Business Process Architectures with Multiplicities: Transformation and Correctness

Rami-Habib Eid-Sabbagh, Marcin Hewelt, Mathias Weske

Technische Berichte Nr. 77

des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 77

Rami-Habib Eid-Sabbagh | Marcin Hewelt | Mathias Weske

Business Process Architectures with Multiplicities

Transformation and Correctness

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.de/ abrufbar.

Universitätsverlag Potsdam 2013
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URL http://pub.ub.uni-potsdam.de/volltexte/2013/6678/
URN urn:nbn:de:kobv:517-opus-66780
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-66780

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-257-5

Business Process Architectures with Multiplicities: Transformation and Correctness

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

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

Abstract. Business processes are instrumental to manage work in organisations. To study the interdependencies between business processes, Business Process Architectures have been introduced. These express trigger and message flow relations between business processes. When we investigate real world Business Process Architectures, we find complex interdependencies, involving multiple process instances. These aspects have not been studied in detail so far, especially concerning correctness properties. In this paper, we propose a modular transformation of BPAs to open nets for the analysis of behavior involving multiple business processes with multiplicities. For this purpose we introduce intermediary nets to portray semantics of multiplicity specifications. We evaluate our approach on a use case from the public sector.

1 Introduction

In today's organisations, business processes play a key role to manage work. Business Process Architectures (BPAs) have been introduced in [1] to represent the interdependencies between related processes. While simple forms of relationships have already been presented, real-world scenarios show that complex relationships between business processes are rather the rule than the exception. This involves the repeated execution of processes as well as multi-communication. By this, we mean communication between multiple instances of several processes instead of one-to-one correspondence between instances. For example in the public administration, a public service often involves the interplay of multiple interacting process instances.

Consider the scenario of applying for a construction permit, illustrated as BPA in Fig. 1. The builder sends a construction permit application to the building authority. This message triggers the "examine application" process. Depending on the type of construction, the application is forwarded to between two and five experts instantiating an appropriate number of "create expert report" processes. On termination each instance returns a message to the "examine application" instance, that waits for the according number of messages, then terminates and returns the decision of the building authority to the applicant. In this example multiple instances of the "expert report" process are triggered and the "examine application" process sends and receives multiple messages.

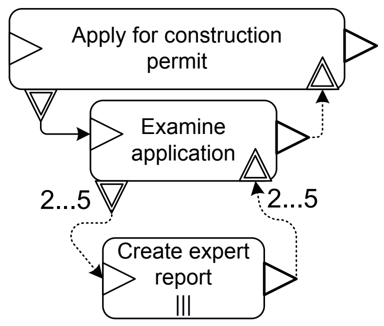


Fig. 1. BPA for a construction permit application

Because the individual processes often are designed independently without the larger context in mind, it is desirable to analyse the behaviour of interacting processes and to assure certain correctness criteria. Additionally the processes often are executed by different administrative units, thus information on how the process interplay can be improved is valuable for the organization.

Business process modeling approaches that allow to express these types of multiplicity do not offer formal analysis. Formal methods based on Petri nets have been successfully applied to model and analyse workflows (Workflow nets [2,3]), services and their composition (service or open nets e.g. [4]) as well as process choreographies (public to private approach [5,6]). However, those elaborated analysis methods do not explicitly deal with multiple instances of processes. Our aim is to model and analyse BPAs with multiple instances and multi-communication. In this contribution we propose a transformation from BPAs with multiplicities into open nets. We introduce intermediary nets to represent and analyse multiple instances and multi-communication in the open nets formalism.

The remainder of this report is structured as follows: Section 2 presents current research on business process interaction. Section 3 introduces the foundations of BPAs and open nets. Section 4 presents the transformation of BPAs and their multiplicity concepts into open nets. We evaluate our approach on a real use case from the public sector in Section 5, and conclude the paper in Section 6.

2 Related Work

Research in the field of BPM evolved from modeling of single processes to modeling the behavior of interacting processes and choreographies, e.g. [7,8,9,10] and analysis of their correctness [11,12,13,14]. To facilitate the modeling of process interaction existing languages were extended, e.g. BPMN with Choreography diagrams, new notations were proposed, e.g. Let's dance [9], and transformations from BPM notations to execution languages were introduced, e.g. BPEL4Chor [10].

However, none of those approaches provide both means to express and formally analyse interactions between multiple processes with several process instances and multi-message communication as they occur for example in the domain of public administration. Most of the proposed solutions address interactions between two processes only and do not consider multiplicities, instead assuming messages to be sent once to only one receiver.

Going a step further, BPMN choreographies also depict the message exchange between two or more processes but do not provide means for correctness analysis. Let's dance was designed for modeling of process interactions. It provides several execution constraints and constructs to model message exchange between multiple participants and instances [9]. For verification only a restricted part of Let's dance diagrams, excluding the "for each"-construct, can be transformed into interaction Petri nets. Hence, analysis is only performed between single processes.

Decker et al. [10] introduce BPEL4Chor, an extension of BPEL, to represent business process choreographies in BPEL. They also provide a transformation of BPMN to BPEL4Chor to join the visual representation of BPMN with the technical expressiveness of BPEL. Both techniques do not provide inherently an analysis of correct business process interaction.

In [7] Proclets are presented to model multi-instance communication between different business processes. They provide a framework to model interacting workflows, their communication channels, as well as multicast messages via ports and associated cardinalities. However, this approach lacks formal analysis techniques.

In contrast to that, Petri net based techniques, e.g. open nets or interaction Petri nets are used to verify correct interaction behavior, compatibility, controllability, and local enforceability [11,12,14,13].

Common patterns of service interaction between two processes were described and examined by [15,13]. Barros et al. [15] also propose three multi-transmission patterns among their basic service interaction patterns. Aalst et al. [13] even look at multi-instance correlation patterns, however only in regard to one to one correspondences. Besides service patterns, they provide means for verification of process interaction for deadlock freedom and controllability but mainly deal with service refinement, replacement and integration.

Similar to their approach our solution builds on the open nets formalism to analyse process interaction behavior. It extends the current research by combining the capabilities of BPAs to express interactions between multiple processes with several process instances as well as multi-communication, with an accordingly adapted open net formalism to analyse such interactions.

3 Foundations

This section prepares the theoretical background for the analysis of BPAs with multiplicities. To this end we expand the definition of *Business Process Architectures* from [1] with a multiplicity equivalence specification, define BPA runs and BPA correctness criteria. Finally, the formalism of open nets is presented.

3.1 Business Process Architectures

Business Process Architectures capture all business processes of an organisation together with their interdependencies, expressed as message and trigger flow relations. Naturally, the question arises whether and in what sense a given BPA is "correct". A preliminary answer might be, that all external requests to the organisation must be handled in such a way that a response is produced. However, only a subset of an organization's processes is involved in handling a specific external request. Such a BPA subset is responsible for a group of related external requests and realizes a service or creates a product of an organisation. For example, the scenario of applying for a construction permit is handled by the interplay of the three processes depicted in Fig. 1. In the following we focus our inquiry on subsets of a BPA.

In contrast to other approaches, BPAs provide means to model *multiplicities*, a term subsuming the sending and receiving of variably many messages to and from multiple process instances of several processes. In the model this is expressed by assigning to each event of the BPA a multiplicity specification, which indicates how many messages or trigger signals the according event can send or needs to receive. Subsection 4.1 will further elaborate on the concept of multiplicities.

Definition 1 (Business Process Architecture, expanded upon [1,16]). A Business Process Architecture is a tuple $(E, V, L, I, \mu, =)$, in which:

- E is a set of events, partitioned into start events, E^S, end events E^E, intermediate throwing events E^T and intermediate catching events E^C.
- V is a partition of E representing a set of business processes.
- $v \in V$ is 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 a flow relation.
- $I \subseteq (E^T \cup E^E) \times E^S$ is a trigger relation.
- $\mu : E \to \mathcal{P}(\mathbb{N}_0)$ denotes the multiplicity set of an event. It contains all valid numbers of messages or trigger signals an event can send or receive. $\mu(e) = \{1\}$ is called *trivial* and omitted in graphical representations.
- $= \subseteq (E^T \times E^C) \cup (E^C \times E^T)$ is an equivalence relation between events of the same process $v \in V$, demanding they send resp. receive the same number of message.

The set $\bullet e = \{e' \in E^E \cup E^T | (e', e) \in I \cup L\}$, called *preset of* e, contains the events with an outgoing relation to $e \in E$. The set $e\bullet = \{e' \in E^S \cup E^C | (e, e') \in I \cup L\}$, called *postset of* e, consists of the events with an incoming relation from $e \in E$ [1].

Business Process Architecture Run. On instance level we define the notion of a *BPA run*, which describes how many instances of each process are instantiated and in which order they interact. The BPA run also determines how many messages or trigger signals an event sends or needs to receive by assigning to each event one element from its multiplicity set. Hence each run consists of a fixed number of process instances, which run in parallel or sequentially. The assignment of multiplicity elements to events must conform to the equivalence specification =. It is either performed at creation of the BPA run or at run time.

A BPA run is started by an initial stimulus that activates all those business processes that are not triggered within the BPA, for instance the desire of a citizen to build a house in Fig. 1. The start events of those processes are considered external [1]. All other start events occur and instantiate their process when they receive the amount of assigned trigger signals from another process in the same BPA run. Events other than start events occur after their preceding event occurred, whereas catching events additionally require to receive the amount of messages assigned to them by this run. Similarly, end and throwing events emit the number of trigger signals or messages assigned to them by that BPA run.

BPA Correctness Criteria. In [16] the authors observe that only few of the proposed and analysed BPA patterns are sound, although they describe the interaction of two processes only. As BPA subsets generally describe the interaction of many processes, the notion of soundness which was introduced for single processes is too restrictive for them. We propose to use the following BPA correctness criteria to decide whether a given BPA is correct.

Every BPA run initially instantiates all those processes whose start events have no incoming triggers ($\bullet b = \emptyset$). A BPA run is called *terminating* if it guarantees for all processes, that the end event of a process will occur eventually once its start event occurred. Hence all processes that are instantiated in a terminating run also terminate. The weaker notion of *lazy termination* allows BPA runs with pending messages or left-behind process instances, if at least one instance of every process, which was instantiated by a run, terminates. We might adapt the example in Fig. 1 in such a way, that three instances of the "expert report" process are instantiated, but the decision on the permit is made already after receiving back two results. This property is comparable to *lazy soundness* in workflow nets [17].

However a BPA run might also fail to terminate, if for a process of a BPA subset one or more occurrences of its start event are part of the run, but its end event is not. Such a BPA run is called a *deadlock*. Similarly, *livelocks* are BPA runs, which are infinite due to business processes triggering each other in a cyclic fashion.

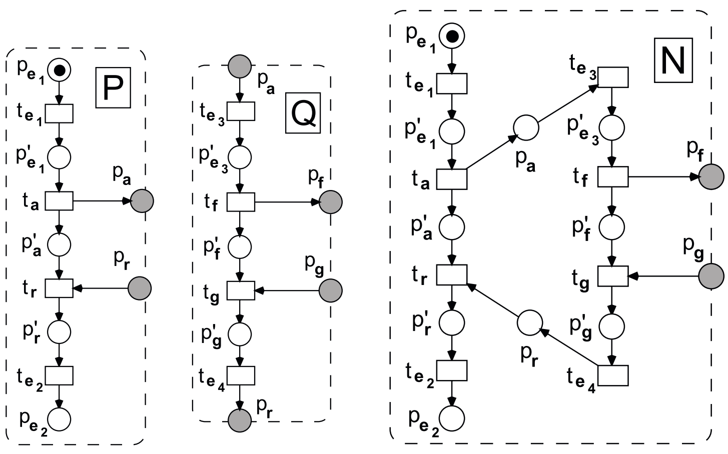
A BPA run need not instantiate all business processes of the BPA subset, however, if it does the run is called *covering*. If a process is instantiated by all terminating runs, it is called *essential*. A process in a BPA is called *dead* if no run instantiates it.

Definition 2 (Correctness Criteria for BPA subsets). A BPA subset is correct if it complies to the following rules:
1. The BPA subset has at least one (lazily) terminating run
2. The BPA subset is free from dead processes
3. The BPA subset contains no livelocks

A BPA is correct if all its subsets are correct.

3.2 Open Nets

The open nets formalism is an extension of classical Petri nets by interface places and final markings. The transformation presented in [16] resulted in so-called *trigger-flow nets*, which are very similar to open nets. To make our research also applicable outside the BPA context, we decided to use open nets, a well recognized formalism, for the transformation. The basic definitions for open nets used here stem from [18].



(a) Open nets P and Q (b) P and Q composed

Fig. 2. Two open nets

For a set X we denote with $MS : X \to \mathbb{N}$ the multiset over X, where each element of X can occur multiple times (i.e. $x \in X$ occurs $MS(x)$ times). We write multisets as a formal sum of their elements e.g. $2 \cdot p_1 + p_2$ for the multiset containing two exemplars of p_1 and one p_2. The empty multiset is denoted as 0.

Definition 3 (Open net). An open net is a tuple $N = (P, T, F, M_0, \Omega)$, in which:

- P is a finite set of places that is partitioned into pairwise disjoint sets of internal places P^N, incoming places P^I, and outgoing places P^O
- T is a finite set of transitions, disjoint with P
- $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the flow relation assigning weights to arcs
- A marking $M : P \to \mathbb{N}$ is a multiset over P i.e. it assigns to each place the number of tokens on this place, M_0 denotes the initial marking of the net, Ω the set of final markings
- $\bullet t$ is called the preset, $t\bullet$ the postset of a transition $t \in T$. Both are multisets over P and denote the amount of token consumed from resp. put on a place.
- $t \in T$ is activated in a marking M, denoted by $m \xrightarrow{t}$ if $\forall p \in P : M(p) \geq \bullet t(p)$ i.e. if there are enough token on p for t to consume
- Firing an activated transition t leads to a follower marking M' defined by $M' = M - \bullet t + t\bullet$

Exemplary open nets P and Q are shown in Fig. 2(a). P has p_a as output and p_r as input place, its initial marking is p_{e_1}, and the only final marking is p_{e_2}. Q has an empty initial marking and as only final marking the empty multiset. p_a and p_g are its input places, p_r and p_f its output places.

Composition of open nets. As the impetus for open nets was to allow modular net composition, interface places were introduced. Composition of open nets works by fusing output places of one net with the matching input places of another net and vice versa. Fused places become internal places in the composed net.

Definition 4 (Composition of open nets). Two open nets N_1 and N_2 are called *composable* if no input place p of one net is also input place of the other net, as well as no output place q of one net is also output place of the other net. If the nets are composable, composition yields open net $N = N_1 \oplus N_2$ with

- $P = P_1 \cup P_2$ and $T = T_1 \cup T_2$
- $P^I = (P_1^I \cup P_2^I) \setminus (P_1^O \cup P_2^O)$
- $P^O = (P_1^O \cup P_2^O) \setminus (P_1^I \cup P_2^I)$
- $P^N = P_1^N \cup P_2^N \cup (P_1^I \cap P_2^O) \cup (P_1^O \cap P_2^I)$
- $M_0 = M_{0_1} + M_{0_2}$
- $\Omega = \{M_1 + M_2 \mid M_1 \in \Omega_1 \wedge M_2 \in \Omega_2\}$

and F being defined as $F(x, y) = F_1(x, y)$ if $(x, y) \in (P_1 \times T_1) \cup (T_1 \times P_2)$ and $F_2(x, y)$ otherwise.

Fig. 2(b) shows the result of composing open nets P and Q from Fig. 2(a). Q's input place p_a was fused with P's output place p_a, output place p_r in Q was fused with input place p_r in P. p_f and p_g remain as interface places in $N := P \oplus Q$. The initial marking of N is the multiset sum of the initial markings of P and Q, resulting in p_{e_1} because M_0^Q was empty. The set of final markings of N results from combining Ω^P with Ω^Q. Here the only final marking of N is p_{e_2}.

4 Transformation of BPA Multiplicities

This section discusses different kinds of multiplicity in BPAs and their according transformation into open nets.

4.1 Multiplicity in BPA

Business Process Architectures exhibit two kinds of multiplicity: a) multiple instances of a business process and b) sending and receiving multiple messages or trigger signals to and from several other process instances. These were described as patterns in [1] but so far not covered by the transformation proposed in [16].

Multiple Process Instances. On model level those processes, that can potentially be instantiated multiple times, are indicated by three vertical bars inside them. The concrete number of times a process is instantiated varies between BPA runs. It depends on the number of trigger signals its start event receives, compared to the multiplicity of its start event, as assigned by this particular run. For start events with trivial multiplicity set, each received trigger signals corresponds to one process instantiation. In Fig. 1 for example, the start event of the "expert report" process is triggered by a throwing event of "examine application" with multiplicity 2 . . . 5. Hence between two and five instances of "expert report" are

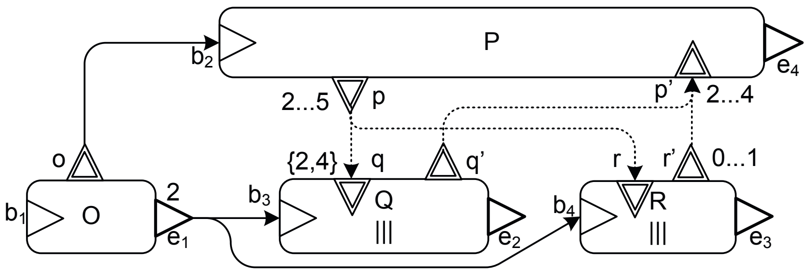


Fig. 3. BPA multiplicity concepts

instantiated. However, if the multiplicity set of the start event is non-trivial, then some runs will assign a multiplicity greater than one and thus in those runs the start event requires several trigger signals, before it can occur and instantiate the process. Start events can be triggered by several other events ($b \in E^S : |\bullet b| > 1$). In such a case each trigger signal from one of the predecessors counts toward the total number of required trigger signals.

Sending and receiving multiple messages. Throwing events can send messages to multiple receiving processes, while catching events can receive messages from multiple sending processes according to the multiplicity assigned to them. In the first case the same amount of messages is delivered to each receiver, while in the second case messages from various senders are collected before being consumed according to the multiplicity specification.

Zero is a valid value in the multiplicity set of a throwing event, meaning that a message (or trigger signal) is not sent at all. At the same time zero is forbidden in the multiplicity for catching events for the following reasons. Optional start events could instantiate an unbounded number of process instances without receiving a trigger signal, while optional receiving events would exhibit the undesired behavior of ignoring incoming messages. If the multiplicity set of an end event contains a zero, this does not mean that the process might not terminate, but rather that it terminates and optionally sends a message or trigger signal.

The BPA in Fig. 3 illustrates those concepts. The triggering of multiple instances of several processes is depicted by the triggering relation of O with Q, and R respectively where end event e_1 has the multiplicity two. In this way two instances of each Q and R are instantiated. The receiving of multiple messages is illustrated by the multiplicity of q ($\mu(q) = \{2, 4\}$) which means that each instance of Q (of which there are two) waits for either two or four messages to proceed. The concept of optional sending is represented by event r' of process R having the multiplicity $(0 \ldots 1)$. Either it sends the message or not. Collecting of messages takes place in catching event p' whose preset contains both q' and r'. As p' has multiplicity $2 \ldots 4$ it expects between 2 and 4 messages in total from (all instances of) Q and R.

Relating event multiplicity specifications. Often the number of messages a process sends is closely related to the number it expects to receive, e.g. in Fig. 1, if three expert reports are requested, also three results are expected back. This relation between two events is captured in the =-relation, to which all BPA runs need to conform. The =-relation reduces the amount of possible BPA runs in the following sense. A single BPA run assigns to each event a number from its multiplicity set, so that each combination of assignments defines a possible run. Runs contradicting the =-relation are considered invalid and can be omitted in the state space of a BPA subset, i.e. the set of all its runs. If the =-relation is not used, all possible runs are valid and the complete state space has to be explored during analysis.

4.2 Transforming Business Process Architectures

For the analysis of BPAs, we employ a transformation into open nets [18], which have been successfully applied to study the composition of services and its correctness. Due to the definition of open net composition we cannot directly express the triggering resp. sending or receiving of a varying amount of instances resp. messages with open nets. Also, the case that one event is in trigger or message flow relation with several other events is not directly covered. To overcome these limitations we adopt the approach of inserting intermediary nets from [16] and extend it with net constructs for multi-communication.

The transformation is conducted in a modular fashion: Each of the BPA's processes is first transformed independently into an open net. In a second step intermediary open nets are created that capture the trigger and message relations and interconnect the process's open nets. In the last step intermediary and processes' nets are composed into one place/transition-net and analysed with the model checking tool LoLA [19].

Transforming Business Processes. We first describe the transformation of a single BPA business process into an open net. It is important to note, that all events are unique and only part of one partition.

Definition 5 (BPA Process Transformation). Given a BPA, let $\langle e_1 e_2 \dots e_n \rangle$ be the sequence of events belonging to the business process $v \in V$ then the process's open net is defined as $N_v = (P_v, T_v, F_v, M_{0_v}, \Omega_v)$, where

- $T_v = \{t_{e_i} | e_i \in v\}$
- $P_v^N = \{p'_{e_i} | e_i \in v \land 1 \leq i < n\}$
- $P_v^O = \{p_{e_i} | e_i \in (E^E \cup E^T) \cap v\} \setminus \{p_{e_n} \,|\, e_n \bullet = \emptyset\}$
- $P_v^I = \{p_{e_i} | e_i \in (E^S \cup E^C) \cap v\} \setminus \{p_{e_1} \,|\, \bullet e_1 = \emptyset\}$
- $P_v = P_v^N \cup P_v^O \cup P_v^I$
- $F = \{(t_{e_i}, p'_{e_i}), (p'_{e_i}, t_{e_{i+1}}) \,|\, t_{e_i} \in T_v \land p'_{e_i} \in P_v^N\} \cup \{(t_{e_i}, p_{e_i}) \,|\, t_{e_i} \in T_v \land p_{e_i} \in P_v^O\} \cup \{(p_{e_i}, t_{e_i}) \,|\, t_{e_i} \in T \land p_{e_i} \in P_v^I\}$
- $M_{0_v} = \emptyset$ if there exists $(t, e_1) \in I$ and p_{e_1} otherwise.
- $\Omega_v = \{\}$ if there exists $(e_n, t) \in I \cup L$ and $\{p_{e_n}\}$ otherwise.

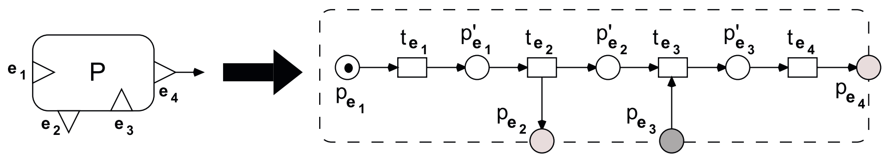


Fig. 4. BPA process to open net transformation

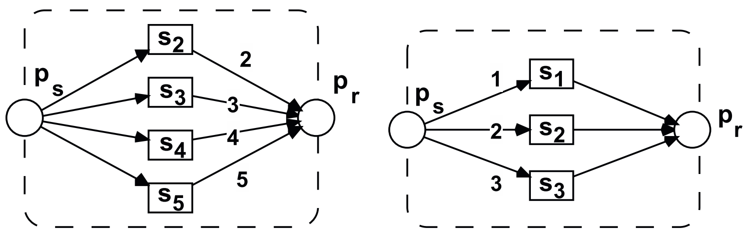
The example in Fig. 4 clarifies this rather technical definition. Each event e yields one transition t_e and, except for the end event, an inner place p'_e. The event's transition is connected to its place, which further connects to the transition of the next event in the sequence. Each event produces another place during transformation, called p_e connected to the events transition. The direction of the arc depends on the event type. If it is a start or catching event (dark gray in Fig. 4) the arc points from place to transition, if it is an end or throwing event (light gray) it points in the opposite direction. The sets P_v^I of input and P_v^O of output places depend partially on the trigger relation I. Only if a start event e_1 is triggered, the place p_{e_1} is an input place, otherwise it is an initially marked internal place as is the case in Fig. 4. Equivalently p_{e_n} is an output place only if e_n has a non-empty postset, otherwise it is internal and part of the final marking. In Fig. 4 there is an arc leaving e_4 indicating a relation, hence p_{e_n} is an output place and not part of a final marking.

Representing multiple instances. There are two ways in open nets to represent the BPA concept of multiple instances. Either each instance is represented by a copy of the process's open net, or each instance is represented by a token (coloured or black) in the process's open net.

Representing each instance by its own open net is problematic when it comes to triggering, as it either requires the creation of nets at runtime or the management of a pool of untriggered nets. Another challenge would be the naming of interface places for each instance net.

Indicating the number of instances as black tokens in one open net eases the composition. In this way the resulting net has less transitions and places than using one net per instance. Hence we decided to use the multiple black token representation for instances. Coloured tokens would allow to distinguish between cases. Research on correlation aspects will be part of future work.

Multicast and multireceive net. Depending on the multiplicity of a throwing event it emits a different number of messages or trigger signals. To capture this in the open nets formalism, we propose to use an intermediary open net called *multicast net.* It is a net schema, because each multiplicity specification entails a different multicast net, consisting of one input place, one output place, and one transition for each element in the multiplicity set of the event. Note, that the same construct is used for triggering multiple instances as well as sending multiple messages.



(a) Multicast net (b) Multireceive net

Fig. 5. Open net constructs to represent multi-triggering and multi-messaging

Definition 6 (Multicast net). Given a BPA the multicast net for a message or trigger flow $(s, r) \in L \cup I$ is defined as $N_{s,r} = (P_{s,r}, T_{s,r}, F_{s,r}, 0, \{0\})$ where

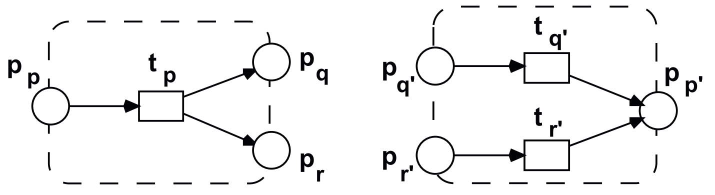
- $P_{s,r} = P^N_{s,r} \cup P^I_{s,r} \cup P^O_{s,r}$ and $P^I_{s,r} = \{p_s\}$, $P^O_{s,r} = \{p_r\}, P^N = \emptyset$
- $T_{s,r} = \{s_i \mid i \in \mu(s)\}$
- $F_{s,r}(p_s, s_i) = 1$, $F_{s,r}(s_i, p_r) = i$ where $i \in \mu(s)$.

Note that the multicast net has the empty multiset as initial and as only final marking.

E.g., let $s \in E^T$ be a throwing event, $r \in E^C$ a catching event and $(s, r) \in L$ be connected by a message flow. Let further $\mu(s) = \{2, 3, 4, 5\}$ be the multiplicity set of s and $\mu(r) = \{1\}$ be r's set. Then the corresponding multicast net is depicted in Fig. 5(a), with the input place p_s, the output place p_r, the four transitions s_2, s_3, s_4, s_5 and the following arcs: p_s is connected to all s_i with arc weight 1 and all s_i are connected to p_r with arc weight i where $i \in \mu(s)$ are the elements of s' multiplicity set. As a result the multicast net produces between two and five tokens on its output place, thus representing the sending of between two and five messages.

We introduce the *multireceive net*, a slightly adapted version of the multicast net, to express that a process instance waits for a certain number of messages before it can continue or that a process is instantiated only after receiving a certain number of trigger signals. The only difference is that arcs from the input place to the transitions now carry the variable weights, while the arc weights between transitions and output place have the value 1. Formally we have $F_{s,r}(p_s, s_i) = i$, $F_{s,r}(s_i, p_r) = 1$ where $i \in \mu(s)$ for the multireceive net, while the rest stays the same. The resulting open net construct is depicted in Fig. 5(b).

Splitter and collector net Processes can not only trigger multiple instances of one process, but also instances of multiple processes. The same is true for sending and receiving messages. In Fig. 3 for example, process P sends messages to both processes Q and R. Formally we have $p\bullet = \{q, r\}$ two events in the postset of throwing event p. The opposite situation, one process receiving messages or trigger signals from several other processes can also be found in Fig. 3 where the catching event p' has the preset $\{q', r'\}$. Note, that it only matters that the preset resp. postset is non-singleton and not in which relation those events are.



(a) Splitter net for event p (b) Collector net for event p'

Fig. 6. Open net constructs for multiple receivers resp. senders

Therefore the following constructions, adapted from [16] apply to both messaging and triggering.

While the splitter net in Fig. 6(a) takes a token from one source and produces one token for each target, the collector net in Fig. 6(b) collects all tokens from several sources in one place. Splitter and collector net are also net schema with multiple incarnations. Principally, the splitter net has one input place, one transition, and one output place for each start or catching event $b \in E^S \cup E^C$ that is in trigger or message flow relation relation with a given end or throwing event $t \in E^E \cup E^T$. The collector net has one place and one transition for each event $e \in \bullet e'$ for a fixed event $e' \in E^S \cup E^C$. All arcs have the weight 1.

Definition 7 (Splitter and collector net). Given a BPA, a throwing or end event $e \in E^T \cup E^E$ and a catching or start event $e' \in E^S \cup E^C$. Then the open net $N_e = (P_e, T_e, F_e, 0, \{0\})$ is called the *splitter net for* e, where

- $P_e = P_e^N \cup P_e^I \cup P_e^O$, $P_e^N = \emptyset$, $P_e^I = \{p_e\}$, $P_e^0 = \{p_b \mid b \in e\bullet\}$
- $T_e = \{t_e\}$
- $F_e(p, t_e) = F_e(t_e, p) = 1 \; \forall p \in P_e$

The initial marking is the empty multiset, which is also the only final marking. The *collector net for* e' is defined similarly except that $P_{e'}^I = \{p_e \mid e \in \bullet e'\}$, $P_{e'}^0 = \{p_{e'}\}$ and $T_{e'} = \{t_e \mid e \in \bullet e'\}$.

4.3 Composition and Analysis

The presented constructs enable us to represent BPAs with multiplicity as open nets and analyse them. Before the analysis, the open nets resulting from the transformation have to be composed according to Def. 4. This composition relies on the names of interface places. Because each event is unique in a BPA and each event is represented by at most one interface place, those are unique too. For each pair of events in trigger or message flow relation at least one intermediary net is created. Those are defined to provide the complementary interface places and make the nets composable. If not for the intermediary nets, the composition would yield unconnected nets leaving all places unfused.

Combinations of intermediary nets In some cases additional care has to be taken to avoid wrong composition, e.g. for events that have both a non-trivial multiplicity set $|\mu(e)| \neq \{1\}$ *and* a non-singleton postset $|e\bullet| > 1$. In such cases multiple intermediary nets are necessary, which per default would have interface places with identical names, thus making the nets non-composable. But since such situations can be derived from the relations, the problem can be circumvented by renaming those places as follows.

Definition 8 (Place renaming for intermediary nets).

a) **Multicast net for** $(e, e') \in L \cup I$

$$P^I = \{p_e\} \qquad P^O = \begin{cases} \{p_{e'}\} & \text{if } e\bullet = \{e'\} \wedge \bullet e' = \{e\} \wedge \mu(e') = \{1\} \\ \{p_{e,e'}\} & \text{if } e\bullet = \{e'\} \wedge \bullet e' = \{e\} \wedge \mu(e') \neq \{1\} \\ \{p''_e\} & \text{if } |e\bullet| > 1 \vee |\bullet e'| > 1 \text{ for } e' \in |e\bullet| \end{cases}$$

b) **Multireceive net for** $(e, e') \in L \cup I$

$$P^I = \begin{cases} \{p_e\} & \text{if } e\bullet = \{e'\} \wedge \bullet e' = \{e\} \wedge \mu(e) = \{1\} \\ \{p_{e,e'}\} & \text{if } e\bullet = \{e'\} \wedge \bullet e' = \{e\} \wedge \mu(e) \neq \{1\} \\ \{p''_{e'}\} & \text{if } |\bullet e'| > 1 \vee |e\bullet| > 1 \text{ for } e \in |\bullet e'| \end{cases} \qquad P^O = \{p_{e'}\}$$

c) **Splitter net for** e (Let $e' \in e\bullet$)

$$\begin{aligned} P^O =& \{p_{e'} \mid |\bullet e'| = 1 \wedge \mu(e') = \{1\}\} \\ &\cup \{p''_{e'} \mid |\bullet e'| = 1 \wedge \mu(e') \neq \{1\}\} \\ &\cup \{p''_{e,e'} \mid |\bullet e'| > 1\} \end{aligned} \qquad P^I = \begin{cases} \{p_e\} & \text{if } \mu(e) = \{1\} \\ \{p''_e\} & \text{otherwise} \end{cases}$$

d) **Collector net for** e' (Let $e \in \bullet e'$)

$$\begin{aligned} P^I =& \{p_e \mid |e\bullet| = 1 \wedge \mu(e) = \{1\}\} \\ &\cup \{p''_e \mid |e\bullet| = 1 \wedge \mu(e) \neq \{1\}\} \\ &\cup \{p''_{e,e'} \mid |e\bullet| > 1\} \end{aligned} \qquad P^O = \begin{cases} \{p_{e'}\} & \text{if } \mu(e') = \{1\} \\ \{p''_{e'}\} & \text{otherwise} \end{cases}$$

Multicast and multireceive nets as well as splitter and collector nets are mirror images of each other, with the roles of pre- and postset, input and output switched. Consider the naming of multicast (multireceive) nets for $(e, e') \in L \cup I$. The name of the only output (input) place depends on whether e (e') has a non-singleton postset (preset) and thus requires a splitter (collector) net [case a_3 above] and whether e' (e) has a trivial [case a_1] or non-trivial [case a_2] multiplicity set.

Splitter (collector) nets have several output (input) places, whose names again depend on the multiplicity of the preset (postset) of the related events. Essentially we distinguish three cases when creating splitter (collector) nets corresponding to the three subsets of the union in the definition of P^O (P^I): 1) no multireceive (multicast) net present, 2) multireceive (multicast) exists and 3) collector (splitter) net present. For the names of input (output) places we need only to differentiate whether a multicast (multireceive) net is present or not. The application of this naming algorithm is illustrated in the use case in Fig. 8.

Algorithm 1 Transformation algorithm for intermediary nets

```
for all $e \in E^E \cup E^T$ do
    if $e\bullet \neq \emptyset$ then
        if $\mu(e) \neq \{1\}$ or
            $(e\bullet = \{e'\}$ and $\mu(e') = \{1\})$ then
            create multicast net
        end if
        if $|e\bullet| > 1$ then
            create splitter net
        end if
    end if
end for
for all $e' \in E^S \cup E^C$ do
    if $\bullet e \neq \emptyset$ then
        if $\mu(e) \neq \{1\}$ then
            create multireceive net
        end if
        if $|\bullet e'| > 1$ then
            create collector net
        end if
    end if
end for
```

The transformation algorithm The algorithm 1 summarizes the creation of intermediary nets in the course of the transformation. First it checks all throwing and end events that have a non-empty postset. A multicast net is created for those events that have a non-trivial multiplicity set. For events whose postset contains several successor events also a splitter net is created. Additionally a multicast net is created for events with trivial multiplicity and a singleton postset, if the only successor event has also a trivial multiplicity set and a singleton preset. This ensures, that for each flow relation between two events at least one intermediary net is created, in the trivial case a multicast net containing only one transition.

In the second step all catching and start events with a non-empty preset are examined. A multireceive net is created, if the multiplicity set of the receiving event is non-trivial. In addition the algorithm creates a collector net for receiving events with several predecessors in its preset. The creation of the individual intermediary nets takes place as described in the corresponding paragraphs of Section 4.2 and in the naming scheme in Definition 8.

Analysis with LoLA To analyse the correctness of a BPA subset LoLAs built-in verification tasks are applied to the composed open net. For this purpose we express the BPA correctness criteria as CTL formulae or state predicates and use model checking to determine if they can be satisfied.

A terminating BPA run is characterized by the final place or places of the net being marked with one or more tokens and any other place in the net being unmarked. If such a state is reachable in the state space, the BPA subset has a terminating run and is thus correct. Lazy termination of a BPA run can be concluded if there is a path in the state space leading to a final marking, such that each process terminates at least once, if it is instantiated at all. In lazy terminating BPA runs unterminated instances and pending messages might stay behind. If a final marking is not reachable, the BPA subset contains only deadlocks and infinite BPA runs (livelocks). Those can automatically be detected by LoLA. Dead processes are found by searching the state space for all those initial places, that are always unmarked. If the transformed open net successfully passes all the verification tasks the BPA subset is correct.

5 Evaluation

To evaluate our approach we consider the usecase of applying for a restaurant business permit combined with a construction permit. An entrepreneur would like to enlarge the facilities chosen for her restaurant, as well as make changes to existing building structures. Fig. 7 shows the BPA subset consisting of seven business processes and their interrelations. E.g. business process p_1, the restaurant business permit application, triggers three business processes p_2, p_3, p_4; the applications for the restaurant permit, the construction permit, as well as the building conversion permit. It is common in the public sector that some of the public administration processes are executed several times by different roles. The construction permit application process p_4, triggers multiple instances of the expert's report process, as depicted by the multiplicity of the throwing event. Another peculiarity is the catching event of the final construction permit evaluation process, which requires between four and six messages from process p_6, which is responsible for checking formal requirements of the expert reports. Only then the final decision on the construction permit can be taken.

On first sight the BPA model seems correct, except for the disparity between the numbers of created instances of process p_5 and the expected messages in process p_7. In [16] the authors presented an approach to verify BPAs, however

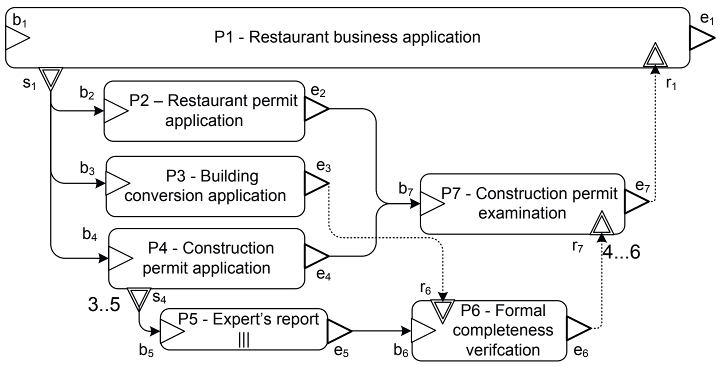

b1
P1 - Restaurant business application
e1
s1
r1
b2
P2 – Restaurant permit application
e2
b3
P3 - Building conversion application
e3
b7
P7 - Construction permit examination
e7
r7
4...6
b4
P4 - Construction permit application
e4
r6
3..5
s4
P5 - Expert's report
|||
b5
e5
b6
P6 - Formal completeness verifcation
e6

Fig. 7. Business Restaurant Permission Application

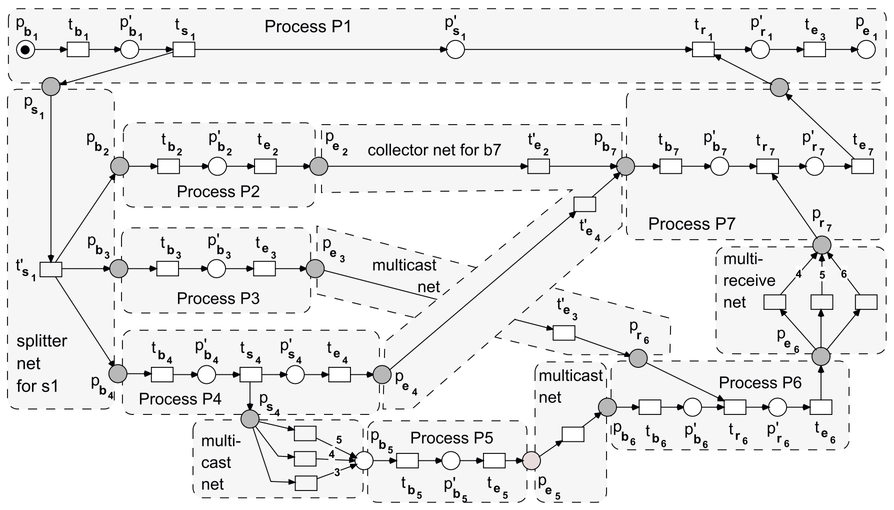


Fig. 8. Extended Open Net BPA Transformation

excluding the complexities that multiplicities create. Fig. 8 shows the transformation of the BPA with multiplicities into an open net which is then used to verify correctness. The analysis with LoLA [19] showed that the resulting open net is structurally sound. However, the net exhibits no firing sequence that would mark the place p_{e_1} which corresponds to the end event of the application process and is the desired final marking. This means that all possible BPA runs are neither terminating nor lazy terminating. Instead the net contains several deadlocks, markings in which no transitions are activated. All those deadlocks have tokens on p'_{b_7} but do not activate t_{r_7} because no tokens are on p_{r_6} Therefore the catching event r_6 in process p_6 fires only once and is dead afterwards. As a consequence process p_7 is blocked after being initiated twice as not enough incoming message are sent by process p_6. All business processes are instantiated, however neither process p_7 nor process p_1 can terminate and only one instance of process p_6 terminates. Consequently the BPA in question is not correct. Without considering multiplicities the transformed BPA would have a lazy terminating BPA run as all processes except from one instance of p_7 could terminate. A strong contrast to the problems found considering multiplicities. Considering and analysing multiplicities in process interaction reveals important aspects for ensuring correct BPAs.

6 Conclusion

Business Process Architectures provide means to model multi-communication between multiple instances of interacting business processes. In this contribution we extended the definition of BPAs with the concept of a BPA run and introduced correctness criteria for BPAs. To analyse BPAs we presented an approach to express the multiplicity-related concepts of BPAs in the formalism of open nets for which a variety of analysis techniques exists. Our main contribution is the introduction of intermediary nets to capture BPA multiplicities inside the frame of open nets. Multicast, multireceive, splitter, and collector nets are composed with the processes' nets by place fusion to form an overall open net. The resulting open net allows analysis with established verification tools.

We evaluated our approach for a non-trivial real world example, the restaurant business application with construction permit. The analysis revealed problems with the process interaction that process modeling notions depicting only single message would have not been able to detect. Future work will focus on the development of a BPA analysis tool to support large scale analysis.

References

1. Eid-Sabbagh, R.H., Dijkman, R.M., Weske, M.: Business Process Architecture: Use and Correctness. In: BPM. Volume 7481 of LNCS., Springer (2012) 65–81
2. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems and Computers **08**(01) (1998) 21–66
3. Verbeek, H.M.W., Basten, T., van der Aalst, W.M.P.: Diagnosing Workflow Processes using Woflan. The Computer Journal **44**(4) (2001) 246–279
4. Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. Annals Of Mathematics, Computing & Teleinformatics **1** (2005) 35–43
5. van der Aalst, W.M.P., Weske, M.: The P2P Approach to Interorganizational Workflows. In: CAiSE. Volume 2068 of LNCS., Springer (2001) 140–156
6. van der Aalst, W.M.P., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: Multiparty Contracts: Agreeing and Implementing Interorganizational Processes. The Computer Journal **53**(1) (2010) 90–106
7. 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**(04) (2001) 443–481
8. Decker, G., Zaha, J., Dumas, M.: Execution Semantics for Service Choreographies. In: Web Services and Formal Methods. Volume 4184 of LNCS. Springer (2006) 163–177
9. Zaha, J., Barros, A., Dumas, M., Hofstede, A.: Let's Dance: A Language for Service Behavior Modeling. In: On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE. Volume 4275 of LNCS. Springer (2006) 145–162
10. Decker, G., Kopp, O., Leymann, F., Pfitzner, K., Weske, M.: Modeling Service Choreographies Using BPMN and BPEL4Chor. In: CAiSE. Volume 5074 of LNCS., Springer (2008) 79–93
11. Martens, A.: Analyzing Web Service Based Business Processes. In: FASE. Volume 3442 of LNCS., Springer (2005) 19–33
12. Decker, G., Weske, M.: Local enforceability in interaction petri nets. In: BPM. Volume 4714 of LNCS., Springer (2007) 305–319
13. van der Aalst, W.M.P., Mooij, A.J., Stahl, C., Wolf, K.: Service Interaction: Patterns, Formalization, and Analysis. In: SFM. Volume 5569 of LNCS., Springer (2009) 42–88

14. Weinberg, D.: Efficient Controllability Analysis of Open Nets. In: Web Services and Formal Methods. Volume 5387 of LNCS. Springer (2009) 224–239
15. Barros, A., Dumas, M., ter Hofstede, A., van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F.: Service Interaction Patterns. In: BPM. Volume 3649 of LNCS., Springer (2005) 302–318
16. Eid-Sabbagh, R.H., Weske, M.: Analyzing Business Process Architectures. In: CAiSE. Volume 7908 of LNCS., Springer (2013)
17. Puhlmann, F., Weske, M.: Investigations on Soundness Regarding Lazy Activities. In: BPM. Volume 4102 of LNCS., Springer (2006) 145–160
18. Massuthe, P., Serebrenik, A., Sidorova, N., Wolf, K.: Can I find a partner? Undecidability of partner existence for open nets. Information Processing Letters **108**(6) (2008) 374–378
19. Schmidt, K.: LoLA: A Low Level Analyser. In: ICATPN 2000, International Conference on Theory and Application of Petri nets. (2000) 465–474

ISBN 978-3-86956-257-5
ISSN 1613-5652