Multi-instance data behavior in BPMN

Maximilian König[1,*], Mathias Weske[1]

[1]*Hasso Plattner Institute, Prof.-Dr.-Helmert-Straße 2-3, 14482 Potsdam*

Abstract

Process models have long-since been used to capture business processes for documentation, communication, analysis, and enactment purposes. Most of the commonly used modeling languages, e.g., BPMN, focus on the control flow of involved activities and participants, while mostly neglecting the data perspective despite the latter gaining more and more relevance for today's companies. This is especially true for multiple instances of the same data, e.g., multiple items being ordered by the same customer. The BPMN standard's concepts fail to capture such behavior sufficiently. In addition, their textual descriptions lack conciseness when it comes to the concepts' semantics. Therefore, this paper proposes new semantics for multi-instance data objects including splitting and merging behavior which lays the foundation for further investigation of multi-instance behavior of process data flow in general.

Keywords

Process Modeling, BPMN, Data Semantics, Multi-Instance Behavior, Colored Petri Nets

1. Introduction

Business process management (BPM) is a mature discipline that is employed to manage business processes throughout their lifecycle [1]. A core concept are process models to visualize the different dimensions of such business processes. Business Process Modeling and Notation (BPMN) [2] provides a standardized process modeling language that is widely adopted for that purpose. It follows an activity-centric approach, hence primarily focusing on the control flow of processes. That entails the order in which certain steps have to be executed, who executes them, and which messages are sent between participants, to realize a business goal. However, the data perspective, i.e., which information is required for or produced by the execution of certain activities, is largely neglected [3].

To that end, BPMN provides concepts to visualize data, namely data objects. They act as an abstraction of concrete data schemas. Changing values throughout a process execution are represented as different states of these data objects. An example is an order placed at an online shop. However, issues arise when a single process instance has to deal with multiple instances of such a data object, e.g., when considering the individual items of the placed order. It cannot easily be represented that some of these items are in stock while others are not.

To address these shortcomings, this paper proposes an extension to the behavior defined by

ER2023: Companion Proceedings of the 42nd International Conference on Conceptual Modeling: ER Forum, 7th SCME, Project Exhibitions, Posters and Demos, and Doctoral Consortium, November 06-09, 2023, Lisbon, Portugal
*Corresponding author.

maximilian.koenig@hpi.de (M. König); mathias.weske@hpi.de (M. Weske)
0000-0002-2244-1179 (M. König); 0000-0002-3346-2442 (M. Weske)

the BPMN standard. Thereby, the capabilities to handle multiple instances of data objects in different states are improved. Besides a textual description of the behavior, a concise semantics is presented as well by defining a translation to colored Petri nets, which is a commonly used formalism for that purpose [4, 5].

The remainder of this paper has the following structure: In section 2, the concepts used throughout the paper are introduced and the conceptual gap to be closed is highlighted. Based thereon, section 3 describes the proposed behavior addressing that gap. Related work is then presented in section 4 before section 5 discusses the paper's results and concludes.

2. Background

This section provides an overview of the concepts used in the remainder of the paper. First, BPMN and its data-related constructs are described. Afterward, colored Petri nets are introduced.

2.1. BPMN

Figure 1 shows an example order handling process represented as a BPMN process model. First, a received order is split into the requested items. After determining whether they are in stock, available items are packaged and shipped, while unavailable items are ordered to be shipped to the customer through a third party. Once all items are shipped or ordered to be shipped, the invoice is sent to the customer and the process ends.

The elements of BPMN process models [2] required for this work can be divided into control flow nodes and data nodes. Control flow nodes comprise activities, i.e., the actions that are performed during a process, events, i.e., instantaneous occurrences of various natures that have an impact on process execution, and gateways representing decisions and concurrency in a process. These nodes are interconnected through control flow arcs defining their temporal and

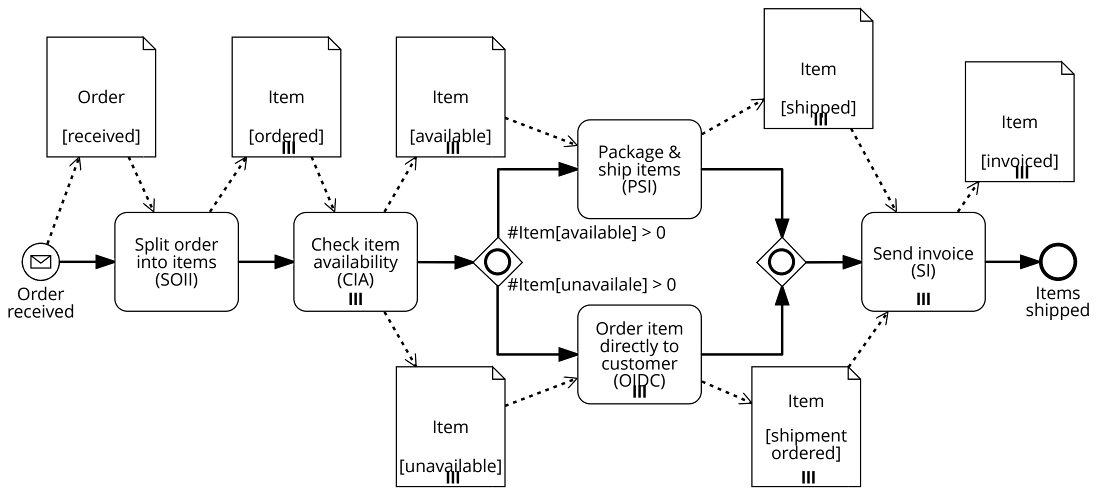

Figure 1: Order processing example process.

logical order. Data nodes represent the data available in the process that is produced by control flow nodes or serves as their precondition. In BPMN, these nodes are called data objects. They are defined through a data class and a state denoted in square brackets, e.g., *Order[received]*. In addition to an enabled ingoing control flow, all ingoing data objects must be available for an activity to be executed. Once finished, they transition ingoing data objects to new states if there is an outgoing object of the same class in a different state. For example, activity *Package & ship items* in Figure 1 transitions data objects of class *Item* from state *available* to *shipped*. In contrast, *Split order into items* creates new *Item* data objects, since they are only outgoing.

Both data objects and activities can be marked to describe multi-instance behavior. Activities tagged with three horizontal bars can be executed multiple times sequentially, three vertical bars indicate that the instances may run concurrently. Regarding data objects, three vertical bars signify a collection of multiple data objects of the same class in the same state. If a data object collection (DOC) is the precondition of a multi-instance activity, the number of activity instances is determined by the number of elements in the DOC. For example, *Procure item* will be executed as often as there are items in state *unavailable*.

The BPMN standard has a number of limitations impacting the applicability of the introduced concepts. On the one hand, it describes most concepts only textually and therefore lacks concise semantics. For example, the meaning of data object states is not defined at all. On the other hand, it defines that only a single instance of every data object, collection or not, may exist at the same time [2, p. 206]. Looking at Figure 1, that would imply that the availability check of ordered items must always result in either all items being available or all items being unavailable. In reality, that is not the case. To express that in BPMN, it would be required to have multiple collections of the same data class in different states that can be merged and split according to the process requirements. Therefore, this paper introduces a semantics for such behavior in BPMN as a starting point for further investigation of multi-instance data behavior in processes.

2.2. Colored Petri nets

The formalism of our choice for such a semantics will be colored Petri nets (CPNs) [6], an extension of classical Petri nets [7] which are commonly used to define the behavior of BPMN [3, 4, 5]. Petri nets are directed bipartite graphs consisting of places, transitions, and directed arcs between them. The set of places with an arc toward a transition is called its preset, the set of places with an arc from a transition is its postset. The state of a net is defined as a distribution of *tokens* over the set of places. Firing a transition consumes a single token from each place in the preset and produces one in each place of the postset.

A major shortcoming of traditional Petri nets is that tokens are not distinguishable, which prevents an efficient representation of multiple instances in a single net. CPNs address that

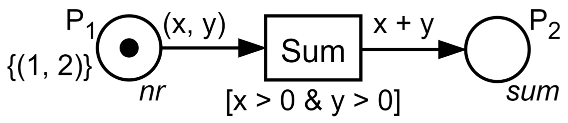

Figure 2: Example colored Petri net.

by introducing types for places, so-called colorsets. Tokens in a place contain concrete data values of its type that can be bound to variables. The variables can then be used to create and modify the data of other tokens. That happens using arc expressions and guards on transitions, which may consist of complex statements including function calls. An example can be found in Figure 2. Place P_1 has colorset $nr\ :\ number \times number$ holding tokens containing a tuple of numbers. In contrast, colorset $sum\ :\ number$ describes tokens with only a single numerical value stored in P_2. Upon firing, transition *Sum* consumes a token from P_1 whose values are captured in variables x and y of type number. However, the transition can only fire if the guard is fulfilled, stating that both values must be greater than 0. Finally, it produces a new token in P_2 holding the sum of x and y.

3. BPMN data object collection behavior

To address the shortcomings of the BPMN standard regarding data object collections, this section proposes an updated semantics allowing for the splitting and merging of data object collections (DOCs). To that end, we provide a textual description of the intended behavior alongside a translational semantics to CPNs.

Before describing the new behavior, we need to introduce a deviation from the BPMN standard. As described in section 2, it allows only a single instance of a data object (collection) at the same time. However, in the context of collections, we must loosen this requirement. It should instead state that *a data object collection in the BPMN model always refers to all data object instances in the referenced state in the current process instance* at a time, without prohibiting the existence of data objects in different states. Thereby, we pave the way to split the ordered items in Figure 1 into a collection of available and a collection of unavailable items. Following the intention of the standard, this definition ensures that it is always unambiguous what an element in the model refers to at instance level. Further, we define that collections of size 0 are not sufficient as a data precondition for control flow nodes. In other words, there must be at least one data object instance in the required state before an activity reading the respective collection gets enabled. As a result of these definitions, the semantics of multiple in- and outgoing DOCs of the same class for a single activity becomes unclear. Before, it was always an exclusive choice, since only a single instance could exist. Now, we must allow that, depending on the process state, only one of the collections, or both collections are required for activity enablement or updated through activity execution. For further specification, BPMN input/output sets can be used, which we will not consider in this work. In the following paragraphs, we define the state transition, splitting, merging, and creation operations for data object collections.

State Transition. Activities reading a DOC and writing the same collection in a different state perform a state transition. That means that all data objects of the ingoing collection are transitioned to the new state. Hence, the number of data objects added to the collection with the new state is the number of elements in the read collection. Figure 3 shows the mapping of that behavior to a colored Petri net at the example of the single-instance activity *Package & ship items*. The activity is represented as a transition and the in- and outgoing control flow as places with colorset *cf*. For every state of every data class, a place is created.

These places are called global places and are unique in the entire process mapping. If a created place represents a data object collection, we assign the colorset *doc* : *string*[] containing a list of IDs, else we assign colorset *do* : *string* for single-instance data objects. The transition is bidirectionally connected to the places representing the in- and outgoing DOCs. Upon firing, it checks that ingoing DOCs have at least one element before appending all their elements to the DOC in the target state. At the same time, the read token is reset to an empty list. Thereby, all items in state *available* are transitioned to state *shipped* in the example. Note that we use the +-operator for concatenating and appending to lists.

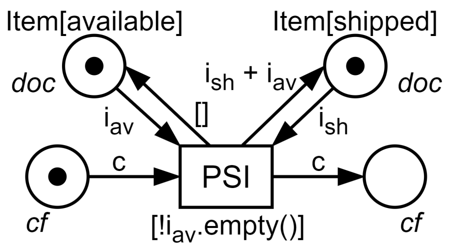

Figure 3: Mapping of a data collection state transition.

Splitting. To split a data object collection into multiple collections, we make the assumption that the decision on the target state is made for every element individually. Therefore, we only allow BPMN multi-instance activities to split a collection. Thereby, we also ensure that the activity is executed as often as there are instances in the collection. In the example, *Check item availability* determines for each ordered item whether it is in stock or not, and transitions it to the respective state. A CPN representation of that activity can be found in Figure 4. In contrast to the previous mapping, the individual processing of every instance requires more than a single transition. Instead, the activity is split into a beginning (CIA_b) and terminating (CIA_t) transition. The former checks the availability of at least one element in the read collection and resets it to an empty list to avoid concurrent access. In addition, a local *doc* place is used to keep track of the yet unprocessed data object instances. From there, a transition for every possible target state can change the first element of the collection to the new state, stored in another local *doc* place. These transitions are enabled until the list is empty. Once it is empty, the terminating

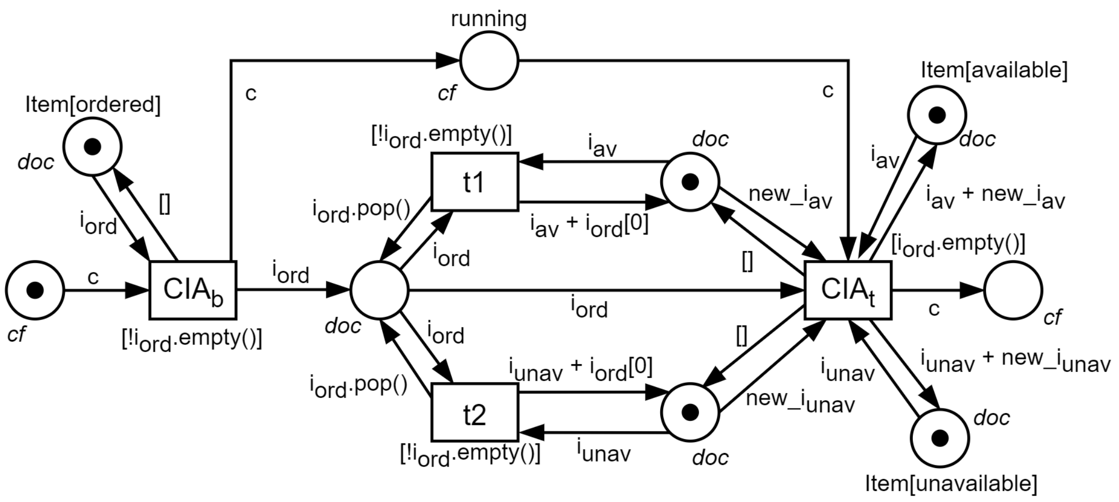

Figure 4: Mapping of the splitting of two DOCs.

transition can fire, resetting the local places to empty lists, and appending the respective data object instances (stored in variables *new_i_{av}* and *new_i_{unav}*) to the global DOCs. In addition to the places for the data objects, another control flow place indicates that the activity is still running.

Merging. After introducing the splitting of collections, we must also provide means to merge them again. Therefore, an activity may read multiple DOCs of the same data class and write only a single DOC of that data class. The result will be that all instances in the ingoing DOCs will be transitioned to the state of the outgoing DOC, and will hence be appended to that collection. If that activity is a multi-instance activity, the number of instances corresponds to the sum of the length of all ingoing DOCs of the same class.

To capture the behavior as a multi-instance activity, we use a similar mapping to the splitting, as visualized in Figure 5. We also create beginning and terminating transitions that read and write the respective control flow and data places. The beginning transition resets all ingoing *doc* places and its guard ensures that at least one of them contains at least one element. Afterward, a local place per read state is created, serving the same purpose as the local places in the CPN for the splitting behavior. Then, a transition per newly created place (i.e., *t1* and *t2*) transitions one data object instance to the target state, which is also stored in a local place. Once all elements of all ingoing DOCs have been processed, the guard of the terminating transition evaluates to true and adds all transitioned data object instances to the global collection.

Please note that, if a single-instance activity is used for DOC merging, a simplified mapping can be applied following the same rules as the state transition with multiple ingoing *doc* places. Analogously, a multi-instance activity performing a 1:1 state transition can be mapped using the same rules as described for the merging behavior.

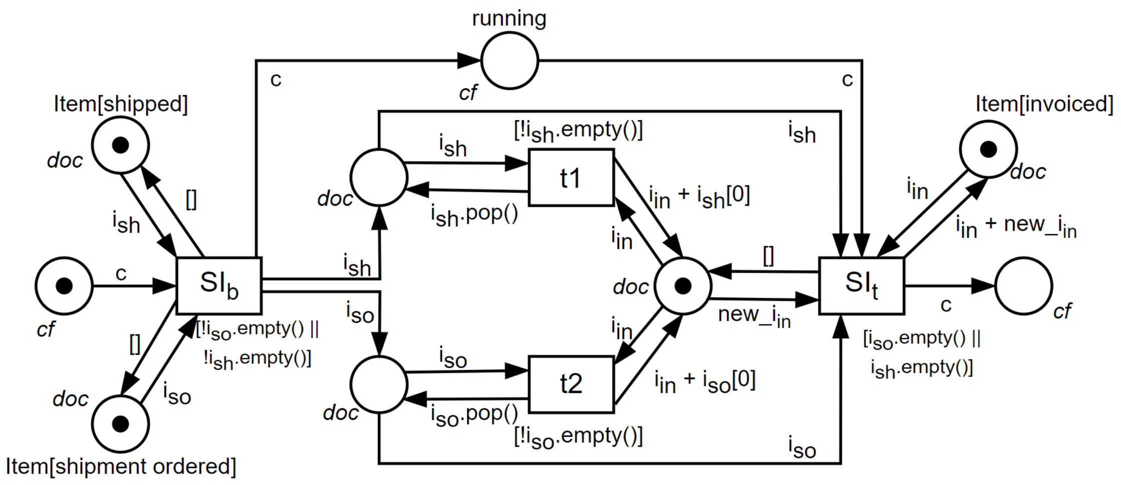

Figure 5: Mapping of the merging of two DOCs.

Creation. For the creation of new data object instances, an activity must have an outgoing DOC without a DOC of the same class as precondition. We assume that at least one new instance must be created if that activity is executed. In Figure 6, an exemplary CPN representation is depicted. We again split the activity into a beginning and terminating transition. Between them, we create a control flow place indicating that the activity is still running. In addition, another transition can fire as long as there is a token in the *running* place to create new instances of the respective data class. Once at least one instance has been created, the terminating transition is enabled and, upon firing, appends the newly created collection to the global DOC.

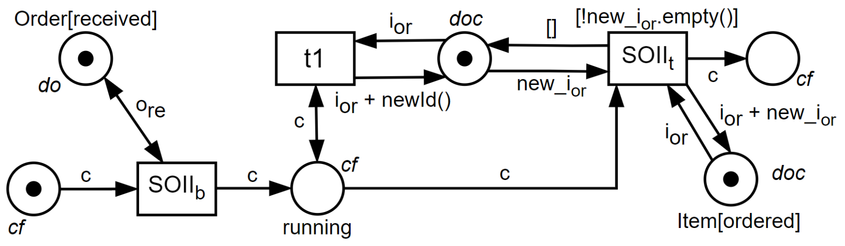

Figure 6: Mapping of the creation of a new DOC.

4. Related work

In related literature, a number of approaches can be found defining translational semantics for BPMN including data concepts. For example, Ramadan et al. and Dechsupa et al. define mapping rules to CPNs covering a wide range of complex control flow constructs as well as some data concepts [4, 5]. However, neither approach deals with multi-instance data. Using a different approach, Meyer et al. first extend BPMN with annotations to visualize complex data dependencies before assigning an execution semantics based on SQL queries [3]. While they consider data object collections, they do not support list operations such as splitting or merging. Instead, they only consider a single list per data class. Corradini et al. translate BPMN models to a Backus-Naur form representation and enrich it with a state notion [8]. Their mapping rules aim at multi-instance behavior, especially regarding BPMN collaboration diagrams, and include data objects. Yet, multi-instance data objects are excluded from consideration. Combi et al. present an extension of BPMN data nodes by introducing activity views, assigning each activity an SQL statement describing the accessed data [9]. Therewith, they allow the modeler to hide complexity from the process model. However, it remains unclear how multiple instances of the same data can be handled in that approach.

5. Discussion & Conclusion

In this paper, we proposed new semantics for BPMN data object collections (DOCs) in order to allow splitting and merging behavior. To do so, we had to lift the assumption that only a single DOC may exist at the same time. Afterward, we introduced the BPMN constructs allowing the new list operations and described their behavior informally using text and formally using a translation to colored Petri nets.

The presented semantics serve as a starting point for future research on multi-instance behavior in process data flow. Currently, only a fraction of BPMN's concepts are covered by our approach, and many complex constructs such as subprocesses, data stores, and cancelation require additional consideration. Further, we did not introduce a sophisticated data locking mechanism as described in [10].

Another aspect to be investigated is the derivation of BPMN modeling guidelines from the newly defined behavior. For example, splitting a data object collection into several requires the handling of the case that one of them is empty after the activity terminates. Otherwise, dead branches in the process may be the result. In Figure 1, that is accomplished using an inclusive split. However, that may not be possible in every situation. Automated detection of such erroneous behavior using contemporary Petri net analysis methods would be desirable.

Finally, adopting the extension in other modeling languages is an interesting line of research. For example, YAWL [11] comes with multi-instance behavior while lacking data concepts.

References

[1] M. Weske, Business Process Management - Concepts, Languages, Architectures, Third Edition, Springer, 2019. doi:10.1007/978-3-662-59432-2.

[2] OMG, Business Process Model and Notation (BPMN), Version 2.0.2, Technical Report, Object Management Group, 2014. https://www.omg.org/spec/BPMN/2.0.2/PDF.

[3] A. Meyer, L. Pufahl, D. Fahland, M. Weske, Modeling and enacting complex data dependencies in business processes, in: BPM 2013. Proceedings, volume 8094 of *LNCS*, Springer, 2013, pp. 171–186. doi:10.1007/978-3-642-40176-3_14.

[4] C. Dechsupa, W. Vatanawood, A. Thongtak, Hierarchical Verification for the BPMN Design Model Using State Space Analysis, IEEE Access 7 (2019) 16795–16815. doi:10.1109/ACCESS.2019.2892958.

[5] M. Ramadan, H. G. Elmongui, R. Hassan, BPMN formalisation using coloured petri nets, in: SEA, 2011, pp. 83–90.

[6] K. Jensen, L. M. Kristensen, L. Wells, Coloured petri nets and CPN tools for modelling and validation of concurrent systems, Int. J. Softw. Tools Technol. Transf. 9 (2007) 213–254. URL: https://doi.org/10.1007/s10009-007-0038-x. doi:10.1007/s10009-007-0038-x.

[7] T. Murata, Petri nets: Properties, analysis and applications, Proc. IEEE 77 (1989) 541–580. doi:10.1109/5.24143.

[8] F. Corradini, C. Muzi, B. Re, L. Rossi, F. Tiezzi, Animating multiple instances in BPMN collaborations: From formal semantics to tool support, in: BPM 2018. Proceedings, volume 11080 of *LNCS*, Springer, 2018, pp. 83–101. doi:10.1007/978-3-319-98648-7_6.

[9] C. Combi, B. Oliboni, M. Weske, F. Zerbato, Conceptual modeling of inter-dependencies between processes and data, in: SAC 2018, ACM, 2018, pp. 110–119. doi:10.1145/3167132.3167141.

[10] A. Meyer, Data Perspective in Business Process Management, PhD Thesis, Universität Potsdam, 2015.

[11] W. M. P. van der Aalst, A. H. M. ter Hofstede, YAWL: yet another workflow language, Inf. Syst. 30 (2005) 245–275. doi:10.1016/j.is.2004.02.002.