Instantiation Semantics for Process Models

Gero Decker[1] and Jan Mendling[2]

[1] Hasso-Plattner-Institute, University of Potsdam, Germany
gero.decker@hpi.uni-potsdam.de
[2] Queensland University of Technology, Brisbane, Australia
j.mendling@qut.edu.au

Abstract. Although several process modeling languages allow one to specify processes with multiple start elements, the precise semantics of such models are often unclear, both from a pragmatic and from a theoretical point of view. This paper addresses the lack of research on this problem and introduces the CASU framework. The contribution of this framework is a systematic description of design alternatives for the specification of instantiation semantics of process modeling languages. We classify six of the most prominent languages by the help of this framework. Our work provides the basis for the design of new correctness criteria as well as for the formalization of EPCs and extension of BPMN. It complements research such as the workflow patterns.

1 Introduction

Process modeling techniques have been widely adopted by businesses and other organizations for documenting their operations. In this context, process models describe business activities along with their temporal and logical relationships within business processes of the organization, either as reflection of the status quo or as a road map for change. Process models are also used for configuring information systems, in particular workflow systems, that create and handle singular cases (or instances) according to the rules defined in the model.

There are several business process modeling languages that define the basic elements for constructing individual business process models. In this paper we consider the six most prominent ones and assume that the reader has some basic understanding of their syntax and semantics. They are in historical order:

– Petri nets (PN) [1], a formalism to specify processes with concurrency. In particular, we will focus on Open Workflow Nets (oWFNs) [2] which extend Workflow nets [3] with interface places.
– Event-driven Process Chains (EPCs) [4], the business process modeling language used within the ARIS framework and the respective toolset [5].
– UML Activity Diagrams (UAD) [6], the process modeling language of UML.
– Yet Another Workflow Language (YAWL) [7], the workflow language that builds on the workflow patterns analysis [8].
– Business Process Execution Language for Web Services (BPEL) [9], the proposed OASIS standard for web service composition and execution.

– Business Process Modeling Notation (BPMN) [10], the OMG standard notation for describing business processes.

In practice these languages tend to be used in different design phases: while executable processes and workflows are often defined as BPEL or YAWL models, Petri nets and UAD specify processes in a way that easily supports software development. EPCs and BPMN are meant to serve as a high-level description of business operations. For an introduction to these languages refer to [11].

In this paper we focus on the problem of process instantiation and its representation in process models. This problem is little understood in theory and practice, and it poses a considerable challenge for mapping conceptual models to executable processes. In particular, such conceptual models tend to have a significant amount of control flow errors like deadlocks [12, 13]. 57% of these errors in the SAP Reference Model, i.e. 102 out of 178 [14, p.150], can be traced back to an unsound combination of multiple start and end events. The BPMN specification acknowledges that the semantics of multiple start events are often unclear [10, p.36]. Even though there has been a considerable amount of academic contributions on the formalization of control flow constructs in all the six mentioned process modeling languages, these works tend to abstract from the problem of process instantiation. Most notably, the original workflow patterns [8] do not cover any instantiation patterns. A revised set of control-flow patterns [15] discusses the effect of external signals on the execution of a process instance (WCP-23 Transient Trigger, WCP-24 Persistent Trigger), but not on instantion.

Against this background, this paper provides a twofold contribution. First, we define a conceptual framework to describe process instantiation semantics as assumed in different process modeling languages. This framework is called CASU since it builds on four pillars: instantiation creation (C), control threads activation (A), event subscription (S), and unsubscription (U). Second, we use this framework to classify and compare the instantiation semantics of the six mentioned modeling languages. In particular, this comparison reveals additional problems of mapping BPMN to BPEL beyond those discussed in [16–18].

The remainder of this paper is structured as follows. In Section 2 we discuss how process instantiation is represented in Petri nets, EPCs, UML-AD, YAWL, BPEL, and BPMN. We define a set of general concepts to make the approaches comparable. Then, Section 3 introduce the CASU framework to describe different instantiation semantics of process modeling languages. We provide a classification of the languages according to this framework. Section 4 discusses the implications of this work, in particular, its relationship to existing research on the verification of process models as well as potential directions for EPC formalization and BPMN extension. Finally, Section 5 concludes the paper.

2 Background on Process Instantiation

Process Instantiation refers to the action and the rules of creating an instance from a process model. Instantiation requires an initial state to be identified for the newly created instance. In this section we discuss explicit and implicit

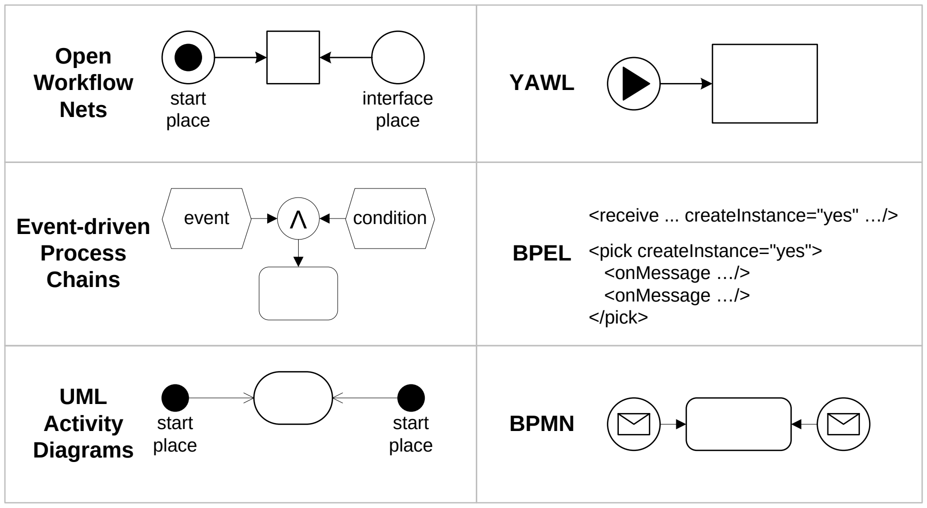

Fig. 1. Entry points in different process modeling languages

definition of an initial state, the role of entry points for it, i.e. start places, start conditions, and start events, as well as the basic architecture for instantiation.

Most prominently, process instantiation requires the definition of the initial state for the new instance. This initial state becomes the starting point for allowed state transitions in the life cycle of the instance. In general there are two ways of defining the initial state of a process instance: explicitly or implicitly. A definition of an initial state is *explicit* if the initial state is part of the definition of the process model. The initial state of a Petri net is traditionally defined explicitly: Murata defines a Petri net as a 5-tuple including places, transitions, arcs, weight function, and the initial marking [19]. The definition of an initial state is *implicit* if it has to be derived from what we call *entry points* of a process model, i.e. model elements without any control flow arc pointing to them. Note that this notion of entry point only refers to the structure of the process model.

Entry points are related to different concepts in process modeling languages, most prominently start places, start conditions, and start events (see Figure 1). In the simplest case, an entry point is a *start place* that receives a control token at the time of instantiation. A *start condition* is a statement about the environment of a process that can be either true or false at a given point in time. Depending on whether that condition yields true or false, the respective entry point is activated, and thus becomes part of the initial state of the instance. Entry points can also refer to *start events*. An event in that sense can be understood as a record of an activity in a system [20]. Therefore, every event has a defined time of occurrence. Start events are special events that respond to records related to a process type.

In some cases the initial state can be derived as unambiguously from entry points as by giving the initial marking of a Petri net explicitly. Modeling languages like Workflow nets and YAWL restrict the number of entry points to one

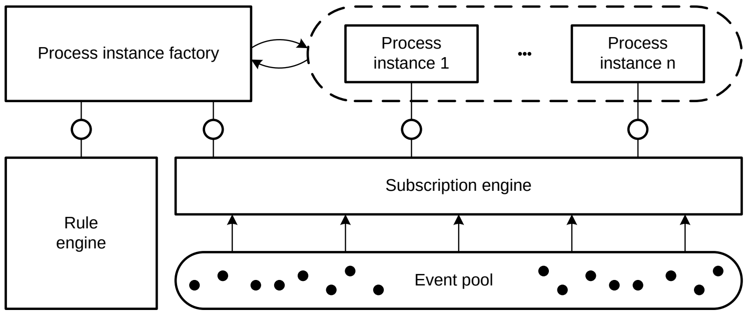

Fig. 2. Process execution environment

unique node such that the initial state assigns a single token to the unique start place. Things are less clear if there are *multiple entry points* in the model. Open Workflow nets extend Workflow nets with an interface: syntactically they are classified as start events according to our definition of an entry point. Yet, they cannot trigger the creation of a new instance. In UML Activity Diagrams the initial state is derived unambiguously by assigning a control token to each initial node [6]. Note that receive activities in UAD are no entry points according to our definition since they have to receive control from an incoming flow to be activated. In contrast to the mentioned languages, the original definition of EPCs [4] does not define a notion of state. Therefore, it is not a priori clear how a combination of entry points, i.e. EPC start events, maps to an initial state. An unambiguous way of deriving the initial state in this case would be to activate all entry points while creating an instance. In contrast to that, the initial state of an EPC is defined non-deterministically [21, 22]. The start events of an EPC are often used to represent both events and conditions [23, p.134]. As a consequence, different initial states are allowed, but there is at most informal information, e.g. in the text labels of the start events, that gives hints when and which initial state has to be used. In BPMN start events can be used (they are optional) to describe instantiation. The specification distinguishes subtypes for message, timer, rule, link, and multiple start events [10]. If there are multiple start events modeled they should be interpreted as independent events, i.e. each event creates a new instance. Still, it is possible to describe the dependency upon multiple events if the events flow to the same activity in the process and this activity specifies the event dependency. In BPEL alternative start events can be defined using the pick activity [9]. Multiple message activities can have the “createInstance” attribute set to “yes”. Upon instantiation subscriptions are issued for all those receive and onMessage elements that did not trigger instantiation and that do not belong to the same pick element the triggering activity belonged to.

As a conceptual framework for those cases where start events apply, we assume a subscription infrastructure including a rule engine involved in process instantiation. Process instances and process instance factories can subscribe for particular events and can have conditions evaluated. Process instance factories typically have durable subscriptions for events, i.e. the subscription takes place at deployment time of a process model and unsubscription at the moment of undeployment. As the name indicates, process instance factories create process instances as a result of certain event occurrences. Subscriptions by process instances have a shorter life span. Subscription can only become effective after the moment of process instantiation. Unsubscription can take place any time during the life time of a process instance, however, it must be before termination.

Figure 2 illustrates the subscription framework. Events occur in an event pool and can be observed by a subscription engine. Subscriptions and unsubscriptions can be issued by the different process instances and by the process instance factory. The subscription engine in turn notifies them upon availability of a corresponding event which in turn is then consumed.

3 A Framework for Process Instantiation

This section discusses different design choices for defining process instantiation semantics. We establish a framework building of four aspects of instantiation that have to be specified by a process modeling language:

Creation (C): When has a new instance to be created?
Activation (A): Which entry points are activated?
Subscription (S): For which start events are subscriptions created?
Unsubscription (U): How long are subscriptions kept?

Based on the first letters we refer to the framework as the CASU framework.

3.1 When to create a new instance?

In essence we can distinguish cases where the process model does not specify when an instance has to be created (C-1), where the process model defines conditions before an instance can be created (C-2 and C-3), and where the process model specifies in response to which event an instance is created (C-4 and C-5). Please note that it is not reasonable to create an instance when a condition is true. While an event is consumed, a condition would remain true and trigger a cascade of new instances before it becomes false at some stage.

C-1 Ignorance The process model is ignorant of instantiation condition. The instantiation of a process instance is controlled by the process environment, and no triggering events are defined.
Example: A process model describes that supply needs must be identified before a request for quote is set up. However, it is not defined what triggers the first activity. Figure 3 shows a corresponding YAWL net.

C-2 Single Condition Filter The start condition of a process model specifies under which circumstances it is possible to create a new process instance.
Example: The start condition of a loan process model specifies that the applicant must be of full age.

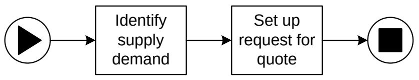

Fig. 3. C-1 Ignorance in a YAWL net

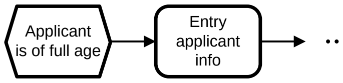

Fig. 4. C-2 Single condition filter in EPC

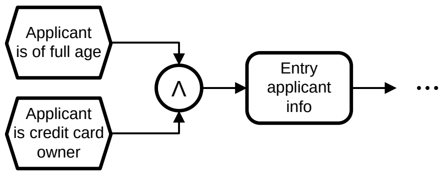

Fig. 5. C-3 Multi condition filter in EPC

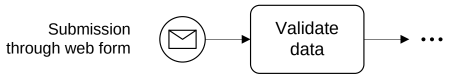

Fig. 6. C-4 Single event trigger in a BPMN diagram

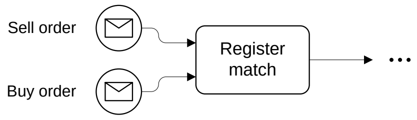

Fig. 7. C-5 Multi event trigger in a BPMN diagram

C-3 Multi Condition Filter Multiple start conditions define a complex condition when a process is allowed to be instantiated.
Example: A loan process model of another bank defines two start conditions: the applicant must be of full age and credit card owner (Figure 5).

C-4 Single Event Trigger The consumption of one start event triggers instantiation.
Example: A Police process model describes that citizens can file charges via a website, triggering instantiation by submitting the web form (Figure 6).

C-5 Multi Event Trigger Consumption of multiple events triggers instantiation. There is a potential race between different process definitions (factories) in case of overlapping event types. When the last required event becomes available, the instance is created and all required events are consumed at one point in time.
Example: Buy and sell events arising from the stock market are automatically correlated triggering trade processes (Figure 7).

3.2 Which entry points are activated?

There are different ways to express in a process model which entry points are activated at instantiation time. An initial state (A-1) defines explicitly the activation. Depending on the type of entry points the activation can be specified implicitly: all start places (A-2), true conditions (A-3), occurred events (A-4), or a combination of the latter (A-5).

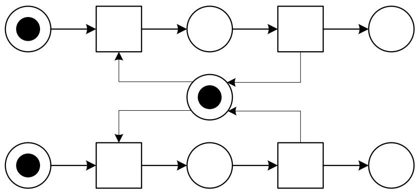

Fig. 8. A-1 Initial state in a Petri net

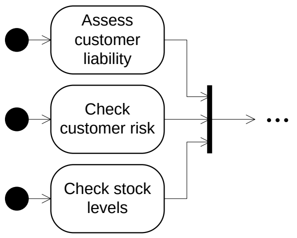

Fig. 9. A-2 All start places in a UML Activity Diagram

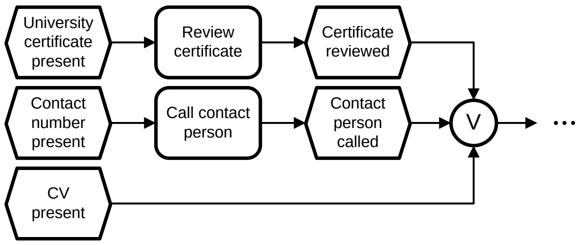

Fig. 10. A-3 True conditions as an EPC

A-1 Initial State The process model explicitly defines the state each process instance is initially in.
Example: A model includes an initial marking with several tokens. One of them represents a semaphore, the others two streams of control (Figure 8).

A-2 All Start Places The process model implicitly defines an initial state through its structure: all start places receive a token at instantiation.
Example: An ordering process model has three start nodes, each receiving a token upon instantiation. These tokens enable the three parallel activities "assess customer liability", "check customer risk" and "check stock levels" (Figure 9).

A-3 True Conditions The environment checks conditions at instantiation and activates the respective start condition nodes.
Example: A job application process model contains the following start conditions: "University certificate present", "contact number present" and "CV present". Only if a contact phone number is present, the former employer is called for getting further information on the candidate. The university certificate must be reviewed if present and the contact person is called if a phone number is present (Figure 10).

A-4 Occurred Events In this case all consumed events (one or more) are mapped to an activation of control threads in the process model. There may be start events that do not belong to this set.
Example: An invoice management process model describes four start events (Figure 11): "paper invoice received", "electronic invoice received", "delivery notification received" and a timer event "second Tuesday of the month".

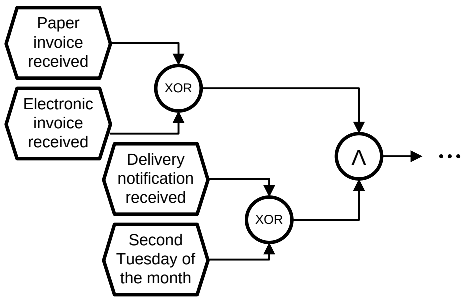
Paper invoice received
Electronic invoice received
XOR
Delivery notification received
Second Tuesday of the month
XOR
∧
...

Fig. 11. A-4 Occurred events as an EPC

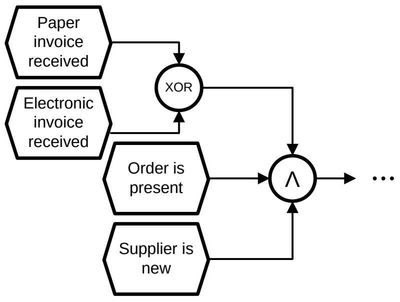

Fig. 12. A-5 Occurred events plus conditions as an EPC

Once a pair of corresponding invoice and delivery notification have arrived or an invoice has arrived and the timer event has occurred, a process instance is created. Upon instantiation those control threads are activated that originate in the respective start events.

A-5 Occurred Events plus Conditions In this case all consumed events map to activated control threads. Additionally, branches can be activated if start conditions yield true at instantiation time.
Example: In a second invoice management process model (Figure 12), the start events “paper invoice received” and “electronic invoice received” appear again. Additionally, there are start conditions “order is present” and “supplier is new”. For each start condition that is fulfilled upon instantiation the corresponding control thread is activated.

3.3 For which non-activated start events are subscriptions created?

When there are start events a decision has to be made whether event subscriptions are made for those remaining start events that did not lead to the instantiation of process. We distinguish the case of subscriptions being created for all of the remaining start events (S-1), for none of them (S-2), or for those that are required for proper execution (S-3).

S-1 All Subscriptions For those start events that are not activated at instantiation time, there is an event subscription created for the process instance. I.e., the remainder branches may be activated later by respective events. Example: A couple applies for a mortgage. With opening the case, there are already several events subscriptions activated that matter later like providing sketch of the house, sale contract, etc. (Figure 13).

S-2 No Subscriptions In this case there are no event subscriptions created for the process instance. I.e., an entry point thread will be either activated at instantiation time or never.
Example: A stock purchase process can be triggered by either a customer representative directly entering the purchase request or by the customer entering the request in a web form (Figure 14).

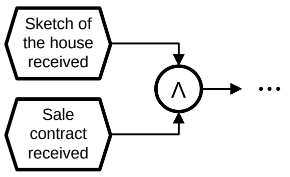

Fig. 13. S-1 All subscriptions as an EPC

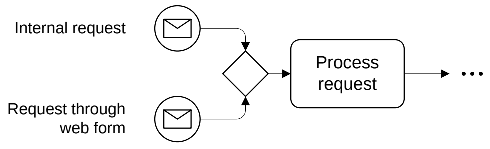

Fig. 14. S-2 No subscriptions in a BPMN diagram

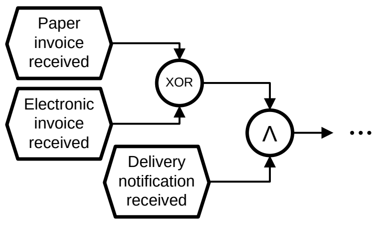

Fig. 15. S-3 Reachable subscriptions as an EPC

S-3 Reachable Subscription Only those event subscriptions are activated that might be required later to complete the process instance properly.
Example: In an invoice management process model similar to that of A-4 there are three start events: the receipt of a paper invoice, of an electronic invoice or a delivery notification can trigger instantiation. As only one invoice is needed for proper termination, only a subscription for the delivery notification is issued in case the receipt of an invoice triggered instantiation. In the other case, subscriptions for both invoice types are issued (Figure 15). BPEL provide respective functionality with the pick as a start activity.

3.4 How long are subscriptions kept?

There may be different ways to unsubscribe for events. In the simplest case, they are kept until consumption (U-1) or at least until the process terminates (U-2). Earlier unsubscriptions can be defined based on timers (U-3), on events (U-4), or on proper completion (U-5). Listing 1 shows respective concepts in BPEL.

U-1 Until Consumption The process cannot terminate before all event subscriptions have led to the consumption of a respective event. A subscription of an instance is never deactivated.
Example: A process model describes the activities of a logistics hub, where containers with RFID tags arrive while routing information for the containers is fed into the system through a different channel. Either the container or its routing information might arrive first, but the process cannot terminate before both are there.

U-2 Until Termination As soon as the process fulfills a termination condition, all subscriptions are deactivated, and the process terminates. This

Listing 1 Unsubscription in BPEL

```
<scope>
 <eventHandlers><onAlarm name="timeout" .. /></eventHandlers>
 <flow>
  <receive name="rcvDeliveryNot" .. createInstance="yes">
   <correlations><correlation set="inv" initiate="join"/></correlations>
  </receive>
  <sequence><pick createInstance="yes">
   <onMessage name="rcvEInvoice" .. >
    <correlations><correlation set="inv" initiate="join"/></correlations>
   .. </onMessage>
   <onMessage name="rcvPInvoice" .. >
    <correlations><correlation set="inv" initiate="join"/></correlations>
   .. </onMessage>
  </pick> ..
  <exit/></sequence>
 </flow>
</scope>
```

seems to be often assumed by EPC modelers.
Example: A BPEL process reaches an exit activity terminating all subscriptions.

U-3 Timer-based After a certain period of time after instantiation, all or individual event subscriptions are cancelled.
Example: A timeout of a pick activity in a BPEL process deactivates an event subscription.

U-4 Event-based If one of alternative events is consumed, the others are not more considered, and deactivated.
Example: In an invoice management process model similar to that of A-4 is represented in BPEL. A pick as a start activity defines alternative start events: the receipt of a paper or an electronic invoice or of a paper or an electronic delivery notification. If the receipt of an invoice triggered instantiation and a paper delivery notification arrives, the subscription for an electronic delivery notification is removed. This also applies for the other combinations.

U-5 Proper Completion An event gets deactivated when proper completion is guaranteed for the current marking if the event is not consumed.
Example: A process model describes how reallocation of passengers to flights of partner airlines works at an airport's service desk. This process model has two start events "Passenger arrives at service desk" and "flight voucher arrives". Immediately upon arrival of the passenger a seat is allocated and immediately upon arrival of an electronic flight voucher this voucher is checked for validity. As vouchers can also be issued in paper form, the passenger might carry this voucher with him. As soon as the voucher is checked, the flight is billed and the passenger can board the aircraft.

3.5 A Classification of Instantiation Semantics

Before assessing the six process modeling languages, the interrelationships between the patterns presented in the CASU framework need to be discussed briefly. While patterns A-1 (Initial state) and A-2 (All start places) do not require support for any particular of the C-patterns, A-3 (True conditions) requires the specification of single or multiple start conditions (C-2, C-3). All patterns related to start events (A-4, A-5, all S-patterns and all U-patterns) rely on the possibility to specify single or multiple event triggers (C-4, C-5). The results of the classification are summarized below in Table 1.

Open Workflow Nets (oWFN) are a particular class of Petri nets that are ignorant of the circumstances of their instantiation (C-1). Furthermore, they define an initial state (A-1). They also include a distinct set of interface places that can be used for message passing. The input places of the interface follow all subscription semantics (S-1) that are kept until completion (U-2).

Start events (also called triggers) are used in *Event-driven Process Chains (EPC)* to represent when a process starts. The cases C-4 (Single Event Trigger) with XOR-join and C-5 (Multiple Event Trigger) with AND-join are described in [5] and [23], but no formalization is available. Although not recommended, the decomposition of EPCs often leads to subprocesses that have conditions as start nodes (C-2 and C-3), e.g. if the subprocess starts immediately after a decision [5, pp.250] or to express external dependencies [5, pp.131]. If the trigger or condition is not made explicit in the start event label, the EPC remains ignorant of the instantiation (C-1). Depending on which of multiple start events and start conditions apply the respective initial state is derived (A-3, A-4, A-5). The whole area of subscription (S-Patterns) and unsubscription (U-Patterns) related to non-activated entry points has not been explicitly defined for EPCs. There seem to be some inconsistent interpretations that need to be resolved in future work: While Rump assumes that there are no subscriptions [21], the concept of external dependency appears to suggest either S-1 (all subscriptions) or S-3 (reachable subscriptions) [5, pp.131]. In neither case unsubscription is discussed. Table 1 reflects this ambiguity by using the $\emptyset$ character.

Although *UML Activity Diagrams (UAD)* include event consumption and event production as first-class citizens of the language, these concepts are not used in the context of process instantiation. The events required for process instantiation are beyond the scope of UAD models. That way UAD only supports C-1 (Ignorance) among the C-patterns. The start nodes are essentially start places that all receive a token upon instantiation (A-2). The remaining patterns A-3 through A-5, the S-patterns and the U-patterns are not supported.

Yet Another Workflow Language (YAWL) concentrates on the control and data flow within process instances. There is one distinguished "start condition" per process model, however, definitions of how and when instantiation takes place are not part of YAWL models. The notion of start conditions or start events are not present. Therefore, it does not support C-1 through C-4, A-3 through A-5, none of the S-patterns as well as none of the U-patterns. The initial state of a process instance is implicitly given: there is exactly one start place that receives

Patterns	oWFN	EPCs	UAD	YAWL	BPEL	BPMN
C-1 Ignorance	+	+	+	+	+	+
C-2 Single Condition Filter	−	+	−	−	−	−
C-3 Multi Condition Filter	−	+	−	−	−	−
C-4 Single Event Trigger	−	+	−	−	+	+
C-5 Multi Event Trigger	−	+	−	−	−	+
A-1 Initial State	+	−	−	−	−	−
A-2 All Start Places	−	−	+	+	−	−
A-3 True Conditions	−	+	−	−	−	−
A-4 Occurred Events	−	+	−	−	+	+
A-5 Occurred Events plus Cond.	−	+	−	−	−	−
S-1 All Subscriptions	+	∅	−	−	+	−
S-2 No Subscriptions	−	∅	−	−	−	+
S-3 Reachable Subscription	−	∅	−	−	+	−
U-1 Until Consumption	−	∅	−	−	+	−
U-2 Until Termination	+	∅	−	−	+	−
U-3 Timer-based	−	∅	−	−	+	−
U-4 Event-based	−	∅	−	−	+	−
U-5 Proper Completion	−	∅	−	−	−	−

Table 1. Instantiation in different process modeling languages

a token upon instantiation (A-2). That way, YAWL is similar to UAD in terms of instantiation semantics with an additional restriction to exactly one start place. Workflow nets share the same instantiation profile with YAWL.

The *Business Process Execution Language (BPEL)* completely lacks the notion of start conditions (C-2, C-3). Process instantiation might be undefined in the case of abstract BPEL (C-1) and must be defined for executable BPEL processes. Here, instantiation is always triggered through individual message receipts (C-4), described in incoming message activities (receive or pick) having the attribute "createInstance" set to yes. Defining combinations of messages that are required for instantiation is not possible (C-5). The start state of a process instance is solely determined by the one start event that triggered process instantiation. Therefore, BPEL does not support A-1, A-2, A-3 and A-5. Subscriptions are issued for all those incoming message activities that have not been involved in process instantiation (S-1). Whenever an onMessage branch of a pick element receives the initial message, no subscriptions are issued for the other onMessage branches of the same pick element. That way, BPEL supports S-3. As illustrated in Listing 1, BPEL supports patterns U-1 through U-4. Termination before having received all start messages can be achieved through the exit element or through throwing exceptions. Timer-based unsubscription (U-3) can be realized by surrounding message activities with a scope that has an onAlarm event handler attached. Event-based unsubscription happens in the context of pick elements (U-4). Beyond these triggers for unsubscription, BPEL does not support pattern U-5.

The *Business Process Modeling Notation (BPMN)* does not include the notion of start conditions. The only entry points available are start events. C-4 (Single Event) is the default case for BPMN processes, where an individual event specified in the model leads to process instantiation. However, no specification might be given, that way realizing Ignorance (C-1). The BPMN specification mentions a special case realizing C-5: Multiple start events are connected to an activity indicating that all start events must have occurred before the activity can start [10, p.36]. BPMN also supports A-4 (Occurred Events) via event-based gateways. However, if C-5 applies there is a slightly different token flow in comparison with the standard semantics of BPMN: While typically each token flowing into an activity leads to a separate activity instance, only one activity instance is created in the presence of the C-5 scenario. All other A-patterns are not supported, in particular, neither A-1 (Initial State), nor A-2 (All Start Places), and the notion of start conditions is absent (A-3 and A-5). Although the BPMN specification is slightly ambiguous regarding multiple start events, we interpret that each start event consumption will lead to a separate process instantiation. No subscriptions for other start events are issued within a newly created process instance (S-2). As a result, BPMN does not support patterns S-1, S-3 and none of the U-patterns.

4 Discussion

In this section we discuss the implications of this research. First, we focus on the suitability of correctness criteria. We then give directions for a formalization of EPC instantiation semantics before finally identifying potential extensions to BPMN. Please note that the formalizations of process modeling languages that we are aware of tend to abstract from the complexity of the instantiation problem, e.g. [24, 25, 22].

Several *correctness criteria* for process models are available including soundness, relaxed soundness, EPC soundness, and controllability. For an overview see [11]. The classical *soundness* property demands a process to complete properly and to have no dead transitions [3]. It can be used to check process models with a unique start and end elements such as Workflow nets and YAWL nets. Multiple start nodes in UAD can be bundled with an AND-join such that it becomes also applicable for them. The *relaxed soundness* property can be used for languages with multiple entry points such as EPCs. It basically requires (1) that an OR-split is introduced to bundle all start elements, and (2) checks whether each node participates in at least one execution sequence that leads to proper completion [26]. The property of *EPC soundness* is stricter: it demands that for every start element there exists an initial marking that guarantees proper completion [22]. This property assumes pattern S-2 (no subscriptions). For oWFNs the property of *controllability* was defined to deal with interface places. An oWFN is essentially controllable if there exists a strategy to interact with it such that it terminates properly [2]. The interesting characteristic of this property is that it is basically applicable for any combinations of subscription and unsubscription patterns including those that consider reachability and proper completion. Still it does not distinguish models for which only one particular strategy exists from those which permit different strategies.

In Section 3.5 we already mentioned that a specification of *EPC instantiation semantics* is missing. The concept of external dependency [5] and its representation as a start event highlights the need to discuss subscription semantics in detail. A start event with external dependency semantics does not trigger the creation of an instance, but defines a point of synchronization with an event from outside the process. We basically see two options to support such external dependencies: either S-3 (reachable subscriptions) or based on S-1 (all subscriptions) with U-5 (proper completion). In the case of S-3 those event subscriptions are activated that might be required later to complete the process instance properly. In this case a reachability graph analysis, e.g. using [22], would be required. While this solution would prevent some deadlocks at AND-joins that merge paths from start events, it still allows problems with lack of synchronization. In case of S-1 with U-5 some of the latter problems can be avoided since events get unsubscribed if no more needed. Beyond this aspect of the semantics, one has to carefully select a state representation for the subscriptions. If a subscription is defined like a special activity that is active, this has consequences for downstream OR-joins: they keep waiting for the event to occur, potentially forever if the event cannot occur anymore. Therefore, it would be preferable that event subscriptions were not visible in the state representation of this case.

The status of *BPMN* as a standards proposal raises questions how and whether it should support more of the CASU patterns. An important consideration in this regard is most likely to extend it such that it remains consistent with the current semantics. The support of the start condition patterns (C-2 and C-3) would require either the introduction of a new element or the redefinition of the rule events. These options either affect the metamodel or the current semantics which is both undesirable. If the creation support (C-1 to C-5) remains unchanged, also the activation patterns (A-1 to A-5) stay the same. With respect to the subscriptions (S-1 and S-3) there are basically two options: either changing the instantiation semantics of BPMN, or to add a subscription attribute to start events. The latter seems more attractive from a consistency perspective. Using a keepSubscription attribute, one would be able to specify S-1 (all subscriptions) and S-2 (no subscriptions). By using a further attribute subscriptionGroup one would be able to specify instantiation behavior similar to BPEL: all start events with the same subscription group assigned would correspond to message receive activities within one pick element. If there is only one start event for a group, it corresponds to a plain receive activity. Clearly, these concepts require correlation mechanisms such as identified in [27]. Unsubscriptions could equally be captured by additional attributes, e.g. by setting subscriptionTimeout (U-3) and properTermination (U-5) attributes. An event-based unsubscription (U-4) can be handled using the previously mentioned subscriptionGroup: as soon as an event of the group occurs, the others are unsubscribed. As BPEL has a more sophisticated profile in terms of subscription and unsubscription patterns (cf. 1, an extension of BPMN with these aspects could simplify the automatic transformation between the languages.

5 Conclusions

Up to now there has been hardly any research dedicated to instantiation semantics of process models. In this paper we have addressed this research gap and introduced the CASU framework. This framework distinguishes the specification of when to create a new process instance (C), of which control threads to be activated upon instantiation (A), of which remaining start events to subscribe for (S), and of when to unsubscribe from these events (U). It builds on general, language-independent concepts and offers a tool for the systematic description and comparison of instantiation semantics of process modeling languages. As such it complements other works such as the workflow patterns.

Based on the CASU framework, we have classified six of the most prominent languages according to their instantiation semantics. In particular, the different profiles of BPMN and BPEL reveal a source of mapping problems between these two languages that has not been identified before. Furthermore, we have shown that the framework provides a basis to discuss the suitability of correctness criteria, the formalization of EPCs, and potential extensions to BPMN. In future research we aim to utilize the CASU framework for analyzing control-flow errors in EPCs. This could lead to new insights regarding which instantiation semantics process modelers assume. In this regard, the explicit description of instantiation semantics by the help of the CASU framework might eventually help to reduce ambiguity and the number of errors in conceptual process modeling.

References

1. Petri, C.: Fundamentals of a Theory of Asynchronous Information Flow. In: 1962 IFIP Congress, Amsterdam, North-Holland (1962) 386–390
2. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting ws-bpel processes using flexible model generation. Data Knowl. Eng. **64** (2008) 38–54
3. van der Aalst, W.: Verification of Workflow Nets. In Azéma, P., Balbo, G., eds.: Application and Theory of Petri Nets 1997. LNCS 1248 (1997) 407–426
4. Keller, G., Nüttgens, M., Scheer, A.W.: Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken, Germany (1992)
5. Davis, R.: Business Process Modelling With Aris: A Practical Guide. (2001)
6. Object Management Group: UML 2.0 Superstructure Specification. (2005)
7. van der Aalst, W., ter Hofstede, A.: YAWL: Yet Another Workflow Language. Information Systems **30** (2005) 245–275
8. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases **14** (2003) 5–51
9. Alves, A. et al.: Web services business process execution language 2.0. (2007)
10. Object Management Group: Business Process Modeling Notation. (2006)
11. Weske, M.: Business Process Management. Springer-Verlag (2007)

12. Mendling, J., Verbeek, H., van Dongen, B., van der Aalst, W., Neumann, G.: Detection and Prediction of Errors in EPCs of the SAP Reference Model. Data & Knowledge Engineering **64** (2008) 312–329
13. Mendling, J., Neumann, G., van der Aalst, W.: Understanding the occurrence of errors in process models based on metrics. In Meersman, R., Tari, Z., eds.: OTM Conference 2007, Proceedings, Part I. LNCS 4803 (2007) 113–130
14. Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. PhD thesis, Vienna University of Economics and Business Administration (2007)
15. Russell, N., ter Hofstede, A., van der Aalst, W., Mulyar, N.: Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22 (2006)
16. Ouyang, C., Dumas, M., Breutel, S., ter Hofstede, A.: Translating standard process models to bpel. In Dubois, E., Pohl, K., eds.: CAiSE 2006. LNCS 4001, 417–432
17. Mendling, J., Lassen, K., Zdun, U.: Transformation strategies between block-oriented and graph-oriented process modelling languages. International Journal of Business Process Integration and Management **3** (2008)
18. van der Aalst, W., Lassen, K.: Translating unstructured workflow processes to readable bpel: theory and implementation. Inf. Softw. Techn. **50** (2008) 131–159
19. Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE **77** (1989) 541–580
20. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley (2001)
21. Rump, F.: Geschäftsprozessmanagement auf der Basis ereignisgesteuerter Prozess-ketten - Formalisierung, Analyse und Ausführung von EPKs. Teubner (1999)
22. Mendling, J., van der Aalst, W.: Formalization and Verification of EPCs with OR-Joins Based on State and Context. In Krogstie, J., Opdahl, A., Sindre, G., eds.: CAiSE 2007, Proceedings. LNCS 4495 (2007) 439–453
23. Scheer, A.W., Thomas, O., Adam, O.: Process Modeling Using Event-Driven Process Chains. In: Process Aware Information Systems (2005) 119–146
24. Eshuis, R., Wieringa, R.: Tool support for verifying uml activity diagrams. IEEE Trans. Software Eng. **30** (2004) 437–447
25. Puhlmann, F., Weske, M.: Investigations on soundness regarding lazy activities. In Dustdar, S., Fiadeiro, J., Sheth, A., eds.: BPM 2006. LNCS 4102 (2006) 145–160
26. Dehnert, J., Aalst, W.: Bridging The Gap Between Business Models And Workflow Specifications. International J. Cooperative Inf. Syst. **13** (2004) 289–332
27. Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation patterns in service-oriented architectures. FASE 2007, Proceedings (2007) 245–259