Deriving Business Process Data Architectures from Process Model Collections

Rami-Habib Eid-Sabbagh, Marcin Hewelt, Andreas Meyer, and Mathias Weske

Hasso Plattner Institute at the University of Potsdam
{rami.eidsabbagh,marcin.hewelt,andreas.meyer,
mathias.weske}@hpi.uni-potsdam.de

Abstract. The focus in BPM shifts from single processes to process interactions. Business process architectures were established as convenient way to model and analyze such interactions on an abstract level focusing on message and trigger relations. Shared data objects are often a means of interrelating processes. In this paper, we extract hidden data dependencies between processes from process models with data annotations and their object life cycles. This information is used to construct a business process architecture, thus enabling analysis with existing methods. We describe and validate our approach on an extract from a case study that demonstrates its applicability to real world use cases.

1 Introduction

The last decade has seen widespread adaptation of business process management resulting in large process model collections. Although process models often need to interact to deliver services or produce goods, they generally are elicited in isolation, prone to miss the interdependencies with other processes. Business process architectures (BPAs) have been proposed to provide an abstracted view on interrelated process models (see [1] for a survey of BPA approaches). Our BPA approach [2, 3] relates processes by trigger and message flows, allows for multiple process instances, n-to-m communication, and offers formal verification.

While previous work dealt with modeling and verification of BPAs, this contribution extracts data dependencies between processes. Even if the interaction of processes is not modeled explicitly, process relations can be deduced by looking at the data objects and how processes manipulate them. If one process produces a certain data object which another process consumes, then those processes have to be carried out in sequence. We summarize identified dependencies for several data objects in a *process data dependency matrix (PDM)* and use it to construct a *business process data architecture (data BPA)*, which reveals and depicts hidden data-related interdependencies and thus eases the management of process model collections. The resulting data BPA can be verified with the method presented in [3] unveiling erroneous process interaction due to data.

Aside BPAs, several techniques have been introduced to depict business process interactions, e.g., choreography diagrams in BPMN [4], service interaction patterns [5], or Proclets [6]. The mentioned alternatives focus on the interaction behavior between processes or services by modeling the message exchange. A BPA represents relations between processes giving them a partial ordering with respect to process execution.

S. Basu et al. (Eds.): ICSOC 2013, LNCS 8274, pp. 533–540, 2013.

In contrast to all aforementioned techniques, we utilize relations between data objects utilized in multiple processes to determine the relations between these processes. This naturally leads to the object-centric process modeling paradigm, e.g., [7], where a process is modeled by the involved data objects and synchronization follows from data state changes represented in object life cycles. This synchronization between multiple object life cycles induces the partial ordering of all data object changes and therefore the execution. Although architectures can be build based on this paradigm by synchronizing object life cycles, no approach exists describing this procedure. Additionally, the control flow view is only provided implicitly, whereas the combination of our approach and our earlier work shows both perspectives. A first step towards integration of control flow and data was presented by Fahland et al. [8], who utilize Proclets to model object-centric processes.

2 Scenario

Throughout the paper, we will use the following scenario taken from a case study on order and delivery processes. Figures 1 to 3 present the five process models of our scenario. Process p_1 (Fig. 1) first analyses a verified order and checks if the ordered products are in stock. If this is the case, they are shipped, the invoice is sent, and the payment is received. Otherwise the products are built. Process p_2 (Fig. 2a) is similar to process p_1 in Fig. 1, except that it sends the product only after the payment was received and allows to reject an unconfirmed order. Process p_3 (Fig. 2b) archives a processed order. Process p_4 (Fig. 3a) verifies the customer and may reject his order. Finally, process p_5 (Fig. 3b) handles invoicing and payment.

These processes utilize the two data objects "order" and "product". Hence two object life cycles (OLCs) are required. An OLC describes the manipulations allowed to be performed upon the corresponding data object. The utilized OLCs will be presented as part of the description of our approach in Section 4. All process models satisfy the notion of weak conformance [9] with respect to the utilized data objects, i.e., process models and OLCs do not contradict.

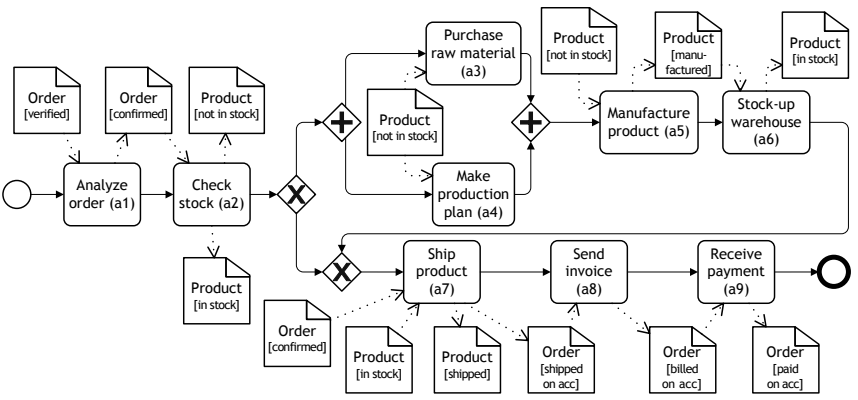
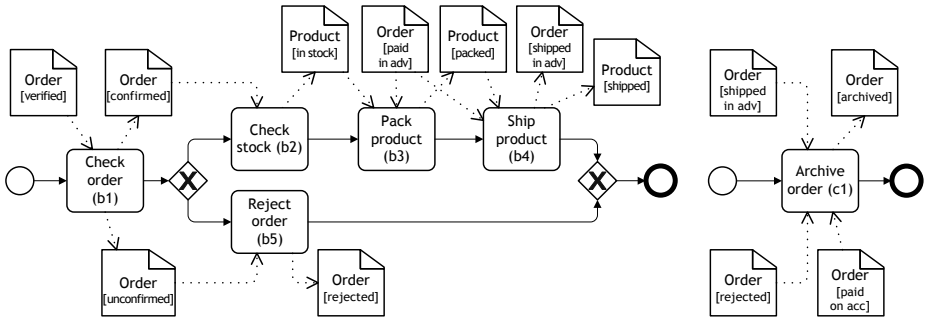
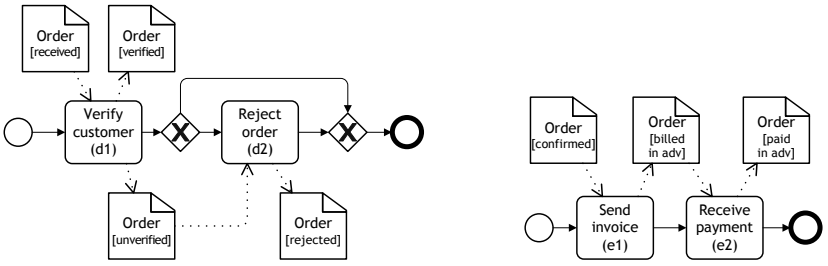


Fig. 1. Process model p_1: Order handling, delivery on account (on acc.)



(a) Process p_2: Order handling, payment in advance (in adv.) (b) p_3: Archiving

Fig. 2. Process models p_2 and p_3



(a) Process p_4: Customer verification (b) Process p_5: Invoice handling

Fig. 3. Process models p_4 and p_5

3 Foundations

Data in process models can be represented by a set of data objects and their states. A data object is an entity processed during process execution and is characterized by states and state transitions. At any point in time, a data object is in exactly one state which constitutes a business situation. We formalize data objects as *object life cycles* (OLCs) which consist of a set of states S, a set of state transitions $T \subseteq S \times S$, an initial, and a final state. Note, that we consider only *acyclic* OLCs for now.

A process model consists of a set of control flow nodes and a relation defining the partial ordering of activities performed during process execution. Further, activities can be annotated with data objects, which represent pre- and postconditions for activities and determine when an activity is enabled (in addition to control flow) as well as their expected outcome. For each process model, we assume that it is structurally sound and that it weakly conforms to all utilized data objects [9], i.e., for each data object modification by an activity there exists a corresponding path in the OLC of this object. We assume that all activities' data object accesses are modeled and that no activity writes a data object it has not read before.

BPA describe all processes of an organization and their interdependencies. Each process is understood as a sequence of events, which are interconnected by message and trigger flows, depicting process interaction on an abstract level. BPAs allow to model *multiplicities*, a term subsuming the sending and receiving of variably many messages and triggers to and from multiple process instances of several processes. [3] proposed

correctness criteria for BPAs as well as a transformation into Open nets which allows to model check the criteria.

Definition 1 (Business Process Architecture, based on [2, 3]). A *Business Process Architecture* is a tuple $(E, V, L, I, \chi, \mu, =)$, where E is a set of events, partitioned in start events, E^S, end events E^E, intermediate throwing events E^T, and intermediate catching events E^C and V is a partition of E representing a set of business processes with $v \in V$ being a sequence of events $v = \langle e_1, ..., e_n \rangle$ such that $e_1 \in E^S$ is a start event, $e_n \in E^E$ an end event, and $e_i \in E^C \cup E^T$ for $1 < i < n$ are intermediate events. $L \subseteq (E^T \cup E^E) \times E^C$ is the message flow relation, $I \subseteq (E^T \cup E^E) \times E^S$ is the trigger relation, and $\chi \subseteq \{((e, e_1), (e, e_2)) \mid (e, e_1), (e, e_2) \in L \cup I\}$ is a conflict relation indicating flows that are mutually exclusive. Function $\mu : E \to \mathcal{P}(\mathbb{N}_0)$ denotes the multiplicity set of an event and $= \subseteq (E^T \times E^C) \cup (E^C \times E^T)$ is an equivalence relation between events of the same process demanding that they send respectively receive the same number of messages. $\diamond$

The conflict relation χ relates different flows from one event e which exclude each other. Assume sending event e has three flows $(e, e_1), (e, e_2) \in L, (e, e_3) \in I$ and $(e, e_1)\, \chi\, (e, e_2)$. Then it sends a trigger signal to e_3 and a message to either e_1 or e_2.

4 Data Dependencies

In this section, we show how to extract data dependencies from process models and OLCs, derive the *Process Data Relation Matrix (PDM)*, and construct a *Business Process Data Architecture (data BPA)*.

4.1 Deriving the Process Data Relation Matrix

Annotating the Object Life Cycle. Activities in process models can read or modify data objects. A modifying access changes the state of the data object, corresponding to a state transition in the OLC. We label the state transitions with pairs of process and activity to record their originator. This defines a relation $map \subseteq (S \times \mathcal{P} \times \mathcal{A} \times S)$, where $\mathcal{P}$ and $\mathcal{A}$ are sets of process models and activities. The map is visualized as arc inscriptions, e.g., the modifying access is reflected by the inscription $p_2[b_3]$ on the arc between states "in stock" and "packed" in Fig. 4. The modifications performed by activities are not limited to directly succeeding data states, but can comprise multiple state transitions, as long as there exists a path in the OLC. Activity $p_1[a_7]$ for example transforms data object "product" from state "in stock" to state "shipped" omitting the intermediary state "packed". In this case an additional dashed arc is added to the annotated OLC. Activities which only read data object state are also annotated, depicted by an arc originating from the read state. For example Fig. 4 indicates that state "not in stock" is read by activities a_3 and a_4 of process p_1.

From the annotated OLC, one can deduce direct data dependencies between activities by looking at consecutive state transitions. Two activities from two different processes are dependent ($p[a_i] \dashrightarrow p'[a_j]$) if one activity $p'[a_j]$ reads or modifies a data object that another activity $p[a_i]$ has modified before. For example the transition "unverified" $\xrightarrow{p_4[d_2]}$ "rejected" of data object "order" is followed by the transition "rejected" $\xrightarrow{p_3[c_1]}$ "archived". Because c_1 modifies a state written by d_2 it can only occur afterwards.

Additionally, we define the conflict relation $\otimes$. Two annotated state transitions $\in map$ are in conflict, when they originate in the same source state and belong to different processes. In the OLC of "product" both $p_2[b_3]$ and $p_1[a_7]$ originate in state "in stock" and hence are in conflict.

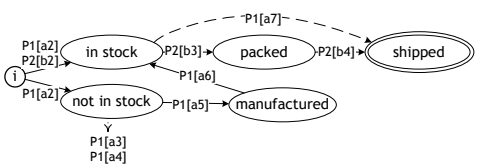


Fig. 4. Annotated product object life cycle

Deriving Process Relations. The two base relations $\blacksquare\!\!\rightarrow$ and $\otimes$ on activity level are lifted to process level, as follows. To determine the relation between two processes, all direct relations between their activities need to be considered along all paths of the OLC of shared data objects because different paths might show contradicting relations.

To determine *exclusive processes* it suffices that one pair of activities be in conflict relation. The OLC in Fig. 5 implies $p_1[a_2] \otimes p_2[b_2]$ (because both originate in the same state) and hence processes p_1 and p_2 are exclusive ($p_1 \# p_2$). The exclusive relation is dominant, in the sense that it overrules other relations in ambiguous situations. Two processes p and p' are called *completely exclusive* ($p[\#]p'$) if a conflict relation involving their activities occurs along each path of the OLC.

Process p' *sequentially* depends on process p with respect to a particular data object D (written as $p \rightarrow_D p'$), if all data accesses of p happen before p' accesses D for the first time on any path of the corresponding OLC. Additionally, there must exist at least one direct data dependency $p[x] \blacksquare\!\!\rightarrow p'[y]$. Regarding the data object "order" process p_3 sequentially depends on p_4 as can be seen from Fig. 5.

We define two variants of the sequential relation. Two processes p and p' are *sequentially overlapping* ($\overset{\uparrow}{\rightarrow}$) if $p \rightarrow p'$ and p additionally reads the last state it has modified in parallel with process p' on handover. A process p' *follows* ($\overset{\uparrow\uparrow\uparrow}{\rightarrow}$) a process p, if p' only reads the modifications performed on a data object by process p, i.e. if there are activities $x_1, \ldots, x_k$ of p and $y_1, \ldots, y_l$ of p' such that $p[x_i] \blacksquare\!\!\rightarrow p'[y_j]$ for some $1 \leq i \leq k, 1 \leq j \leq l$ and each y_j has only reading access.

Two processes p, p' are *interacting* ($\overset{\curlyvee\curlyvee}{\rightarrow}$) if on some path of an OLC direct data dependencies between activities of p and p' occur and vice versa, i.e. $p[x_i] \blacksquare\!\!\rightarrow p'[y_j]$ and $p'[y_k] \blacksquare\!\!\rightarrow p[x_l]$ for some activities x_i, x_l of p and y_j, y_k of p'. Additionally $x_i < x_l$ and $y_j \leq y_k$ have to hold regarding the behavioral profiles of p and p'. Because process p induces the first data dependency on the path it is the initiator of the interaction. If the condition for interacting processes holds on different paths of an OLC the initiator has to be the same process on each path. Otherwise the processes are considered *contradicting* ($\perp$). Generally, interacting means that two processes take turns operating on a data object.

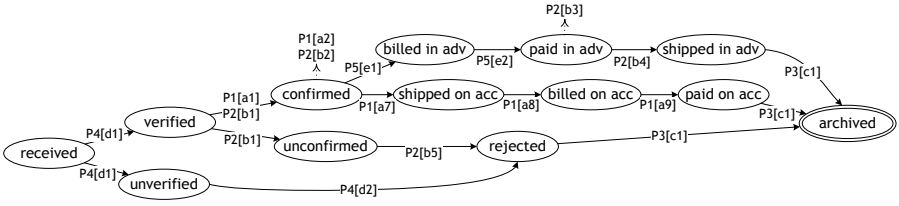


Fig. 5. Annotated order object life cycle

In the OLC of data object "order", we find direct data dependencies $p_2[b_1] \rightarrow\!\!\blacksquare\; p_5[e_1]$ and $p_5[e_2] \rightarrow\!\!\blacksquare\; p_2[b_3]$ (on the top path "verified"–"confirmed"–"billed in advance"–"paid in advance"–"shipped in advance"), and $b_1 < b_3$ as well as $e_1 < e_2$ hold. Therefore we can deduce that $p_2 \overset{\text{xw}}{\rightarrow} p_5$.

Aggregation of Process Relations. So far we considered only single data objects to determine process relations. However, as processes might be related via several data objects we need to consider the cases in which the determined relations differ.

If the sequential relations for two processes agree in regard to all data objects they both accesss, the processes are said to be sequentially dependent ($\rightarrow$). Similarly, this applies for sequentially overlapping and following relations. If, on the other hand, the sequential relation differs for data objects D and D', e.g. $p \rightarrow_D p'$ and $p' \rightarrow_{D'} p$, then processes p, p' are interacting, except when their behavioral profiles contradict. This happens for example, when $p[a] \rightarrow\!\!\blacksquare\; p'[b_1]$ in the OLC of D, $p'[b_2] \rightarrow\!\!\blacksquare\; p[a]$ in the OLC of D' and $b_1 < b_2$ in the behavioral profile of p' all hold.
Two processes are *contradicting*, if they are contradicting for at least one data object. A PDM with contradicting processes when turned into a data BPA will fail to terminate, because of deadlocks during execution. If two processes p and p' are exclusive in regard to one data object, while the other data objects yield different relations, we assume their overall relation to be contradicting. However, processes we identified as contradicting might succeed for some process traces, because activities causing the contradiction were not executed in this trace. In such ambiguous cases, process traces need to be considered which will be part of future work.

Table 1 shows the PDM for the running example, which defines the coarse structure of the data BPA that must conform to the identified relations. As processes p_3, p_4, and p_5 only access the data object "order", their relations are determined by this data object. Processes p_1, p_2 and p_3 sequentially depend on process p_4, process p_3 additionally is sequentially dependent on both p_1 and p_2. Processes p_1 and p_2 use both data objects, but because their relation is exclusive for both, we can unambiguously identify their overall relation as exclusive in the PDM. Processes p_1 and p_5 are also exclusive because of $p_1.[a_7] \otimes p_5.[e_1]$ in the OLC of the data object "order".

Table 1. PDM for scenario

Processes	p1	p2	p3	p4	p5
p1	–	#	→	←	#
p2	#	–	→	←	$\overset{\text{xw}}{\rightarrow}$
p3	←	←	–	←	–
p4	→	→	→	–	–
p5	#	$\overset{\text{xw}}{\leftarrow}$	–	–	–

4.2 Extracting the Business Process Data Architecture

The data dependencies represented in the annotated OLCs constitute an architecture of their own, expressible as *data BPA*. Although the process relations identified in the PDM (see Table 1) determine the overall structure of the data BPA, e.g. process precedence, or exclusivity, they are too coarse-grained to determine the type and order of events and relations in the BPA. Therefore, data BPA extraction requires the annotated OLCs as well as behavioral profiles [10] of the process models.

From Activities to Events. To create a BPA, activities in the process models need to be mapped onto events, $\beta : A \mapsto E$. However, there is no one-to-one mapping because

internal activities are ignored. Only those activities are mapped, that occur in the data dependency relation ($domain(\beta) = \{x \mid x \blacksquare\!\!\rightarrow y \text{ or } y \blacksquare\!\!\rightarrow x\}$). Each pair $x \blacksquare\!\!\rightarrow y$ becomes a sending event $\beta(x)$ and a receiving event $\beta(y)$, which are related in the BPA. Consider the annotated OLC of data object "order" in Fig. 5. Activity $p_2[b_1]$ modifies the state from "verified" to "confirmed", a state which is read by activity $p_5[e_1]$. Processes p_2 and p_5 have further data dependencies $p_2[b_1] \blacksquare\!\!\rightarrow p_5[e_1]$, $p_5[e_2] \blacksquare\!\!\rightarrow p_2[b_3]$ and $p_5[e_2] \blacksquare\!\!\rightarrow p_2[b_4]$, which each turn into a pair of a sending and a receiving event. Activity $p_2[b_4]$ is furthermore in data dependency with $p_3[c_1]$, meaning that $\beta(b_4)$ and $\beta(c_1)$ are in relation. Since in the BPA formalism no event can be both sending and receiving, a second event is introduced for b_4 such that technically $\beta : A \mapsto 2^E$.

The Order of Events. To determine the types of relations and events and to order the events inside BPA processes, we employ behavioral profiles [10]. If according to the behavioral profile an activity a is minimal, i.e. it is the first activity in the process model, $\beta(a)$ is a start event. If a is maximal, i.e. the last activity, $\beta(a)$ is an end event, otherwise it is an intermediary event. The type of relation is now easy to decide, as all flows ending in a start event are trigger flows, and otherwise are message flows. In the running example, d_1 is the last activity in p_4 and hence $\beta(d_1)$ becomes an end event. Activity a_1 is minimal and hence mapped to a start event. The flow $(\beta(d_1), \beta(a_1))$ therefore is a trigger flow $\in I$. Since receiving event $\beta(b_3)$ is neither the first nor the last activity of p_2, it becomes an intermediate catching event $\in E^C$ and $(\beta(e_2), \beta(b_3))$ becomes a message flow $\in L$.

The data BPA process p_2 contains several events, start event $\beta(b_1)$, intermediate throwing event $\beta(b_2)$, intermediate catching events $\beta(b_3)$ and $\beta(b_4)$ and the end event $\beta(b_4)$. To determine the order of the events we consult the behavioral profile and get $b_1 < b_2 < b_3 < b_4$. This order on the activities translates into an order on the BPA events.

Conflicting Processes. The conflict relation $\otimes$ affects the creation of the data BPA in the following way. In Fig. 5, activities $p_1[a_1]$ and $p_2[b_1]$ both transform data object "order" from state "verified" to "confirmed". Hence, those activities are conflicting and the data BPA must prevent the processes, to which $\beta(a_1)$ and $\beta(b_1)$ belong, to be instantiated at the same time. To achieve this, the trigger flows which instantiate exclusive processes need to be in the BPA conflict relation χ. In the running example both $\beta(a_1)$ and $\beta(b_1)$ are the start events of their respective processes and are both triggered by $\beta(d_1)$, because of $d_1 \blacksquare\!\!\rightarrow a_1$ and $d_1 \blacksquare\!\!\rightarrow b_1$. For the data BPA, this means that trigger flows $(\beta(d_1), \beta(a_1)), (\beta(d_1), \beta(b_1)) \in \chi$. Graphically, this is depicted as a XOR gateway in the resulting data BPA in Fig. 6.

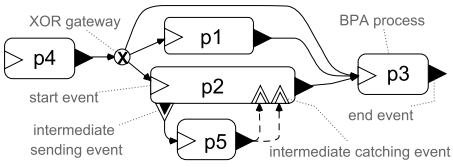


Fig. 6. Resulting data BPA

Multiplicity. As described in Section 3, an activity a might have several data objects as its precondition, e.g. data object "D" in state "s" and data object "E" in state "t". Assume that "D[s]" is written by $p_1[b]$ and "E[t]" is written by $p_2[c]$ and that p_1 and p_2 are non-exclusive, hence implying data dependencies $b \blacksquare\!\!\rightarrow a$ and $c \blacksquare\!\!\rightarrow a$. In the BPA this translates into sending events $\beta(b)$ and $\beta(c)$ both in flow relation with receiving event $\beta(a)$. Since a needs data objects from both b and c, also $\beta(a)$ needs messages

from both $\beta(b)$ and $\beta(c)$. To express this condition, the multiplicity of $\beta(a)$ is set to the total number of activities on which a depends, in this case $\mu(\beta(a)) = \{2\}$.

However, the situation looks different if a has "D" in state "s" and "D" in state "t" as its precondition. Because now either state would suffice to enable a, the receiving event $\beta(a)$ would need a message from either p_1 or p_2, and the multiplicity would be trivial.

5 Conclusion

In this contribution, we presented an approach to extract data interdependencies between a set of process models and vizualize them as business process data architecture (data BPA). Our approach assumes that process models are annotated with data objects and that weakly conforming object life cycles (OLCs) for this objects are given.

The data modifications performed by process activities are annotated in the OLCs to derive direct data dependencies ($\rightarrow\!\!\blacksquare$ and $\otimes$) between activities. Based on that, we extract relations between processes for single and multiple data objects and summarize them in the process data dependency matrix (PDM). Another result is the data BPA, which visualizes found interdependencies and allows formal analysis.

To unambiguously determine some process relations it is required to consider execution traces of process models, which was not part of this contribution. In future work we will also work on lifting the restriction of acyclic OLCs. Our next goal is to combine the data BPAs from this contribution with control flow based BPAs and check their conformance, to accurately represent interdependencies in process model collections.

References

1. Dijkman, R., Vanderfeesten, I., Reijers, H.A.: The Road to a Business Process Architecture: an Overview of Approaches and their Use. Technical Report WP-350, Eindhoven University of Technology (2011)
2. Eid-Sabbagh, R.-H., Dijkman, R., Weske, M.: Business Process Architecture: Use and Correctness. In: Barros, A., Gal, A., Kindler, E. (eds.) BPM 2012. LNCS, vol. 7481, pp. 65–81. Springer, Heidelberg (2012)
3. Eid-Sabbagh, R.-H., Hewelt, M., Weske, M.: Business Process Architectures with Multiplicities: Transformation and Correctness. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 227–234. Springer, Heidelberg (2013)
4. OMG: Business Process Model and Notation (BPMN), Version 2.0 (2011)
5. van der Aalst, W.M.P., Mooij, A.J., Stahl, C., Wolf, K.: Service Interaction: Patterns, Formalization, and Analysis. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009. LNCS, vol. 5569, pp. 42–88. Springer, Heidelberg (2009)
6. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A Framework for Lightweight Interacting Workflow Processes. International Journal of Cooperative Information Systems 10(4), 443–481 (2001)
7. Cohn, D., Hull, R.: Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. IEEE Data Engineering Bulletin 32(3), 3–9 (2009)
8. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking of Interacting Processes with Overlapping Instances. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 345–361. Springer, Heidelberg (2011)
9. Meyer, A., Polyvyanyy, A., Weske, M.: Weak Conformance of Process Models with respect to Data Objects. In: Services and their Composition (ZEUS), pp. 74–80 (2012)
10. Weidlich, M., Mendling, J., Weske, M.: Efficient Consistency Measurement Based on Behavioral Profiles of Process Models. IEEE Trans. Software Eng. 37(3), 410–429 (2011)