flow relation).*

A marking of a Petri Net is a mapping: $M : P \to N$ *which assigns a non-negative number of tokens to each place.* M_0 *is the initial marking. For each transition there are directly preceding places called input places:* $\bullet t = \{s \in S \mid W(s,t) > 0\}$ *and subsequent places called output places:* $t\bullet = \{s \in S | W(t,s) > 0\}$. *A transition* t *can fire when all its input places have enough tokens; firing leads to a new state* M. M *results from* M *by* t *consuming a token from each of its input places and producing a token in each of its output places. The set of all states reachable from the initial state* M_0 *is the state space of the Petri Net.*

It is possible to check whether a Petri Net fulfills a property ϕ defined on its state space, by completely exploring this space. This method is known as model checking [9]. To specify properties, some powerful formal languages such as Computation Tree Logic are known.

Definition 2. *(Computation Tree Logic – CTL) An atomic proposition* α *is a CTL formula. If* Φ_1 *and* Φ_2 *are CTL formulas then* $\neg\Phi_1$, $\Phi_1 \wedge \Phi_2$, $\Phi_1 \vee \Phi_2$, $AX\Phi_1$, $EX\Phi_1$, $AG\Phi_1$, $EF\Phi_1$, $A[\Phi_1 \cup \Phi_2$, $E[\Phi_1 \cup \Phi_2]$ *are CTL formulas.*
The operators always occur in pairs: a path operator (A *or* E*) and a state operator (*X, G, F *or* $\cup$*).* A *means the formula holds in all subsequent execution paths,* E *means that at least one execution path exists where the formula holds.* X *means that the formula holds in the next state,* G *means that the formula holds in all subsequent states,* F *means that the formula holds at least in one subsequent state, and* $[\Phi_1 \cup \Phi_2]$ *means that* Φ_1 *holds until* Φ_2 *is fulfilled.*

B. Data-Value Specific Definitions

Definition 3 is the basis for both modeling and compliance checking of data-value-aware processes. It extends the definition in [13] with typed data objects. Supporting several

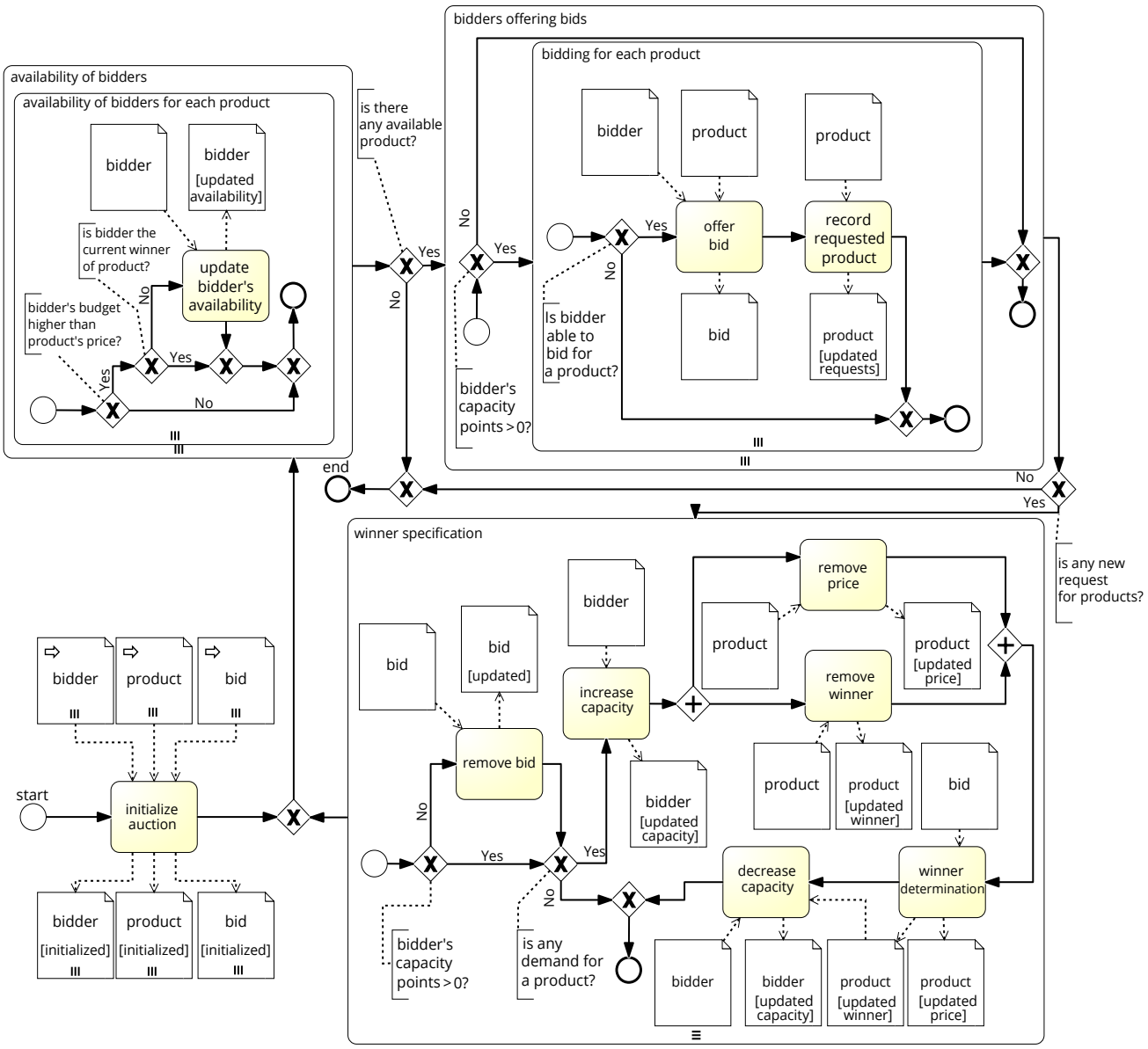


Fig. 1. BPMN model for SMR auction

data objects of the same type, e.g., several *bidder*s, *bid*s and *product*s, requires type-specific keys for data objects.

Definition 3. *(Process Domain) A Process Domain $\mathcal{D}$ is a tuple $\mathcal{D} = (\mathbb{T}, \mathbb{E}, \mathbb{O}, \mathbb{O}_{\mathbb{T}}, \mathbb{A}, \mathbb{D}, dom, att, type)$ where:*

- *$\mathbb{T}$ is a set of activity types*
- *$\mathbb{E}$ is a set of event types*
- *$\mathbb{O}$ is a universe of data objects*
- *$\mathbb{O}_{\mathbb{T}}$ is a set of data-object types*
- *$\mathbb{A}$ is a set of attributes*
- *$\mathbb{D}$ is a set of discrete data domains*
- *$dom : \mathbb{A} \to \mathbb{D}$ is a function assigning a data domain to each attribute*
- *$att : \mathbb{O}_{\mathbb{T}} \to A \subseteq \mathbb{A}$ is a function assigning a set of attributes to each data-object type.*
- *$type : \mathbb{O} \to \mathbb{O}_{\mathbb{T}}$ is a function assigning a data-object type to each data object.*

We further define:

- *Each data object has a key that is confined to its type. A key is a set of attributes that uniquely identifies a data object in the set of objects of its type. Elements of the key are underlined. In the following, we refer to a data object o as $type(o).key$. Elements of key are separated by dots.*
- *A data value (dv) is a single value in the domain of an attribute.*

Example 1. *Consider Process* SMR auction*. There, the process domain may be:*
$\mathcal{D} = (\mathbb{T}, \mathbb{E}, \mathbb{O}, \mathbb{O}_{\mathbb{T}}, \mathbb{A}, \mathbb{D}, dom, att, type)$ with:

- $\mathbb{T}$:= { *decrease eligibility, update price, ..*}
- $\mathbb{E}$:= { *start, end, .. }*
- $\mathbb{O}$:= { *bidder.1, .., product.1, .., bid.2.1, ... }*
- $\mathbb{O}_{\mathbb{T}}$:= { *bidder, product, bid,.. }*
- $\mathbb{A}$:= { *bidderID, capacity, productID, winner, price, ...*}
- $\mathbb{D}$:= { *{0, 1, 2}, {1, 2}, [1; 10],* $\mathbb{N}$ = *{1, 2, ...}, ... }*
- *dom(bidderID) := dom(productID) := {1, 2}; dom(winner) := {1, 2} ; dom(capacity) := {0, 1, 2}; dom(price) := [1; 10]; ...*
- *att(product):={productID, winner,..}; att(bidder):={bidderID, capacity,..}; att(bid) := {bidderID, productID, price ...} ; ...*
- *type(bidder.1) := type(bidder.2) := bidder; type(product.2) := product; ...*

To illustrate further, we refer to an object of type bid with bidderID = 1 and productID = 2 as $bid.1.2$.

A process graph commonly represents a process model. This formalism allows to be independent from a specific high-level process-modeling language. We now provide a general definition of process graphs in Definition 4. We follow the graphical notation of BPMN.

Definition 4. *(Process Graph) Let* $\mathcal{D} = (\mathbb{T}, \mathbb{E}, \mathbb{O}, \mathbb{O}_{\mathbb{T}}, \mathbb{A}, \mathbb{D}, dom, att, type)$ *be a process domain. A Process Graph is a tuple* $P = (N, F, O, I, atype, etype)$ *where:*

- $N = T_P \cup E_P \cup G_P$ *is a finite set of nodes that is partitioned into the set of activities* T_P*, the set of events* E_P*, and the set of gateways* G_P*.*
- $F \subseteq N \times N$ *represents the sequence flow relation between nodes.*
- $O \subseteq \mathbb{O}$ *is a finite set of data objects.*
- $I \subseteq O \times N \cup N \times O$ *is a finite set of arcs connecting data objects to nodes, so-called data associations.*
- $atype : T_P \rightarrow \mathbb{T}$ *is a function assigning an activity type to an activity in* P*.*
- $etype : E_P \rightarrow \mathbb{E}$ *is a function assigning an event type to an event in* P*.*

Moreover, we use the following notation:

- *InputSet(n) is the set of data objects having an arc to node* $n \in N$*,*
- *OutputSet(n) is the set of data objects having an arc from node* $n \in T_P \cup E_P$*.*

IV. Our Verification Approach

Figure 2 is an overview of our approach. The gray boxes indicate the contributions of this paper. Task *enhance process model* enriches the designed process model by specifying the usage of data values and their modifications. To this end, a specification will make use of our new Data-Value Enhancement Functions (EF), see Section IV-A. Task *data-value-aware transformation* transforms the enhanced process model to a Petri Net. To do so, our approach generates subnets reflecting the semantics of both the preconditions and the effects on data values, see Section IV-B. Task *specify data-value centered properties* specifies properties as CTL formulas on the resulting Petri Net, Section IV-C. In the last step, Task *model checking* checks the properties with an off-the-shelf model checker.

A. Process Enhancement with Data Values

Data values used in a process influence the process execution. In particular, the conditions of gateways determine the branch the process will follow. These conditions use the values of data objects which preceding activities have assigned.

Example 2. *In the SMR auction process model, a gateway with condition bidder's capacity points > 0 checks the capacity rule of the SMR auction. The gateway takes the decision based on the value of bidder's capacity points. However, Activity decrease capacity can modify this value during the process. This modification may affect the execution of the process.*

To take the usage of data values into account, we allow to specify data values as *preconditions* as well as *effects* of a process element, as follows. We refer to the preconditions as *data-value conditions*. We deal with the effects of process elements on data values with so-called *data-value functions*.

Definition 5 (Data-Value Condition)**.** *Let* P *be a process graph from Data Domain* $\mathcal{D}$*. A Data-Value Condition is an expression of the form* $(a \otimes dv)$ *where* a *is an attribute in* P*,* dv *is a value in the domain of Attribute* a*, and* $\otimes := \{=, \neq, <, >, \leqslant, ...\}$*. We define* $DVConds$ *as the set of all data-value conditions with the elements of Process Graph* P*.*

Definition 6 (Data-Value Function)**.** *Let* P *be a process graph from Data Domain* $\mathcal{D}$*. A Data-Value Function is a function with attributes of* P *as input and output parameters. The types of the parameters are from a domain in* $\mathbb{D}$*. We define* $DVFuncs$ *as the set of all data-value functions with elements of* P*, i.e.:*
$DVFuncs : \times_{i=1}^{n} D_i \rightarrow \times_{j=1}^{m} D_j$*, where* $D_i, D_j \in \mathbb{D}$*.*

Example 3. *Consider Process Model* P *in Example 1. There, we have the following data-value conditions and data-value functions:*
$DVConds := \{(capacity \geqslant 1), ...\}$*,* $DVFuncs := \left\{ f_1(dv) = \begin{cases} dv - 1 & dv \geqslant 1 \\ 0 & dv < 1 \end{cases}, ... \right\}$
Suppose that data value dv *in function* f_1 *belongs to Attribute* $price$ *of Data object* $product.1$*. Then function* f_1 *reduces the price of this product by 1.*

So far, process models do not allow to specify conditions on data values or modifications. So we propose so-called *Data-Value Enhancement Functions* (EF) to annotate process graph elements of P with the usage of data values.

Definition 7. *(Data-Value Enhancement Functions) Let* P *be a process graph from Data Domain* $\mathcal{D}$*. Data-Value Enhancement Functions are a set of functions to enhance process graph elements of* P *as follows:*

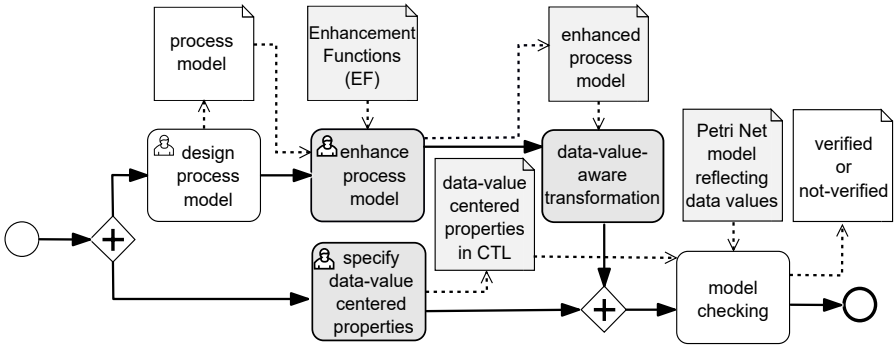


Fig. 2. Overview of our approach on verification of data-value-aware process models

EnhancementO : $O \rightarrow (\mathbb{A}, pk)$ *is a function to enhance a data object* $o \in InputSet(t) \cup OutputSet(t) \cup InputSet(g)$*, where* $t \in T_P \cup E_P$*,* $g \in G_P$ *with attributes* $A \subseteq att(o) \subseteq \mathbb{A}$*. In this,* pk *is the key value of Data object o. It allows to deal with several data objects from the same type.* O_{enh} *is the set of data objects in* P *enhanced with EnhancementO.*

EnhancementA : $T_P \cup E_P \rightarrow DVFuncs$ *is a function to enrich activities and events in* P*. It assigns a data-value function* $f : I \rightarrow O$ *to an Activity/Event* t *with* $InputSet(t) \subseteq O_{enh}$ *and* $OutputSet(t) \subseteq O_{enh}$*.* I *is the domains of the enhanced data objects in* $InputSet(t)$ *and* O *is the domains of the enhanced data objects in* $OutputSet(t)$*. Activity/Event* t *reads* $InputSet(t)$ *and applies Function* f*. It writes the output parameters of Function* f *to* $OutputSet(t)$*.* A_{enh} *is the set of activities and events in* P *enhanced with EnhancementA.*

EnhancementG : $G_P \rightarrow DVConds$ *is a function which assigns a data-value condition* $\in DVConds$ *to a gateway* g *with* $InputSet(g) \subseteq O_{enh}$*. The data-value condition specifies a decision formula based on the attributes belonging to an enhanced data object in* $InputSet(g)$*.* G_{enh} *is the set of gateways in* P *enhanced with EnhancementG.*

Example 4. *Figure 3(a) shows the enhancement of Data object product with EnhancementO. The enhancement specifies Attributes price and winner of Data object product.1. Figure 3(b) shows the enhancement of Activity decrease to reduce the price of product.1. Activity decrease reads the price of* $product.1$ *and writes the modified value to the same data object. Figure 3(c) illustrates an enhanced gateway to check if the capacity of* $bidder.1$ *is greater than zero.*

Process-graph elements form the core of many process specification languages such as BPMN or BPEL. As we only enhance the process-graph elements, one has to add a certain number of workflow elements to the process graph in order to model iterative or multi-instance subprocesses of high-level process modeling languages. To illustrate, all bidders in an SMR auction have the same *bidding* subprocess, consisting of process-graph elements. To model this subprocess without iterative subprocesses, one needs to add the subprocess separately for each single bidder into the process model. This leads to more BPMN elements. For a discussion of the number of additional elements see Section V.

B. Transformation of Data-Value-Aware Processes to Petri Nets

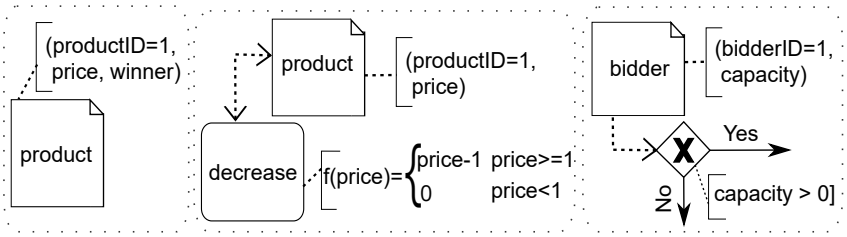
To facilitate rigorous analyses, various approaches transform process models to Petri Nets. [14] is a survey of transformations from process models to Petri Nets. However, all these approaches do not transform data-value-aware processes, let alone ones that modify data values. So we propose a new approach to transform a data-value-aware process to a Petri Net, see Algorithm 1.

First we transform the control flow, regardless of the data objects, with the rules in [15] (Line 2). However, the resulting "control flow" Petri Net does not reflect the semantics of the data value usages. To overcome this problem, our algorithm now works in two parts. The first part, the so-called *mapping*, generates new places corresponding to data values used in the process (Lines 3, 4). The second part creates subnets for the process elements which use data values that extend the already generated Petri Net. This is called *unfolding* (Lines 5, 6).

Algorithm 1 Transformation: From a Data-Value-Aware Process to a Petri Net

```
1: procedure ProcessModelPetriNet(P)
2:     Petri Net pn ← transform control flow of P
3:     for all data value dv in P do
4:         pn.place ← new place p.dv                ▷ Mapping
5:     for each element ∈ {A_enh ∪ G_enh} do
6:         subnet ← CreateSubNet(element, pn)
7:         extend pn with subnet                    ▷ Unfolding
8:     return pn
```

1. Mapping. The mapping generates a new place for each data value which belongs to a data object in Process Graph P (Lines 3, 4). The number of data objects for each data-object type in P depends on the domain of *key*s. For example, if the key of Data-object type *bidder* has a domain of $\{1, 2\}$, it generates two sets of places, one set for $bidder.1$, another



(a) EnhancementO (b) EnhancementA (c) EnhancementG

Fig. 3. Example of enhanced process graph elements

set for $bidder.2$. These new places are called *data-value place*s. A data-value place has a name with the following structure: *p.*[*object*].*pk.*[*attribute*].*dv*. Here, [*object*]/[*attribute*] is the name of the data-object type/attribute the data value belongs to. *pk* is the *key value* of the data object, and *dv* is its data value. The naming allows to identify places relevant for the definition of properties.

Example 5. *Suppose that data-object type bidder has Attributes capacity with domain* $\{0, 1, 2\}$ *and bidderID as the key with domain* $\{1, 2\}$. *The mapping generates 6 places:*

$\{$*p.bidder.1.capacity.0, p.bidder.1.capacity.1, p.bidder.1.capacity.2, p.bidder.2.capacity.0, p.bidder.2.capacity.1, p.bidder.2.capacity.2*$\}$.

2. Unfolding. For every enhanced element of the control flow, i. e., the elements of $A_{enh} \cup G_{enh}$, we create a new subnet, to reflect the semantics of the use of data values (Lines 5-7 of Algorithm 1). To do so, we propose Algorithm 2 described next. It creates different subnets depending on whether the enhanced element is an activity/event or a gateway.

a) Creating a subnet for an enhanced activity/event *element*: Observe that the element is associated with a data-value function. The algorithm creates a new transition for every combination of data values belonging to the enhanced data objects of $InputSet(element)$ (Lines 4-6). Each of these transitions has the corresponding data-value places as *input places* (Line 7). The algorithm computes the output parameters of the data-value function (Line 8). The transition created has the data-value places of the output parameters as its *output places* (Line 9). All transitions have an outgoing arc to Place *p.end* (Line 10).

Example 6. *Fig. 4(a) shows the subnet created for the enhanced activity of Fig. 3(b). The activity decreases the price of product.1 by 1. The gray places and transitions already exist as a result of the transformation of the control flow (without data aspect) into a Petri Net. The algorithm creates a separate transition for each data value* dv *in the domain of Attribute* $price$ *of Data object product.1,* $[1; 10] \cap \mathbb{N}$. *Each of these transitions gets the corresponding data-value places as input places, for instance p.product.1.price.10. The transitions also get the data-value places of the output parameter of Function* f *as output places, e. g., p.product.1.price.9. In the subnet generated, we handle all places representing the input parameter values (possible values are 1 to 10) by a transition and yield a resulting value decreased by 1.*

b) Creating a subnet for an enhanced gateway *element*: Observe that the element is associated with a data-value condition. We create a single transition for each data value specified in a data-value condition for an enhanced data object in the $InputSet(element)$ (Lines 11, 13). Each of these transitions has the corresponding data-value places as its *input place* (Line 14). Since the result of the data-value condition is either *True* or *False*, we create two new places for the result: *p.True* and *p.False*. Depending on whether the data value represented by the *input place* of a transition satisfies the data-value condition, the *output place* of the transition is Place *p.True* or *p.False* (Lines 15, 16).

Example 7. *Fig. 4(b) shows the subnet created for the enhanced gateway in Fig. 3(c). For all data values of Attribute capacity of Data object* $bidder.1$*, the algorithm creates a new transition t. Each data-value place is at the same time an input and output place of transition t, because it has to keep its data value.*

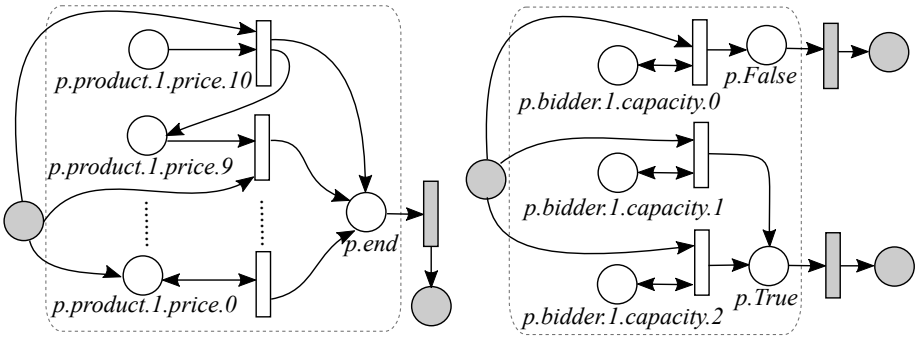
C. Specification of Properties

The result of the transformation is a Petri Net of the process reflecting the semantics of the use of data values. To verify the Petri Net created, one must specify data-value centered properties in a formal language such as CTL. In the following, we say how properties referring to data values can be defined in CTL.

Definition 8. *(Data-Value Centered Property) Let* P *be a process graph. A Flow Condition* FC_ϕ *is a set of Activities or Events in* P. *A Data Condition* DC_ϕ *is a set of data values in* P. *A Data-Value Centered Property is a CTL formula whose atomic formulas each refer to the marking of a corresponding place of an element in* FC_ϕ *or* DC_ϕ.

Example 8. *Consider the following question: "Can* $bidder.1$ *win* $product.2$ *at a price of 8 at the end of the auction?" – The respective data-value property is:*

$$EF(p.product.2.price.8 \wedge p.product.2.winner.1 \wedge e.end)$$



a) created subnet for an enhanced activity b) created subnet for an enhanced gateway

Fig. 4. Example of created subnets

Algorithm 2 Create Subnet

```
1: procedure CREATESUBNET(element, pn)
2:     create Petri Net subnet
3:     if element ∈ A_en then
4:         domElement ← dom(InputSet(element))
5:         for all combination of dv⃗ ∈ domElement do
6:             subnet.transition ← new transition t
7:             t.inputPlace = pn.getPlace(dv⃗)
8:             dv⃗_result ← element.dvFunction(dv⃗)
9:             t.outputPlace = pn.getPlace(dv⃗_result),
10:                            subnet.getPlace(p.end)
11:    else element ∈ G_enh
12:        for all dv ∈ dom(InputSet(element)) do
13:            subnet.transition ← new transition t
14:            t.inputPlace = pn.getPlace(dv)
15:            element.dvCondition(dv) == true ?
16:            t.outputPlace = p.True : p.False
17:    return subnet
```

In this formula, $p.product.2.winner.1 \in DC_\phi$ shows bidder.1 wins product.2. $p.product.2.price.8 \in DC_\phi$ shows the price of 8 for product.2 and $e.end \in FC_\phi$ is the result of the transformation of an end event, i. e., it represents the end of the process.

Example 9. *"Can all bidders win at least one product?" translates to:*

$$EFAG((p.product.1.winner.1 \vee p.product.2.winner.1) \wedge (p.product.1.winner.2 \vee p.product.2.winner.2))$$

Example 10. *"Can $product.1$ be sold for the price of 2?" translates to:*

$$EF(p.product.1.price.2 \wedge e.end)$$

Verifying the property of Example 10 allows to find the lowest revenue of the auctioneer (R_{lowest}). To do so, we first find the lowest final *price* of *products*, starting with the reserve prices as in Example 10. In case the property is not satisfied, we now verify a property with an increased price, until there is a state that fulfills the property. In the next step, we find the lowest combination of prices in a similar way, starting with the lowest final price of products. Suppose that the lowest price for *product.1* and *product.2* is 4 and 6, respectively. Then the first step is to verify:

$$EF(p.product.1.price.4 \wedge p.product.2.price.6 \wedge e.end)$$

If this verification fails, we perform the verification for other combinations of increased prices. For this, we do not have to look at all prices in combination, but only at those between a maximum and a minimum possible price. The maximum price is known by the maximum budget of a bidder for a certain product, and the minimum is the result of a verification as explained in Example 10. More sophisticated search strategies can restrict combinations to be verified further. This is future work. By verifying the combination of prices, we find the lowest auctioneer revenue possible.

Next, one can find the minimum profit of the bidder (P_{lowest}) associated with R_{lowest}. We check if bidders with the lowest budget can be winners. Suppose that *bidder.1* and *bidder.2* have the lowest budget for *product.1* and *product.2*, respectively. The first formula to check is:

$$EF((p.product.1.price.4 \wedge p.product.2.price.6) \wedge (p.product.1.winner.1 \wedge p.product.2.winner.2) \wedge e.end)$$

If the model checker cannot find a state, i. e., an execution path fulfilling the property, we change the winners. This results in new formulas to be checked. This is done until the formula is satisfied.

One can also find the lowest market *efficiency* (E_{lowest}) while the auctioneer's revenue is minimal. To define efficiency, we use a measure described in [4]. It quantifies how the surplus resulting from the worst allocation (S_{lowest}) can deviate from the surplus with a random allocation S_{random}:

$$E_{lowest} = \frac{S_{lowest} - S_{random}}{S_{optimal} - S_{random}} \times 100 \text{ percent}$$

Here, the surplus resulting from an optimal allocation ($S_{optimal}$) is computed by giving the products to bidders with the highest budget. S_{lowest} is the sum of the lowest revenue and profit.

V. Implementation and Evaluation

We have implemented a framework to verify data-value centered properties in SMR auctions. Its input of our framework is a process model in BPMN format. To transform the process model into a Petri Net, we have implemented Algorithm 1. The generated Petri Net is verified against properties specified as CTL formulas. To do this, we use the state-space generation and model checking algorithm of the LoLA framework [16].

Evaluation setting. To evaluate our framework, we first model and enhance the SMR auction process including eligibility and capacity rules. We compute the auction measures for three different settings.

- *Process 1*: Three bidders compete to win two products.
- *Process 2*: Two bidders compete to win three products.
- *Process 3*: Four bidders compete to win six products – the size of the German 4G spectrum auction in 2010 [17].

In all settings, we have assigned a random budget to each bidder for a certain product in the range of $[2 - 10]$, similarly to [4]. We also have defined a reserve price of 1 for all products, so all bidders can afford all products at the beginning of the auction. All bidders have a *capacity point* and an *eligibility point* of 2. In this evaluation, we first detect the lowest *revenue*, *bidder's profit* and *efficiency of auction*. Then we analyze the run time of the verification process.

Enhancement of SMR process model. We have modeled and enhanced the three processes with BPMN. Note that the BPMN activities (without subprocesses), events and gateways directly refer to the process graph representation. Table I lists the numbers of BPMN elements after the enhancement. As explained, we have to add a certain number of BPMN elements when increasing the number of bidders or of products. For example, to model the *bidding* subprocess not as a multi-instance subprocess, one must add the same subprocess for each single bidder separately. This leads to more BPMN elements of the model, by a constant factor. To illustrate, adding a new bidder increases the number of activities and gateways of the BPMN model by 15 and 27, respectively.

Transformation of SMR process model into Petri Net. We have used the enhanced SMR BPMN models with the characteristics listed in Table I as input to our transformation into Petri Nets. With this transformation, a new bidder increases the number of bidder objects by 1, and the possible values for bids and winners increase as well. So the result of the transformation is a larger Petri Net because of the unfolding of subnets due to the usage of data values. This can lead to higher verification times and state-space explosion. We will observe the limits of our current approach due to this effect with Process 3.

Verification of SMR process model. We have verified the data-value centered properties to compute the auction measures. This has been possible for Processes 1 and 2, but not for Process 3, because of (the expected) state-space explosion, so the remaining discussion focuses on those two processes. To detect the lowest possible revenue, one must verify all possible combinations of product prices. *product.1* and *product.2* have 8 respectively 9 different possible prices, based on the budgets of the bidders. This requires to check 8×9 properties in Process 1. Using the same procedure for Process 2 requires to check 432 properties. However, if we limit the range of product prices by a minimum and maximum value (see Section IV-C), the number of combinations of prices/properties is restricted to only 4×5 and $4 \times 6 \times 1$ for Processes 1 and 2 respectively. See Table III. Our verification of Process 1 reveals that the auctioneer's revenue is minimal when assigning *product.1* and *product.2* to *bidder.1* for the price of 5. In this case, the efficiency of the auction and bidder's profit are 100% and 7, respectively. In Process 2, *product.1* has been sold to *bidder.2* for the price of 5, although both bidders would have had more budget. Here, the auctioneer's revenue is 15. So the efficiency is 82.60%. This is interesting: Even for small auctions, the conventional SMR design has inefficiencies, and our approach can reveal them. So an auction designer can now take this format as a starting point for improvement, by also applying our method to new designs and compare results. Put differently, there is a value in verifying small auctions.

Performance of verification. We have studied the run time of verifying properties and the ones of transforming enhanced process models into Petri Nets. We have verified 24 respectively 20 properties – the combinations of possible prices limited by a minimum and maximum value, for Processes 1 and 2, respectively. Verification of a single property takes less than 4 minutes in Processes 1 and 2, using a computer with 16 GB main memory and 2 Intel 3 GHz CPUs. Transformation of Processes 1 and 2 into Petri Nets has been within less than 1 second, see Table II. However, our evaluation attempts of Process 3 have revealed the limitations of a generic approach to Petri Nets transformation. This calls for further work, which is beyond the scope of this current paper, see Section VII.

VI. Related Work

There is a growing body of research addressing the data perspective of workflow verification, see [18] for an overview. [19] is one of the first approaches based on Petri Nets to detect and resolve modeling errors which occur in the presence of data. They transform BPMN 1.2 to Petri Nets by mapping options for data objects. This approach only considers data flow anomalies of whole objects but do not allow for verifying arbitrary properties, which are based on values of the data objects in the process. [7] features another approach to detect data anomalies based on Petri Nets. They unfold the execution semantics of BPMN process models regarding data and formalized data-flow errors in a set of

TABLE I
NUMBER OF ENRICHED BPMN ELEMENTS FOR SMR AUCTIONS WITH DIFFERENT SIZE

Process	Activities	Gateways	Annotations	Data associations
2 bidders & 2 products	50	55	145	94
3 bidders & 2 products	65	82	189	105
2 bidders & 3 products	69	81	206	119
4 bidders & 6 products	156	213	744	216

TABLE II
THE RESULTS OF TRANSFORMATION AND VERIFICATION OF DIFFERENT SMR AUCTION SIZES

Process	Places	Transitions	Transformation	No. Properties	Verification(longest)
Process 1	589	715	0.3 sec	20	2 min and 09 sec
Process 2	869	1123	0.4 sec	24	3 min and 20 sec
Process 3	4979	4876	1.6 sec	-	-

TABLE III
VERIFICATION OF PROPERTIES IN PROCESSES 1 AND 2

Process	Product	Winner	Price	Revenue	Profit	Efficiency
Process 1	product.1	bidder.1	5	10	7	100%
	product.2	bidder.1	5			
Process 2	product.1	bidder.1	5	15	4	82.60%
	product.2	bidder.2	4			
	product.3	bidder.1	6			

anti-patterns. They only consider the data-flow errors and do not verify data-value centered properties. [20] formalizes data-aware compliance rules by Linear Temporal Logic with Past Operators and employs a model checker to verify properties. When detecting an error, they apply Temporal Logic Querying (TLQ) to explain the violation. However, they cannot support conditions based on numeric data states in their process model. Then, verification of properties based on data values is not possible. Another approach to deal with conditions on numeric data is [21]. They verify temporal properties of a business process by employing bounded model checking in answer set programming. The article addresses the verification of properties with data values. However, the challenge of modifications of data values remains.

The authors of [22] propose a methodology to verify the manipulation of data objects. They define a domain specific language (DSL) to describe the manipulation of attributes of data objects. To this end, they annotate the process model and check whether the possible evolution of a data object during the process may create or update an object erroneously. In particular, they verify *completeness* and *consistency* of data objects. However, manipulation of data values by functions is not covered. Next, they can verify consistency of only one data object, i. e., verification of properties defined on a combination of states of different objects is not possible. In [23] the authors combine a declarative specification of the process model with a specification of a (relational) data scheme as well as with a communication perspective. They propose an Object-Centric behavioral Constraint modeling language and apply it to process mining – but not to verification. Our approach for modelling the data of the process is similar, but we extend it with the enhancement functions and conditions to prepare it for verification.

In [13] the authors provide an abstraction technique of the data domains that finds intervals utilizing the conditions of gateways. They do not specify modifications of data values. In particular, they specify conditions of XOR gateways based on data states, but they do not specify how the process modifies the data values yielding the data states. So this information gets lost. Hence, they can only verify properties in process models which modify data values on that coarse, abstracted data state.

Model checking with data has some tradition in the data-centric community. In [24] they present an approach to apply data-centric verification to activity-centric processes. However, they focus on reachability analysis, and do not support checking of arbitrary logic formulas as in our approach. Other data-centric approaches like Active XML [25] and DCDS [26] also handle verification of business processes. However, they lack an intuitive control-flow perspective. A so-called artifact system [27] features a restricted class of artifact systems and LTL-FO properties to make the satisfaction decidable. There, artifacts carry values of an attribute which external services can update. These services work with an underlying database. However, their approach does not work in the presence of data dependency (integrity constraints on the database). They do not support arithmetic operations, which play a significant role in applications such as spectrum auctions. [28] alleviates these problems by extending the artifact model. However, they impose syntactic restrictions on process models and properties which limit applicability.

A recent method [29] provides a parameterized verification of data-aware BPMN called DAB. There, the process affects data in three possible ways: *Insert&Set*, *Delete&Set* and *Conditional update*. By encoding DABs in an array-based artifact system framework, SMT-based model checking of array-based systems [30] can be used to verify safety properties of DABs. However, this approach can only specify updating single

data values possibly under conditions by so-called conditional update specifications. Functions that modify data values, e.g., *increase price* in our use case, cannot be expressed in a DAB. While the verification is decidable by imposing restrictions on the size of arrays, it also cannot provide counterexamples.

VII. Conclusions

This article has featured a new generic method to verify data-value-aware processes. – as is the case with SMR spectrum auctions and with other settings. We have proposed so-called enhancement functions, to formulate annotations specifying the usage of data values. To leverage existing model checking approaches and tools, we then have transformed the enhanced process model into Petri Nets. To this end, we have proposed a new algorithm considering data values and their modifications. So one can freely define data-value centered properties as CTL formulas for verification. Our verification efforts for various settings of SMR auctions have been successful. They have pointed to inefficiencies of existing auction designs. This holds even though we have only been able to verify slightly smaller settings than ones having taken place in reality. The restriction just mentioned is due to a well-known severe limitation of any verification approach based on model checking, namely state-space explosion. This calls for research on reducing Petri Nets, using domain knowledge of data-value centered processes. As future work, we target at such a reduction, taking inspiration from [31]. The reduction proposed in [31] is however confined to data flows of objects in processes and not on data values, i. e., it does not solve this current problem. All in all, our research not only is an advancement in the field of process verification, but also a valuable contribution on auction research.

References

[1] Engelmann, D., Grimm, V.: Bidding behaviour in multi-unit auctions–an experimental investigation. The Economic Journal **119**(537), pp. 855–882 (2009)

[2] Wolfstetter, E.: The Swiss UMTS spectrum auction flop: Bad luck or bad design. No. 534. Available at SSRN: https://ssrn.com/abstract=279683 (2011)

[3] Ausubel, LM., Cramton, P., et al. The clock-proxy auction: A practical combinatorial auction design. Handbook of Spectrum Auction Design, pp. 120–140. (2006)

[4] Brunner, C., Goeree, J. K., et al.: An experimental test of flexible combinatorial spectrum auction formats. Microeconomics **2**(1), pp. 39–57 (2010)

[5] Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems: Specification. Springer Science and Business Media (2012)

[6] Van der Aalst, W. M. A.: The application of Petri nets to workflow management. Journal of circuits, systems, and computers **8**(01), pp. 21–66 (1998)

[7] Von Stackelberg, S., et al.: Detecting data-flow errors in BPMN 2.0. Open Journal of Information Systems (OJIS) **1**(2), pp. 1–19 (2014)

[8] Clarke, E. M., Emerson, E. A., Sistla, A. P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems 8 (TOPLAS) **8**(2), pp. 244–263 (1986)

[9] Clarke, E. M., et al.: Model Checking. MIT Press, Cambridge, USA, (1999)

[10] Business Process Model and Notation, http://www.omg.org/spec/BPMN/2.0/PDF. Last accessed 2019

[11] Milgrom, P., et al.: Putting auction theory to work. Cambridge University (2004)

[12] Kwasnica, A., Sherstyuk, K.: Multiunit auctions. Journal of Economic Surveys. **27**(3), 461-490 (2013)

[13] Knuplesch, D., et al.: On enabling data-aware compliance checking of business process models. Springer, Berlin, pp. 332–346 (2010)

[14] Lohmann, N., et al.: Petri net transformations for business processes -– a survey. In Transactions on Petri Nets and other Models of Concurrency II, Springer, Berlin, Heidelberg, pp. 46–63 (2009)

[15] Dijkman, R. M., et al.: Semantics and analysis of business process models in BPMN. Information and Software Technology **50**(12), pp. 1281–1294 (2008)

[16] Schmidt, K.: LoLA a low level analyser. Intl. Conf. on Application and Theory of Petri Nets, Springer, Berlin, Heidelberg, pp. 465–474 (2000)

[17] Cramton, P., Ockenfels, A.: The German 4G spectrum auction: Design and behaviour. The Economic Journal. Vol. 127, Issue 605, F305–F324 (2017)

[18] Dell'Aversana, R.: Verification of data aware business process models: A Methodological Survey of Research Results and Challenges. 12th Intl. Conf. Distributed Computing and Artificial Intelligence, Springer, Cham, pp. 393–397 (2015)

[19] Awad, A., et al.: Diagnosing and repairing data anomalies in process models. Intl. Conf. Business Process Management, Springer, Berlin, Heidelberg, pp. 5–16 (2009)

[20] Awad, A., et al.: Specification, verification and explanation of violation for data aware compliance rules. Service-oriented Computing, Springer, Berlin, Heidelberg, pp. 500–515 (2009)

[21] Giordano, L., et al.: Business process verification with constraint temporal answer set programming.Theory and Practice of Logic Programming **13**(4-5), pp. 641–655 (2013)

[22] Pérez-Álvarez, J. M., Gómez-López, M. T., Eshuis, R., Montali, M., Gasca, R. M. (2020). Verifying the manipulation of data objects according to business process and data models. Knowledge and Information Systems. https://doi.org/10.1007/s10115-019-01431-5

[23] Li, G., Medeiros de Carvalho, R., van der Aalst, W. M. P. (2019). Object-Centric Behavioral Constraint Models: A Hybrid Model for Behavioral and Data Perspectives. Proc. SAC'19, April, ACM, pp. 48–56 (2019)

[24] De Masellis, R., et al.: Add data into business process verification: Bridging the gap between theory and practice. Proc. Artificial Intelligence. pp. 1091–1099 (2017)

[25] Abiteboul, S., et al.: Modeling and verifying active XML artifacts. IEEE Data Eng. Bull., IEEE (2009)

[26] Bagheri, B., et al.: Verification of relational data-centric dynamic systems with external services. Proc. ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Proc. ER, LNCS 6412, Springer, Berlin, pp. 163–174 (2013)

[27] Deutsch, A., et al.: Automatic verification of data-centric business processes. Proc. BPM, LNCS 6896, Springer, Berlin, pp. 252–267 (2009)

[28] Damaggio, E., Deutsch, A., Vianu, V.: Artifact systems with data dependencies and arithmetic. ACM Transactions on Database Systems (TODS) **37**(3), No. 22 (2012)

[29] Calvanese, Diego et al. (2019). Formal modeling and SMT-based parameterized verification of data-aware BPMN. Proc. BPM, Springer, LNCS 11675, pp. 157–175 (2019)

[30] Ghilardi, S., Ranise, S.: Backward reachability of array-based systems by SMT solving: termination and invariant synthesis. Log. Methods Comput. Sci. 6(4), 1–48 (2010)

[31] Mülle, J., Tex, Ch., Böhm, K.: A practical data-flow verification scheme for business processes. Information Systems **81**(March), pp. 136–151 (2019)