Describing Behavior of Processes with Many-to-Many Interactions

Dirk Fahland(✉)

Eindhoven University of Technology, Eindhoven, The Netherlands
d.fahland@tue.nl

Abstract. Processes are a key application area for formal models of concurrency. The core concepts of Petri nets have been adopted in research and industrial practice to describe and analyze the behavior of processes where each instance is executed in isolation. Unaddressed challenges arise when instances of processes may interact with each other in a one-to-many or many-to-many fashion. So far, behavioral models for describing such behavior either also include an explicit data model of the processes to describe many-to-many interactions, or cannot provide precise operational semantics.

In this paper, we study the problem in detail through a fundamental example and evolve a few existing concepts from net theory towards many-to-many interactions. Specifically, we show that three concepts are required to provide an operational, true concurrency semantics to describe the behavior of processes with many-to-many interactions: unbounded dynamic synchronization of transitions, cardinality constraints limiting the size of the synchronization, and history-based correlation of token identities. The resulting formal model is orthogonal to all existing data modeling techniques, and thus allows to study the behavior of such processes in isolation, and to combine the model with existing and future data modeling techniques.

Keywords: Multi-instance processes · Many-to-many interactions · Modeling · True-concurrency semantics · Petri nets

1 Introduction

Processes are a key application area for formal models of concurrency, specifically Petri nets, as their precise semantics allows both describing and reasoning about process behavior [1]. The basic semantic concepts of Petri nets, locality of transitions which synchronize by "passing" tokens, are at the core of industrial process modeling languages [21] designed to describe the execution of a process in a *process instance* which is isolated from all other instances. At the same time, processes behavior in practice is often not truly isolated in single process instances, but instances are subject to interaction with other instances, data objects, or other processes. Modeling and analyzing such processes has been the

S. Donatelli and S. Haar (Eds.): PETRI NETS 2019, LNCS 11522, pp. 3–24, 2019.
https://doi.org/10.1007/978-3-030-21571-2_1

focus of numerous works such as proclets [3], artifact-centric modeling [6], UML-based models [5], BPMN extensions [16], DB-nets [19], object-centric declarative modeling [2], and relational process structures [27].

The existing body of work can be considered in two major groups. Artifact-centric modeling that can address one-to-many and many-to-many relations first defines a relational data model; process behavior is then defined by a control-flow model for each entity in the data model, and logical constraints and conditions that synchronize the steps in one entity based on data values in the data model of another entity [5,6]. DB-nets adopt this principle to Petri net theory through Coloured Petri nets [19]. Numerous decidability results and verification techniques are available for these types of models, such as [5,7,17]. However, the behavior described by these models cannot be derived easily by a modeler through visual analysis as the interaction of one entity with other entities depend on complex data conditions not shown in the visual model, inhibiting their application in practice [22]. While object-centric declarative models [2,12] make the dependency between behavior and data visually explicit, declarative constraints themselves are challenging to interpret. Proclets do explicitly describe interactions between instances of multiple processes [3] but do not provide sufficient semantic concepts to describe many-to-many interactions [10]. Relational process structures [26,27] turn the observations of [10] into a model with implemented operational semantics for describing many-to-many interactions through so-called coordination processes. However, the language requires numerous syntactic concepts, and no formal semantics is available, prohibiting analysis, which also applies to data-aware BPMN extensions [16].

In this paper, we investigate semantic concepts that are required to provide formal semantics for a modeling language that is able to describe many-to-many interactions. The language shall bear a minimal number of syntactic and semantic concepts building on established concepts from Petri net theory. Our hope is that such a minimal, yet maximally net-affine language allows to project or build richer modeling languages on top of our proposed language, while allowing to apply or evolve existing Petri net analysis techniques for analyzing behavior with many-to-many interactions.

In Sect. 2, we study a basic example of many-to-many interactions through the formal model of Proclets and analyze the core challenges that arise in describing the behavior of such processes. In Sect. 3, we show that these challenges can be overcome by a paradigm shift in describing many-to-many relations. Just as many-to-many relations in a data model have to be reified into its own entity, we show that many-to-many interactions require reifying the message exchange into its own entity. Based on this insight, we then propose in Sect. 4 *synchronous proclets* as a formal model that extends [3] with *dynamic unbounded synchronization* of transitions. We provide a formal true concurrency semantics for our model. We then show in Sect. 5 how the semantics of relations between entities can be realized on the level transition occurrences as *cardinality* and *correlation* constraints over pairs of instance identifiers to fully describe many-to-many

interactions in an operational semantics. We discuss some implications of our work in Sect. 6.

2 Multi-dimensional Dynamics - A Simple Example

The running example for this paper describes a very simple logistics process. After an order has been created, it gets fulfilled by sending packages to the customer. Typically, not all products are available in the same warehouse, resulting the order to be split into *multiple* packages. Packages are transported to the customer through a delivery process, where multiple packages are loaded for one delivery tour; these packages may originate from *multiple* orders, resulting in a many-to-many relationship between orders and deliveries. Packages are then delivered one by one, being either successfully delivered or the package could not be delivered, leading either to a retry in a new delivery or in considering the package as undeliverable. The customer is billed only after all deliveries of all packages in the order concluded.

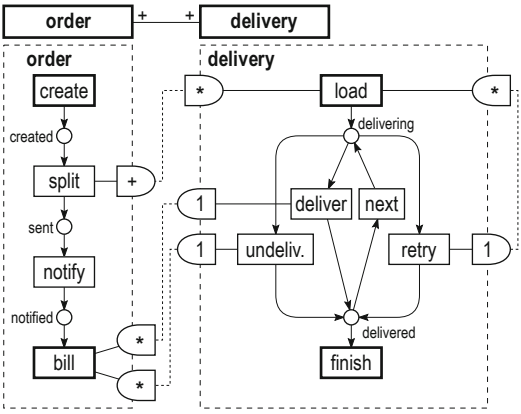

Fig. 1. Proclet model describing asynchronous message exchange between multiple instances

The proclet model [3] in Fig. 1 describes this process. The behavior of order and delivery instances are described in their respective *proclet*, initial and final transitions describe the creation and termination of instances. Instances of order and delivery *interact* by exchanging messages via channels (dashed lines); the cardinality inscriptions at the ports indicate how many messages are sent or received in the occurrence of the transition in one instance. Figure 2 shows a *partially-ordered run* that satisfies the model of Fig. 1: order17 gets split into two packages 1 and 3, while order18 requires just a single package 2. Packages 1 and 2 are loaded into delivery23 where package 2 requires a retry with delivery24 where it is joined by package 3.

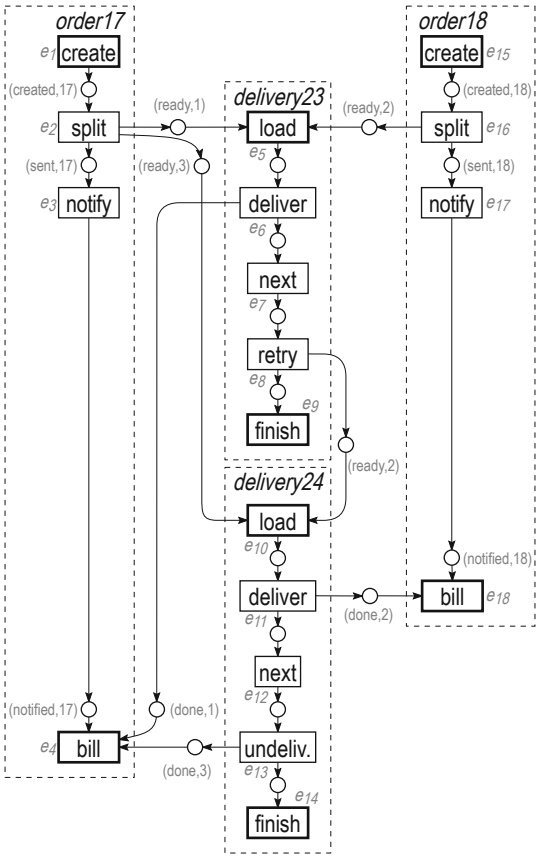

Fig. 2. A partially-ordered run of the proclet model in Fig. 1

The proclet semantics [3] however is not defined on instance identifiers and hence also allows undesired behaviors in many-to-many interactions, such as the run in Fig. 3: First, package 1 gets duplicated and then both delivered and gets a retry with delivery24. Second, package 3 originates in order17 but gets billed for order18. Third, package 2 is created and loaded but then disappears in the run.

3 Reifying Behavior of Relations into Conversations

The core problem of the proclet model in Fig. 1 is that order and delivery are in a many-to-many relation that is not explicitly described. We know from data modeling that implementing a many-to-many relationship in a relational data model requires to reify the relationship into its own entity, which then also results in the well-known Second Normal Form (2NF). By the same reasoning, we reify

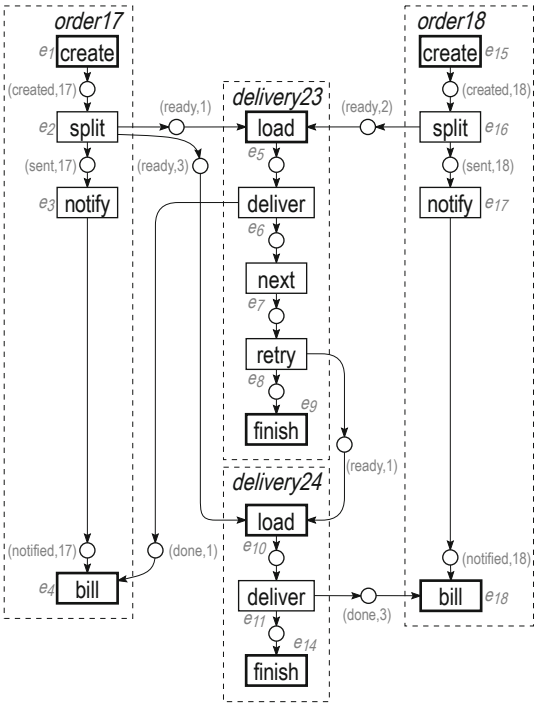

Fig. 3. Another partially-ordered run of the proclet model in Fig. 1 describing undesired behavior

the structural many-to-many relation in Fig. 1 into a package entity which has one-to-many relations to order and delivery.

The first central idea of this paper is that we also reify the *behavioral* relation between order and delivery, i.e., the channels, into its own sub-model. The resulting composition, shown in Fig. 4, contains a proclet for the package entity. Where the model in Fig. 1 described interaction through asynchronous message exchange, the model in Fig. 4 uses *synchronization of transitions* along the indicated channels. Hence, we call this model a *synchronous* proclet model. Crucially, *multiple* instances of a package may synchronize with an order instance in a single transition occurrence. Figure 5 shows a distributed run of the synchronous proclet system of Fig. 4, where the run of each instance is shown separately and the dashed lines between events indicate synchronization. For example, events e_2, e_2', and e_2'' in order17, package1, and package3 occur together at once in a single *synchronized event* which also causes the creation of the package1, and package3 instances. The run in Fig. 5 is identical to the run in Fig. 2 except for the condition loaded in the different package instances.

In line with the idea of a relational model in 2NF, the synchronization channels in the model in Fig. 4 only contain one-to-many, and one-to-one cardinalities. We consider a model such as the one in Fig. 4 to be in *behavioral second normal*

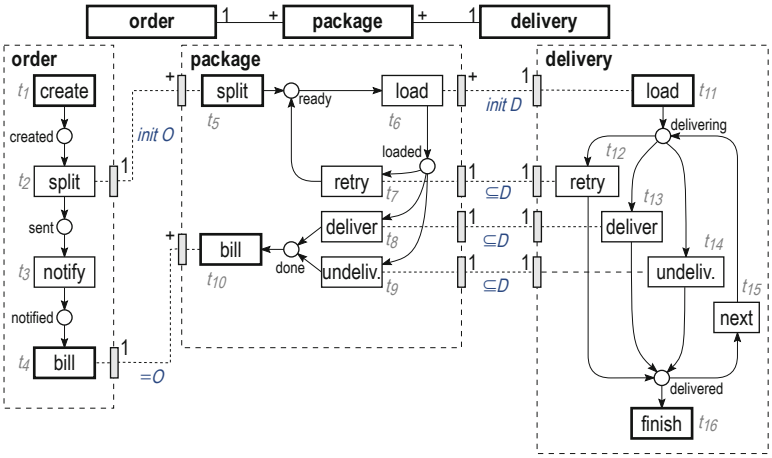

Fig. 4. Synchronous Proclet model

form: During any interaction between two entities, one entity is always uniquely identified, by having only one instance participate in the interaction. Specifically, a package always relates and interacts with one delivery and one order. However, that relation is dynamic as package2 first relates to delivery23 and later to delivery24. To ensure that "the right" instances remain synchronized, we propose using *correlation* constraints: the annotation init O and =O shall ensure that only package instances created by an order also synchronize in the bill step.

4 Dynamic Unbounded Synchronization

In the following, we develop the required formal concepts for the model proposed in Sect. 3. We first define dynamic synchronization of transitions in a true-concurrency semantics. In Sect. 5 we constrain synchronization through cardinality and correlation constraints.

4.1 Notation on Nets

A *net* $N = (P, T, F)$ consists of a set P of *places*, a set T of *transitions*, $P \cap T = \emptyset$, arcs $F \subseteq (P \times T) \cup (T \times P)$. We call $X_N = P \cup T$ the *nodes* of N. We write ${}^\bullet t$ and $t^\bullet$ for the set of pre-places and post-places of $t \in T_N$ along the arcs F, respectively; pre- and post-transitions of a place are defined correspondingly. We write $N_1 \cap N_2$, $N_1 \cup N_2$, and $N_1 \subseteq N_2$ for intersection, union, and subset of nets, respectively, which is defined element-wise on the sets of nodes and arcs of N_1 and N_2, and we write $\emptyset$ for the empty net.

A *labeled* net $N = (P, T, F, \ell)$ additionally defines a *labeling* $\ell : P \cup T \to \Sigma$ assigning each node $x \in X_N$ a label $\ell(x) \in \Sigma$; w.l.o.g, we assume for any two

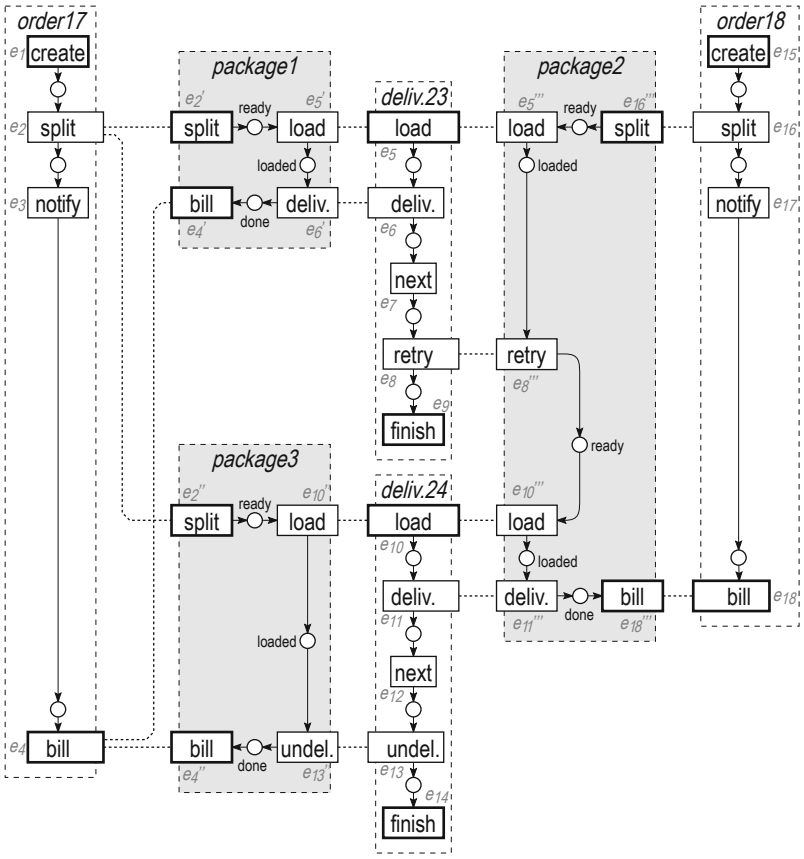

Fig. 5. Synchronization of multiple partially ordered runs corresponding to the synchronous proclet model of Fig. 4

labeled nets N_1, N_2 that $\ell_1(x) = \ell_2(x)$ for any node $x \in X_1 \cap X_2$. We use labels to synchronize occurrences of different transitions with the same label.

For a node $x \in X_N$ and a fresh node $x^* \notin X_N$ we write $N[x/x^*]$ for the net obtained by replacing in N simultaneously all occurrences of x by x^*.

An *occurrence net* $\pi = (B, E, G)$ is a net (B, E, G) where each place $b \in B$ is called a condition, each $e \in E$ is called an event, the transitive closure G^+ is acyclic, the set $past(x) = \{y \mid (y, x) \in G^+\}$ is finite for each $x \in B \cup E$, and each $b \in B$ has at most one pre-event and at most one post-event, i.e., $|{}^\bullet b| \leq 1$ and $|b^\bullet| \leq 1$. We will consider *labeled* occurrence nets $\pi = (B, E, G, \lambda)$, where each condition (event) is labeled with a *set* of labels of the from (x, id) where x refers to a place (transition) of another net, and id is an instance identifier. The behavior of any net N (with an initial marking m_0) can be described as a set of occurrence nets $R(N, m_0)$ called the runs of N.

4.2 Entities, Instances, and Synchronous Proclet System

A classical process model N_E describes the processing of a single entity E, e.g., an order, a delivery. This can be formalized as a single labeled net $N_E = (P_E, T_E, F_E, \ell_E)$. The behavior of *one instance* of E, e.g., a concrete order, then follows from consuming and producing "black" tokens in N_E which defines a firing sequence or a run of N_E [1].

Our aim is to describe the behavior of *multiple entities* and *multiples instances* of these entities *together in one* run as discussed in Sect. 2. We adopt Petri nets with token identities [11,24,25] to distinguish different instances of an entity E. Let $\mathcal{I}$ be an infinite set of instance *identifiers*. Each $id \in \mathcal{I}$ is a unique identifier. By distributing $\mathcal{I}$ (as tokens) over the places of N_E, we describe the state of instance id. The state of N_E (for several instances) is then a distribution of *multiple* tokens from $\mathcal{I}$ over the places of N_E.

To describe the interplay of multiple entities, we adopt concepts of *proclets* [3,4] and *open nets* [14]. A system describes the behavior of each entity in its own net; the nets of all entities are *composed* along channels. Where the earlier works use asynchronous channels for composition, we use *synchronous channels* which are connecting pairs of transitions, as motivated in Sect. 3. This gives rise to the notion of a *synchronous proclet system*:

Definition 1 (Synchronous Proclet System). *A* synchronous proclet system $S = (\{N_1, \ldots, N_n\}, m_\nu, C)$ *defines*

1. *a set of labeled nets* $N_i = (P_i, T_i, F_i, \ell_i), i = 1, \ldots, n$, *each called a* proclet *of* S,
2. *an* initial marking $m_\nu : \bigcup_{i=1}^{n} P_i \to \mathbb{N}^{\mathcal{I}}$ *assigning each place* p *a multiset* $m(p)$ *of identifier tokens such that proclets have disjoint sets of identifiers,* $\forall 1 \leq i < j \leq n, \forall p_i \in P_i, p_j \in P_j : m_\nu(p_i) \cap m_\nu(p_j) = \emptyset$, *and*
3. *a set* C *of* channels *where each channel* $(t_i, t_j) \in C$ *is a pair of identically-labeled transitions from two different proclets:* $\ell(t_i) = \ell(t_j)$, $t_i \in T_i, t_j \in T_j, 1 \leq i \neq j \leq n$.

Figure 4 shows a synchronous proclet system for entities order, package, and delivery with several channels, for example, channel (t_2, t_5) connects two split-labeled transitions in order and package. Each proclet in Fig. 4 has a transition without pre-places and the initial marking is *empty*. This allows creating an unbounded number of new instances of any of the three entities in a run.

4.3 Intuitive Semantics for Synchronous Proclet Systems

A single proclet N_i describes the behavior of a single entity E_i. We assume each instance of E_i is identified by an identifier id. A distribution of id tokens over the places in N_i describes the current state of this instance. The instance id advances by an *occurrence* of an enabled transition of N_i in that instance id: Any transition $t \in N_E$ is *enabled* in instance id when each pre-place of t contains an id token; firing t in instance id then consumes and produces id tokens as usual. A *new*

instance of E_i can be created by generating a new identifier id_ν as proposed for ν-nets [25]. We limit the creation of new identifiers to transitions without pre-set. Such an "initial" transition t_{init} is always enabled (as it has no pre-places); t_{init} may occur in instance $id_\nu \in \mathcal{I}$ only if id_ν is a fresh identifier never seen before; its occurrence then produces one id_ν token on each post-place of t_{init}. For example, in the run in Fig. 5, we see three occurrence of t_5 (split) in proclet package, each occurrence creates a different token package1, package2, package3 $\in \mathcal{I}$ on place ready describing the creation of three different package instances.

In the entire proclet system, a local transition that is not connected via any channel, such as t_3 (notify) in order in Fig. 4, always occurs on its own. However, for transitions that are connected to each other via a channel, such as t_2 and t_5 (split), their occurrences *may synchronize*.

The modality "may synchronize" is important in the context of true concurrency semantics. Considering the partially-ordered run in Fig. 5, we can see two occurrence of t_2 (split) in instances order17 and order18, and three occurrences of t_5 in package1–package3. Bearing in mind that all instances are concurrent to each other, we may not enforce that one occurrence of t_2, say, in order17 *must* synchronize with *all* occurrences of t_5 in package1–package3. If we did, we would silently introduce a notion of global state and a global coordination mechanism which knows all order instances in that state, i.e., at a particular point in time. Rather, by synchronizing on *non-deterministically chosen subsets of possible occurrences*, we can express local knowledge. The occurrence of t_2 in order17 synchronizes with occurrences of t_5 in package1 and package2 in Fig. 5 because they happen to be "close to each other"—because package1 and package2 are created for order17. In other words, this non-determinism on synchronizing transition occurrences allows to abstract from a rather complex data-driven mechanism describing *why* occurrences synchronize while preserving *that* occurrences synchronize. While this very broad modality also leads to undesired behaviors intermittently, the notion of channel will allow us to rule out those undesired behaviors through a *local mechanism* only.

4.4 Partial Order Semantics for Synchronous Proclet Systems

In the following, we capture these principles in a true concurrency semantics of runs by an inductive definition over labeled occurrence nets. Specifically, we adopt the ideas proposed for plain nets [8,23] to our setting of multiple, synchronizing instances:

A *run* describes a partial order of transition occurrences which we represent as a special *labeled*, acyclic net π as shown in Fig. 2. A place b in π is called *condition*; its label $\lambda(b) = (p, id)$ describing the presence of a token id in a place p. A transition e in π is called an *event*; its label $\lambda(e) = t$ describes the occurrence of t. For example, the event e_2 in the run in Fig. 2 describes the occurrence of a transition which consumes token 17 from place created produces token 17 on sent and tokens 1 and 3 on place ready; e_2 is un-ordered, or *concurrent*, to e_{16}.

We construct such runs inductively. The initial state of a proclet system is a set of *initial conditions* representing the initial marking m_ν. The run in

Fig. 2 has no initial conditions. To extend a run, we term the *occurrence* of a transition t in an instance id. An occurrence o of t is again small net with a single event e labeled with (t, id); e has in its pre-set the conditions required to enabled t in instance id: for each pre-place p of t, there is a pre-condition of e with label (p, id); the post-set of e is defined correspondingly. Each occurrence of t describes the enabling condition for t and the effect of t in that particular instance id. Figure 6 (top) shows two occurrences of t_2 (split) in instance order17 and in instance order18, and three occurrences of t_5 (split) in instances package1-package3.

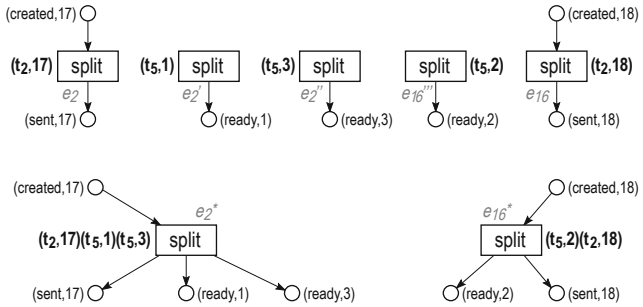

Fig. 6. Occurrences of transitions t_2 and t_5 in different instances (top), and synchronized occurrences of t_2 and t_5 in different instances (bottom)

Intuitively, a run is obtained by repeatedly appending occurrences (of enabled transitions) to the run. The *maximal* conditions of a run π describe which places hold which tokens, that is, the current marking of the system. For instance, the prefix π_1 shown in Fig. 7 has reached the marking where only place created holds the tokens 17 and 18. An occurrence o of a transition t is then *enabled* in a run π if the pre-conditions of o also occur in the maximal condition of π. For instance, the occurrence of t_2 (split) in order17 shown in Fig. 6 (top) is enabled in π_1 in Fig. 7. Also the occurrences of t_5 in package1-package3 and of t_2 in order18 are enabled π_1.

In a classical net, we could now append the occurrence of t_2 in order17 to π_1. In a synchronous proclet system, occurrences of enabled transitions that are connected via a channel may synchronize. For example, we may synchronize the occurrences of t_2 in order17 with the occurrences of t_5 in package1 and package3 which we express as a *synchronized occurrence*. The synchronized occurrence unifies the events in all individual occurrences into a single event e^* and otherwise preserves all pre- and post-conditions; we label e^* with the multiset of transitions occurring together as shown in Fig. 6 (bottom). The synchronized occurrence is then appended to the run. For example, we obtain run π_2 in Fig. 7 by appending the synchronized occurrence of t_2, t_5, t_5 in order17, package1, package3 to run π_1, where synchronization is shown in Fig. 7 through a dashed line. In run π_2

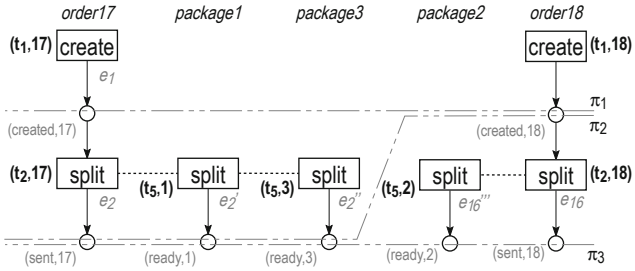

Fig. 7. Prefixes of the run of Fig. 5

the synchronized occurrence of t_2, t_5 in order17 and package2 (shown in Fig. 6 (bottom)) is enabled; appending it yields run π_3.

4.5 Formal Semantics for Dynamic Synchronization

We formalize the semantics of proclet systems laid out in Sect. 4.4 by an inductive definition. For the remainder of this section, let $S = (\{N_1, \ldots, N_n\}, m_\nu, C)$ be a proclet system, and let $T_S = \bigcup_{i=1}^{n} T_i$ be all transitions in S.

An *occurrence* of a transition t in an instance id is an occurrence net o with a single event e describing the occurrence of t; the labeling of o describes an injective homomorphism between the pre-places of t and the pre-conditions of e, and between the post-conditions and post-places, respectively. Technically, the event's label is a singleton set $\{(t, id)\}$ and a condition's label is a singleton set $\{(p, id)\}$; using sets of labels will allow us later to establish an injective homomorphism from synchronized occurrences of transitions to the transitions in the system definition.

Definition 2 (Occurrence of a transition). *Let $t \in T_S$ be a transition. Let $id \in \mathcal{I}$ be an identifier.*

An occurrence of t in id *is a labeled net $o = (B_{pre} \uplus B_{post}, \{e_t\}, G, \lambda)$ with*

1. *for each $p \in {}^\bullet t$ exists a* condition $b_p \in B_{pre} : \lambda(b_p) = \{(p, id)\}(b_p, e_t) \in G$
2. *for each $p \in t^\bullet$ exists a* condition $b_p \in B_{post} : \lambda(b_p) = \{(p, id)\}, (e_t, b_p) \in G$
3. $\lambda(e_t) = \{(t, id)\}$.

The labeling ℓ_i canonically lifts to o: for each node $x \in B_{pre} \cup B_{post} \cup \{e_t\}$ referring to $\lambda(x) = \{(y, id)\}$ set $\ell(x) := \ell_i(y)$.

We call $pre_o := B_{pre}$ the precondition *of o, we call $con_o := B_{post} \cup \{e_t\}$ the contribution of o, and write $e_o := e_t$ for the event of o. Let $O(t, id)$ be the set of all occurrences of t in id.*

Figure 6 shows among others occurrences of t_2 in order1 and of t_5 in package2. Each transition (also in the same instance) may occur multiple times in a run, see for instance t_6 (load) in package2 in Fig. 5. To make our task of composing runs from transition occurrences easier, and not having to reason about the identities

of events or conditions, we therefore consider the set $O(t, id)$ of all occurrences of t in id. Technically, it contains an infinite supply of isomorphic copies of *the* occurrence of t in id. When composing runs and when synchronizing occurrences, we will simply take a suitable copy.

We now turn to synchronizing transition occurrences. Although we will later only synchronize transition occurrences along channels, we propose here a simpler and more general definition. Where an occurrence of a transition is parameterized by the transition t and the instance id, a *synchronized occurrence* is parameterized by a label a and a finite *set* $I \subseteq \mathcal{I}$ of instances. For each instance $id \in I$, one transition with label a participates in the synchronized occurrences. We first define, given a and I, the sets of occurrences of transitions that may be synchronized. In the spirit of high-level nets which also parameterize transition occurrences through variables, we call this set of occurrences an *occurrence binding* to a and I. The synchronized occurrence is then a canonical composition of all occurrences in the binding.

Definition 3 (Occurrence binding). *Let $a \in \Sigma$ be a label. Let $I \subseteq \mathcal{I}$ be a nonempty finite set of identifiers.*

A set O_I of occurrences is an occurrence binding *for a and I (in system S) iff for each $id \in I$ exists exactly one occurrence $o_{id} \in O(t, id)$ of some $t \in T_S$ with $o_{id} \in O_I$. We write $O(a, I)$ for the set of all occurrence bindings for a and I.*

In Fig. 6, the occurrences $o_1 \in O(t_5, 1), o_3 \in O(t_5, 3), o_{17} \in O(t_2, 17)$ form an occurrence binding $\{o_1, o_3, o_{17}\}$ for split and $I = \{1, 3, 17\}$. In this example, there is no other occurrence binding for this label and set of instances (up to isomorphism of the occurrences themselves). Without loss of generality, we may assume that any two occurrences $o, o' \in O_I$ in an occurrence binding $O_I \in O(a, I)$ are pair-wise disjoint, i.e., $o \cap o' = \emptyset$.

We obtain a *synchronized occurrence* of instances I at a by composing the occurrences in the binding for a and I along the events labeled a. For example, the synchronized occurrence of $\{1, 3, 17\}$ at split is shown in Fig. 6 (bottom left). To aid the composition, we use (1) simultaneous replacement $o[e/e^*]$ of an event by a new event $e*$, as defined in Sect. 4.1, and (2) we lift the union of nets in Sect. 4.1 to union of occurrences $o_1 \cup o_2$ by defining $\lambda_1 \cup \lambda_2(x) := \lambda_1(x) \cup \lambda_2(x)$ for each $x \in X_1 \cup X_2$. The formal definition of a synchronized occurrence reads as follows.

Definition 4 (Synchronized occurrence of a label). *Let $a \in \Sigma$, let $I \subseteq \mathcal{I}$ be a nonempty finite set of identifiers, and let $O_I \in O(a, I)$ be an occurrence binding for a and I in system S.*

The synchronized occurrence *of instances I at label a is the net*

$$\tilde{o} = \bigcup_{o_{id} \in O_I} (o_{id}[e_o/e^*])$$

obtained by replacing the event e_o in each occurrence o_{id} be a fresh event $e^* \notin X_i$, for all $i = 1, \ldots, n$ which unifies *the occurrences along the event, and compos-*

ing all occurrences by union. We write $\tilde{O}(a, I)$ *for the set of all synchronized occurrences of* I *at* a.

Figure 6 (bottom left) shows the synchronized occurrence of order17, package1, and package3 at split. Note that the event of this occurrence is labeled with the set $\{(t_2, 17), (t_5, 1), (t_5, 3)\}$ resulting from the synchronizing composition of events labeled with $(t_2, 17)$, $(t_5, 1)$, and $(t_5, 3)$. Note that this synchronized occurrence describes the *instantiation* of two new packages package1, package2 by order17.

Labeling event e^* with a set $\lambda(e^*)$ does not allow us to describe auto-concurrency of a transition, i.e., a synchronization of two occurrences of (t, id) when ${}^\bullet t$ is marked with two or more id tokens. This case is excluded by our definition of occurrence binding (Definition 3) allowing for each $id \in I$ exactly one occurrence; this limitation could be overcome by using multisets.

The labeling $\ell(.)$ which we lifted to occurrences of a transition (Definition 2) also lifts to synchronized occurrences, as the synchronization in Definition 4 only merges events $e_1, \ldots, e_n$ carrying the same $\ell(e_1) = \ldots = \ell(e_n) = a$. Also the precondition, contribution, and the event of an occurrence lift to synchronized occurrences written $pre_{\tilde{o}}$, $con_{\tilde{o}}$, and $e_{\tilde{o}}$, respectively.

Each synchronized occurrence structurally preserves its constituting occurrences through the labeling $\lambda(.)$. Let $o = (B, E, G, \lambda)$ be a labeled occurrence net. Let $I \subseteq \mathcal{I}$. We write $o|_I$ for the restriction of o to those nodes labeled with an $id \in I$: $o|_I = (B \cap Z, E \cap Z, G|_{Z \times Z}, \lambda|_I)$ with $Z = \{x \in X_o | (y, id) \in \lambda(x), id \in I\}$ and $\lambda|_I(x) := \{(y, id) \in \lambda(x) | id \in I\}$. Restricting any synchronized occurrence to a single identifier results in the occurrence of the corresponding transition in that instance.

Lemma 1. *Let* $a \in \Sigma$, *let* $I \subseteq \mathcal{I}$ *be a nonempty finite set of identifiers, and let* $O_I \in O(a, I)$ *be an occurrence binding for* a *and* I *in system* S. *For each* $\tilde{o} \in \tilde{O}(a, I)$ *holds: for each* $id \in I$, $o := \tilde{o}|_{\{id\}} \in O(t, id)$ *where* $\lambda(e_o) = \{(t, id)\}$.

Note that occurrence bindings and synchronized occurrences also apply to singleton sets of identifiers. In that case any "synchronized" occurrence $\tilde{o} \in \tilde{O}(a, \{id\})$ is an occurrence $\tilde{o} \in O(t, id)$. In case the system has only one transition t_a labeled with a, synchronized occurrences and transition occurrences coincide: $\tilde{O}(a, \{id\}) = O(t, id)$; see for instance t_3 (notify) in Fig. 4. This allows us to consider from now on only synchronized transition occurrences.

We may now give a formal inductive definition of the partially-ordered runs of a system with dynamic unbounded synchronization. This definition still does not consider specific semantics of channels of a proclet system which we discuss in Sect. 5.

Definition 5 (Runs with dynamic unbounded synchronization). *Let* $S = (\{N_1, \ldots, N_n\}, m_\nu, C)$ *be a proclet system. The set of runs of* S *with dynamic unbounded synchronization is the smallest set* $R(S)$ *such that*

1. *The initial run* $\pi_0 = (B, \emptyset, \emptyset, \lambda)$ *that provides for each id token on a place* p *in* m_ν *a corresponding condition, i.e.,* $|\{b \in B | \lambda(b) = \{(p, id)\}| = m_\nu(p)(id)$, *is in* $R(S)$.

2. *Let $\pi \in R(S)$, let $a \in \Sigma$, and $I \subseteq \mathcal{I}$ be finite. Let $\tilde{o} \in \tilde{O}(a, I)$ be a synchronized occurrence of I at a.*
   (a) *$\tilde{o}$ is* enabled *in π iff exactly the preconditions of $\tilde{o}$ occur at the end of π, i.e., $pre_{\tilde{o}} \subseteq \max \pi := \{b \in B_\pi | b^\bullet = \emptyset\}$ and, w.l.o.g, $con_{\tilde{o}} \cap X_\pi = \emptyset$*
   (b) *if $\tilde{o}$ is* enabled *in π then appending $\tilde{o}$ to π is a run of S, formally $\pi \cup \tilde{o} \in R(S)$*

The run of Fig. 5 is a run of the proclet system in Fig. 4 according to Definition 5. The wrong run in Fig. 3 is *not* a run of that proclet system: by events e_6 instance package1 moves to state done, hence events e_7 and e_8 cannot be synchronized occurrences in instance package1 that lead to state ready.

The following lemma states that the labeling λ of the runs of S defines a local injective homomorphism to the syntax of S in the same way as the labeling in the runs of a classical net N defines a local injective homomorphism the syntax of N [9]. The lemma follows straight from the inductive definition and from Lemma 1.

Lemma 2. *Let $S = (\{N_1, \ldots, N_n\}, m_\nu, C)$ be a proclet system, let $\pi \in R(S)$. Then π is an occurrence net where*

1. *for each $b \in B_\pi$: $\lambda(b) = \{(p, id)\}$ for some $p \in P_i, 1 \leq i \leq n$*
2. *for each $e \in E_\pi$: $\lambda(e) = \{(t_1, id_1), \ldots, (t_k, id_k)$ with $t_j \in T_i, 1 \leq i \leq n$ for each $1 \leq j \leq k$, and $\ell(t_j) = \ell(t_y)$ for all $j, y = 1, \ldots, k$*
3. *$|\{b \in B | \lambda(b) = \{(p, id)\}| = m_\nu(p)(id)$, for each $p \in P_i, 1 \leq i \leq n$*
4. *for each instance id occurring in π and the run $\pi|_{id} = (B', E', G', \lambda')$ of instance id holds: for each event $e \in E'$ with $\lambda(e) = \{(t, id)\}$, $t \in N_i, 1 \leq i \leq n$, λ' defines an injective homomorphism from $\{e\} \cup {}^\bullet e \cup e^\bullet$ to $\{t\} \cup {}^\bullet t \cup t^\bullet$ in N_i.*

5 Relational Synchronization

While the runs defined in Sect. 4 provide an operational formal semantics for proclet systems, Definition 5 is not restrictive enough to correctly model the intended behaviors. It, for instance, allows two order instances to synchronize with a single package in an occurrence of split, e.g., synchronizing o_{17}, o_{18}, o_2 in Fig. 6. Likewise, Definition 5 allows that a package instance created by order17 does not synchronize with order17 but with order18 at bill. In this section, we rule out such behavior by constraining the occurrence bindings via the channels. We first define *cardinality constraints* and *correlation constraints* for synchronization at channels, and then provide the semantics of both constraints.

5.1 Cardinality and Correlation Constraints

We adopt the notion of cardinality constraints known from data modeling, and applied in relational process structures [27], to channels (t_i, t_j) between two proclets N_i and N_j, see Definition 1. Each channel constraint specifies how many

occurrences of t_i may synchronize with how many occurrences of t_j in one synchronized occurrence. As each occurrence of t_i and t_j is related to a specific instance, we thus constrain which instances of N_i and N_j may synchronize in one step. A *cardinality constraint* specifies for each transition of a channel a lower bound l and an upper bound u between 0 and ∞. We will later formalize that in any synchronized occurrence, the number of occurrences of the transition has to be between these two bounds. For example, according to the channel constraint for (t_2, t_5) in Fig. 4, exactly one instance of order synchronizes with one or more instances of package at any occurrence of split.

To ensure consistency of synchronization over multiple steps, we adopt the concept of *correlation identifiers*. Correlation in message-based interaction between processes [20,21] is achieved by specifying a particular attribute of a message as a *correlation attribute* a. A process instance R receiving a message m from an unknown sender instance S *initializes* a local *correlation key* $k := m.a$ with the value of a in m. To send a response to the unknown sender instance S, R creates a message m_2 where attribute $m_2.a := k$ holds the value of k. If R later only wants to receive a response from S (and no other sender instance), R will only accept a message m_3 where $m_3.a = k$. This is called *matching of correlation keys*. This concept can be extended to multi-instance interaction, using local data for correlation, instead of dedicated correlation keys [16].

For the synchronous interaction model proposed in this paper, we define correlation over synchronous channels instead of messages. A channel c_{init} can be labeled to *initialize* a correlation set S, meaning all instances which synchronize at a step over c_{init} are in S. Another channel c_{match} can be labeled to *match* a previously initialized correlation set S, meaning the instances synchronizing at a step over c_{match} have to be either a subset of S or equal to S. For example, according to the correlation constraints at channels (t_2, t_5) (split) and (t_4, t_{10}) (bill), exactly the package instances which were created at split by an order instance must synchronize at bill with the same order instance. In contrast, the package instances synchronizing at a deliver step with a delivery instance only have to be a *subset* of the package instances loaded into the delivery.

Definition 6 (Channel constraints). *Let* $S = (\{N_1, \ldots, N_n\}, m_\nu, C)$ *be a synchronous proclet system.*

1. *A* cardinality constraint *for* C *is a function card which specifies for each channel* $c = (t, t') \in C$ *a lower and an upper bound for each transition* t *and* t' *in the channel* $card(c) = ((l, u) : (l', u'))$ *with* $0 \leq l \leq u \leq \infty$, $0 \leq l' \leq u' \leq \infty$.
2. *A* correlation constraint $K = (C_{init}, C^{\subseteq}_{match}, C^{=}_{match})$ *for* C *specifies a set of* initializing *channels* $C_{init} \subset C$, *a set of* partially matching *channels* $C^{\subseteq}_{match}$, *and a set of* fully matching *channels* $C^{=}_{match}$, *where all sets of channels are pair-wise disjoint.*

Given a cardinality constraint card and a set corr of correlation constraints for C, *we call* $S = (\{N_1, \ldots, N_n\}, m_\nu, C, card, corr)$ *a* constrained *synchronous proclet system.*

Channel constraints are visualized as shown in Fig. 4. Cardinality constraints are indicated at the ends of the edges indicating channels, we use the standard abbreviations of ? for $(0,1)$, 1 for $(1,1)$, $*$ for $(0,\infty)$, and $+$ for $(1,\infty)$. The channels in Fig. 4 are constrained by $1:+$ and $1:1$ cardinality constraints. Correlation constraints are annotated in the middle of a channel, marking initialization with $\mathsf{init}K$, partial matching with $\subseteq \mathsf{K}$, and full matching with $= \mathsf{K}$. The proclet system in Fig. 4 has two correlation constraints O and D. Note that in general, a channel may be part of different correlation constraints, initializing in one constraint while matching in another constraint.

5.2 Semantics of Cardinality Constraints

A cardinality constraint of a channel (t,t') restricts the number of occurrences of t and t' synchronizing in a step. We formalize this by restricting the *occurrence bindings* of a label (Definition 3) to adhere to the channel constraints of all transitions involved. For any set O of transition occurrences, let $O[t] = \{o \in O \mid \ell(e_o) = t\}$ be the set all occurrences of transition t in O. Further, we write $O[t_1,t_2] = \{(o_1,o_2) \mid o_1 \in O[t_1], o_2 \in O[t_2]\}$ for the *relation of occurrences* between t_1 and t_2. By Lemma 1, we may also write $\tilde{O}[t_1,t_2] = O[t_1,t_2]$ for the synchronized occurrence $\tilde{O}$ of binding O. Writing $inst(o) = id$ for the instance of an occurrence $o \in O(t,id)$, we obtain $inst(O[t_1,t_2]) = \{(inst(o_1), inst(o_2)) \mid (o_1,o_2) \in O[t_1,t_2]\}$.

If we interpret a channel (t_1,t_2) between two proclets N_1 and N_2 as a *relation* between two transitions in two different proclets, then $O[t_1,t_2]$ are the "records" of this relation that we can observe in O. Assuming there is a relational data model that underlies the proclet system and provides relational tables for the entities E_1 and E_2 described by N_1 and N_2, then $inst(O[t_1,t_2])$ are the "records" of the relationship between E_1 and E_2. In this spirit, the cardinality constraint only allows occurrence bindings where the constraints on this relation is satisfied.

Definition 7 (Occurrence binding satisfies cardinality constraint). *Let $S = (\{N_1,\dots,N_n\}, m_\nu, C, card, corr)$ be a constrained proclet system. Let $O_I \in O(a,I)$ be an occurrence binding for instances $I \subseteq \mathcal{I}$ at $a \in \Sigma$.*

The occurrence binding O_I satisfies *the cardinality constraint $card(t_1,t_2) = ((l_1,u_1):(l_2,u_2))$ of channel $(t_1,t_2) \in C$ iff if $\ell(t_1) = a$, then $l_1 \le |O_I[t_1]| \le u_1$ and $l_2 \le |O_I[t_2]| \le u_2$. We then also say that the synchronized occurrence $\tilde{O}_I$ of O_I is an* occurrence *of channel (t_1,t_2).*

O_I satisfies the cardinality constraints of S iff O_I satisfies the cardinality constraint of each channel of S. We then also say that the synchronized occurrence $\tilde{O}_I$ satisfies the cardinality constraints of S.

For example, considering the occurrences $o_1, o_2, o_3, o_{17}, o_{17}$ in Fig. 6, the occurrence binding $\{o_{17}, o_1, o_3\}$ satisfies the $1:+$ cardinality constraint of (t_2,t_5) in Fig. 4, whereas the occurrence binding $\{o_{17}, o_{18}, o_3\}$ does not satisfy this constraint. A run of S can only be extended with a synchronized occurrence $\tilde{O}$ if the occurrence binding O satisfies the cardinality constraints of S.

Following the reasoning of Sect. 3, $1:+$ and $1:1$ cardinality constraints have the most natural interpretation from an operational perspective as in any occurrence of the channel, the "1" side can take a local "coordinating" role for synchronization with the "+" side.

5.3 Semantics of Correlation Constraints

Correlation between different transition occurrences is a *behavioral property*. Thus, we will not extend the notion of state of a proclet system to hold values of correlation properties which can be initialized and matched. Rather, we give a behavioral definition over the history of the run.

A correlation constraint may be initialized multiple times in a run, each time with the relation $inst(\tilde{O}[t,t'])$ of instances involved in the synchronized occurrence $\tilde{O}$. For example, consider the synchronized occurrence $\tilde{O}_{\mathsf{split},17}$ of $(t_2,17),(t_5,1),(t_5,3)$ at split (synchronization of e_2, e_2', e_2'') in the run of Fig. 8. According to Fig. 4, $\tilde{O}_{\mathsf{split},17}$ initializes the correlation constraint O with $inst(\tilde{O}_{\mathsf{split},17}[t_2,t_5]) = \{(17,1),(17,3)\}$. Likewise, the synchronized occurrence $\tilde{O}_{\mathsf{split},18}$ of $(t_2,18),(t_5,2)$ at split (synchronization of e_{16}, e_{16}') initializes the constraint O with $inst(\tilde{O}_{\mathsf{split},18}[t_2,t_5]) = \{(18,2)\}$.

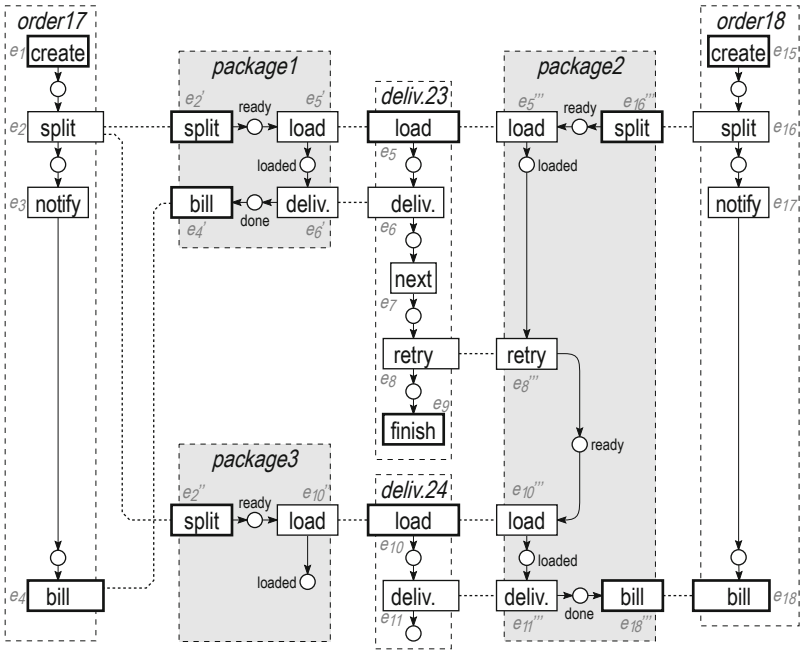

Fig. 8. The synchronized occurrence of order17 and package1 at bill violates the matching constraint = O of Fig. 4

While initialization can occur arbitrarily, matching is constrained: any synchronized occurrence involving channel (t_4, t_{10}) at bill has to match an initialization of O that occurred before. For instance, the synchronized occurrence $\tilde{O}_{\mathsf{bill},17}$ of $(t_4, 17), (t_{10}, 1)$ at bill (synchronization of e_4, e'_4) in Fig. 8 satisfies the cardinality constraint $1 : +$ of channel (t_4, t_{10}) and involves the relation $inst(\tilde{O}_{\mathsf{bill},17}[t_4, t_{10}]) = \{(17, 1)\}$. This occurrence violates the matching constraint $= \mathsf{O}$ of channel (t_4, t_{10}) because the initialization of cardinality constraint O that precedes $\tilde{O}_{\mathsf{bill},17}$ (at e_4, e'_4) is the synchronized occurrence $\tilde{O}_{\mathsf{split},17}$ (at e_2, e'_2, e''_2), and $inst(\tilde{O}_{\mathsf{bill},17}[t_4, t_{10}]) = \{(17, 1)\} \neq inst(\tilde{O}_{\mathsf{split},17}[t_2, t_5]) = \{(17, 1), (17, 3)\}$. In contrast, the synchronized occurrence $\tilde{O}'_{\mathsf{bill},17}$ shown in Fig. 5 satisfies the correlation constraint $= \mathsf{O}$.

Definition 8 (Occurrence satisfies correlation constraint in a run). *Let $S = (\{N_1, \ldots, N_n\}, m_\nu, C, card, corr)$ be a constrained proclet system. Let π be a labeled occurrence net. Let $\tilde{o}_m \in \tilde{O}(a_m, I_m)$ be a synchronized occurrence of instances I_m at a_m such that $pre_{\tilde{o}_m} \subseteq \max \pi$.*

$\tilde{o}_m$ satisfies *correlation constraint $(X_{init}, X^{\subseteq}_{match}, X^{=}_{match})$ in π iff*

1. *if $\tilde{o}_m$ is an occurrence of a channel $(t_m, t'_m) \in X^{\subseteq}_{match}$, then there exists a synchronized occurrence $\tilde{o}_i \in \tilde{O}(a_i, I_i)$ of an initializing channel $(t_i, t'_i) \in X_{init}$ such that*
   (a) *the initializing occurrence $\tilde{o}_i$ is in π, i.e., $\tilde{o}_i \subseteq \pi$,*
   (b) *and precedes the matching occurrence $\tilde{o}_m$, i.e., for the event $e_{\tilde{o}_i}$ of $\tilde{o}_i$ exists a path in π to* some *$b \in pre_{\tilde{o}_m}$: $(e_{\tilde{o}_m}, b) \in G^+$, and*
   (c) *the relation of instances involved in $\tilde{o}_m$ matches the relation of instances involved in $\tilde{o}_i$, i.e., $id(\tilde{o}_m[t_m, t'_m]) \subseteq id(\tilde{o}_i[t_i, t'_i]))$*
2. *if $\tilde{o}_m$ is an occurrence of a channel $(t_m, t'_m) \in X^{=}_{match}$, then additionally $id(\tilde{o}_m[t_m, t'_m]) = id(\tilde{o}_i[t_i, t'_i]))$ has to hold.*

$\tilde{o}_m$ satisfies the correlation constraints of S iff $\tilde{o}_m$ satisfies each correlation constraint in $corr$.

5.4 Runs of a Constrained Proclet System

We can now easily extend Definition 5 to limit the runs of a proclet system to those allowed by the cardinality and correlation constraints.

Definition 9 (Runs of a constrained proclet system). *Let $S = (\{N_1, \ldots, N_n\}, m_\nu, C, card, corr)$ be a constrained proclet system. The set of runs of S is the smallest set $R(S)$ such that*

1. *The initial run $\pi_0 \in R(S)$ as in Definition 5.*
2. *Let $\pi \in R(S)$, let $a \in \Sigma$, and $I \subseteq \mathcal{I}$ be finite. Let $\tilde{o} \in \tilde{O}(a, I)$ be a synchronized occurrence of I at a.*
   (a) *$\tilde{o}$ is* enabled *in π iff*
      i. *exactly the preconditions of $\tilde{o}$ occur at the end of π, i.e., $pre_{\tilde{o}} \subseteq \max \pi$,*

ii. $\tilde{o}$ *satisfies the cardinality constraints card of* S *(Definition 7), and*
iii. $\tilde{o}$ *satisfies the cardinality constraints corr of* S *in* π *(Definition 8).*

(b) if $\tilde{o}$ *is* enabled *in* π *then appending* $\tilde{o}$ *to* π *is a run of* S*, formally* $\pi \cup \tilde{o} \in R(S)$.

The occurrence net of Fig. 5 is a run of the constrained proclet system in Fig. 4 (assuming all events connected by dashed lines, such as e_2, e_2', e_2'' are synchronized into a single event). The occurrence net of Fig. 8 is not a run of that system. However, the system of Fig. 4 cannot ensure termination of delivery instances: finish should only occur when all packages have been handled. The proclet system of Fig. 9 ensures correct termination through an extended package life-cycle and an additional channel. The system also illustrates that a proclet may interact with more than two other proclets, in this case a return process that must be completed for each undelivered package prior to billing.

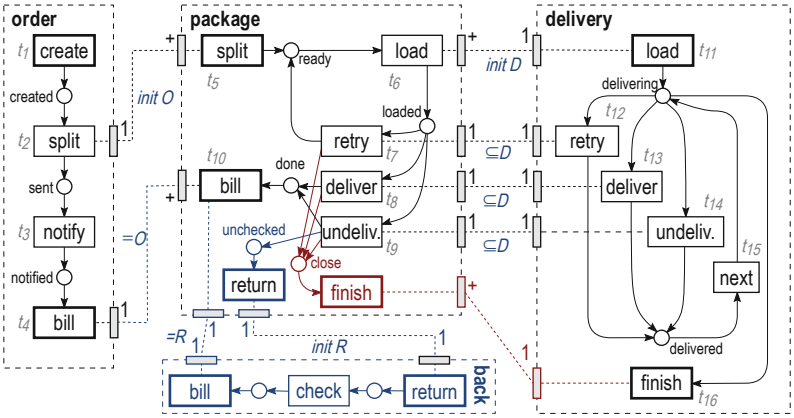

Fig. 9. Extension of the proclet system of Fig. 4

6 Conclusion

We have shown that by reifying a behavioral many-to-many relation into its own entity with a behavioral model, the behavior of processes with many-to-many interactions can be described both visually clear and formally precise. The foundational concept is that of *dynamic unbounded synchronization*, which can be seen as a generalization of the synchronization in artifact-choreographies with 1-to-1 relationships proposed by Lohmann and Wolf [15]. We have shown that this basic behavioral model can be extended orthogonally with cardinality and correlation constraints that limit the allowed synchronization much in the same way as guards in Coloured Petri Nets limit firing modes. We believe that

this restriction to purely behavioral concepts allows adoption and integration with other more data-aware modeling techniques such as [19,27].

The only "higher-level" concept required by our model are case identity tokens as in ν-nets, and we only use equality of token identities, and subsets of pairs of token identities. This suggests that verification results on ν-nets [18] may be lifted to our model. Though, we suspect undecidability to arise in the general case for correlation constraints testing for equality.

Further, the structure of a "reified" process model in behavioral second normal form (having only 1-to-1 and 1-to-many synchronizations) allows some further reflection on the structure of such processes. The asynchronous message exchange between two processes is replaced by an entity explicitly describing the interaction. This entity such as the package in our running example, is a "passive" (data) object, as changes to its state are due to activities in the processes operating on it, making the processes "active" entities. This leads for many, but possibly not all use cases to a bipartite structure of entities. Two "active" processes never synchronize directly as each transition describes a task that is manipulating a "passive" object; two "passive" objects never synchronize directly as they require an "active" process to trigger the necessary state change. In this understanding, an asynchronous message channel is passive object, synchronizing with sender and receiver, and each instance of the channel is a message. Lohmann and Wolf [13] have shown that this interpretation allow for formulating new types of research questions: given a set of objects, synthesize the active processes synchronizing them; given a set of processes, synthesize the passive objects realizing their synchronization.

References

1. van der Aalst, W.M.P.: The application of petri nets to workflow management. J. Circ. Syst. Comput. **8**(1), 21–66 (1998)
2. van der Aalst, W.M.P., Artale, A., Montali, M., Tritini, S.: Object-centric behavioral constraints: integrating data and declarative process modelling. In: Proceedings of the 30th International Workshop on Description Logics, Montpellier. CEUR Workshop Proceedings, vol. 1879. CEUR-WS.org (2017)
3. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: a framework for lightweight interacting workflow processes. Int. J. Cooperative Inf. Syst. **10**(4), 443–481 (2001)
4. van der Aalst, W.M.P., Mans, R.S., Russell, N.C.: Workflow support using proclets: divide, interact, and conquer. IEEE Data Eng. Bull. **32**(3), 16–22 (2009)
5. Calvanese, D., Montali, M., Estañol, M., Teniente, E.: Verifiable UML artifact-centric business process models. In: CIKM 2014, pp. 1289–1298. ACM (2014)
6. Cohn, D., Hull, R.: Business artifacts: a data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. **32**(3), 3–9 (2009)

7. Damaggio, E., Deutsch, A., Hull, R., Vianu, V.: Automatic verification of data-centric business processes. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 3–16. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23059-2_3
8. Desel, J., Erwin, T.: Hybrid specifications: looking at workflows from a run-time perspective. Comput. Syst. Sci. Eng. **15**(5), 291–302 (2000)
9. Engelfriet, J.: Branching processes of petri nets. Acta Inf. **28**(6), 575–591 (1991)
10. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.: Many-to-many: some observations on interactions in artifact choreographies. In: Eichhorn, D., Koschmider, A., Zhang, H. (eds.) ZEUS 2011. CEUR Workshop Proceedings, vol. 705, pp. 9–15. CEUR-WS.org (2011)
11. van Hee, K.M., Sidorova, N., Voorhoeve, M., van der Werf, J.M.E.M.: Generation of database transactions with petri nets. Fundam. Inform. **93**(1–3), 171–184 (2009)
12. Li, G., de Carvalho, R.M., van der Aalst, W.M.P.: Automatic discovery of object-centric behavioral constraint models. In: Abramowicz, W. (ed.) BIS 2017. LNBIP, vol. 288, pp. 43–58. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59336-4_4
13. Lohmann, N.: Compliance by design for artifact-centric business processes. Inf. Syst. **38**(4), 606–618 (2013)
14. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 321–341. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73094-1_20
15. Lohmann, N., Wolf, K.: Artifact-centric choreographies. In: Maglio, P.P., Weske, M., Yang, J., Fantinato, M. (eds.) ICSOC 2010. LNCS, vol. 6470, pp. 32–46. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17358-5_3
16. Meyer, A., Pufahl, L., Batoulis, K., Fahland, D., Weske, M.: Automating data exchange in process choreographies. Inf. Syst. **53**, 296–329 (2015)
17. Montali, M., Calvanese, D.: Soundness of data-aware, case-centric processes. STTT **18**(5), 535–558 (2016)
18. Montali, M., Rivkin, A.: Model checking petri nets with names using data-centric dynamic systems. Formal Asp. Comput. **28**(4), 615–641 (2016)
19. Montali, M., Rivkin, A.: DB-Nets: on the marriage of colored petri nets and relational databases. In: Koutny, M., Kleijn, J., Penczek, W. (eds.) Transactions on Petri Nets and Other Models of Concurrency XII. LNCS, vol. 10470, pp. 91–118. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-55862-1_5
20. OASIS: Web Services Business Process Execution Language, Version 2.0, April 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
21. OMG: Business Process Model and Notation (BPMN), Version 2.0, January 2011. http://www.omg.org/spec/BPMN/2.0/
22. Reijers, H.A., et al.: Evaluating data-centric process approaches: does the human factor factor in? Softw. Syst. Model. **16**(3), 649–662 (2017)
23. Reisig, W.: Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-33278-4
24. Rosa-Velardo, F., Alonso, O.M., de Frutos-Escrig, D.: Mobile synchronizing petri nets: a choreographic approach for coordination in ubiquitous systems. Electr. Notes Theor. Comput. Sci. **150**(1), 103–126 (2006)
25. Rosa-Velardo, F., de Frutos-Escrig, D.: Name creation vs. replication in petri net systems. Fundam. Inform. **88**(3), 329–356 (2008)

26. Steinau, S., Andrews, K., Reichert, M.: Modeling process interactions with coordination processes. In: Panetto, H., Debruyne, C., Proper, H., Ardagna, C., Roman, D., Meersman, R. (eds.) OTM 2018, Part I. LNCS, vol. 11229. Springer, Cham (2018)
27. Steinau, S., Andrews, K., Reichert, M.: The relational process structure. In: Krogstie, J., Reijers, H.A. (eds.) CAiSE 2018. LNCS, vol. 10816, pp. 53–67. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91563-0_4