Process Instantiation

G. Decker [a] J. Mendling [b,*]

[a] *Hasso-Plattner-Institute, University of Potsdam, Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Germany*

[b] *Humboldt University, Unter den Linden 6, D-10099 Berlin, Germany*

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 (from Creation, Activation, Subscription, Unsubscription). 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 prominent languages by the help of this framework. We validate the relevance of the CASU framework through empirical investigations involving a large set of process models from practice. Our work provides the basis for the design of new correctness criteria as well as for the formalization of Event-Driven Process Chains (EPCs) and extension of the Business Process Modeling Notation (BPMN). It complements research such as the workflow patterns.

Key words: Business Process Modeling, Process Instantiation, Events, Workflow Patterns

* Corresponding author
Email addresses: gero.decker@hpi.uni-potsdam.de (G. Decker), jan.mendling@wiwi.hu-berlin.de (J. Mendling).

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 prominent language 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 (UML-AD) [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 Version 2.0 (BPEL) [9], the proposed OASIS standard for web service composition and execution.
- Business Process Modeling Notation Version 1.1 (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 mod-

els, Petri nets and UML-AD 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.38]. 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 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 threefold 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 analyze a large set of process models. We quantify the occurrence of the CASU-patterns and correlate the patterns with modeling errors. Third, 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

[16,17,18]. This paper is an extension of a previous publication: in comparison to [19] we have added a validation of the framework using real-world process models. Furthermore, we have extended the discussion and provided more detail in the introduction of the framework.

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 introduces 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 presents our empirical findings of analyzing the start event combinations of the SAP Reference Model. Section 5 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 6 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 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. For a formal discussion of the model instantiation concept refer to [20].

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

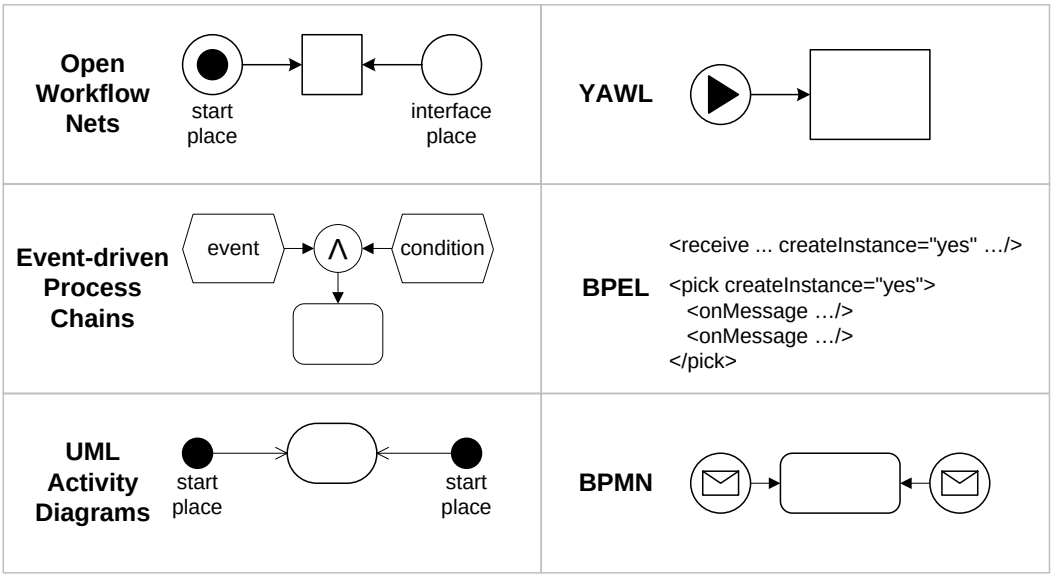

Figure 1. Entry points in different process modeling languages

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 [21]. 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 [22]. 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 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 UML-AD 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 [23,24]. The start events of an EPC are often used to represent both events and conditions [25, 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, conditional, 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.

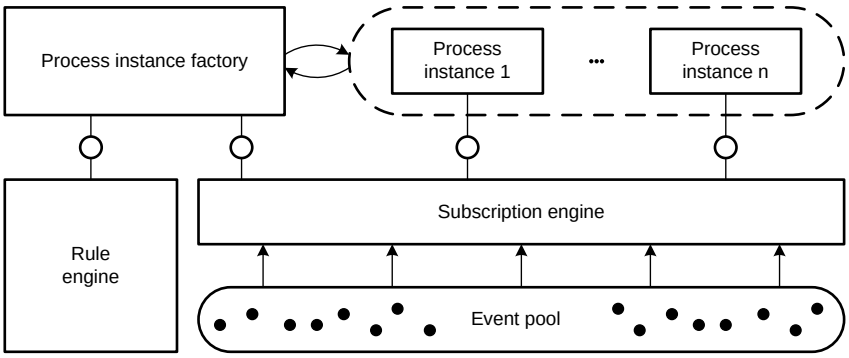

Figure 2. Process execution environment

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. For more details on subscription architectures refer to [26].

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.

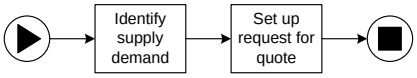

Figure 3. C-1 Ignorance in a YAWL net

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 the 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.

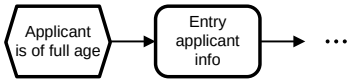

Figure 4. C-2 Single condition filter in EPC

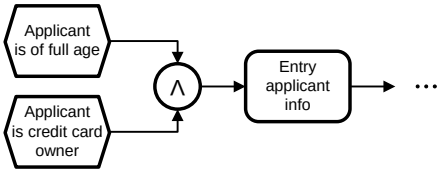

Figure 5. C-3 Multi condition filter in EPC

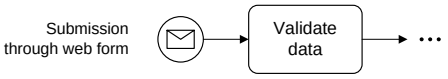

Figure 6. C-4 Single event trigger in a BPMN diagram

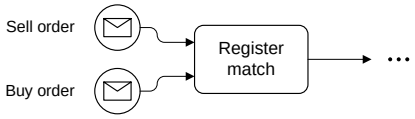

Figure 7. C-5 Multi event trigger in a BPMN diagram

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 (Figure 4).

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 must own a credit card (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).

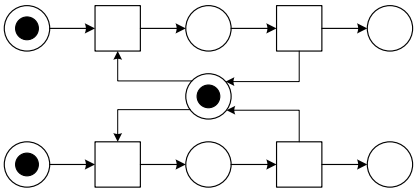

Figure 8. A-1 Initial state in a Petri net

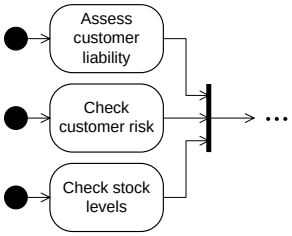

Figure 9. A-2 All start places in a UML Activity Diagram

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).

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

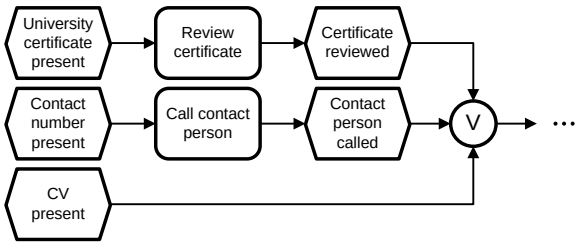

Figure 10. A-3 True conditions as an EPC

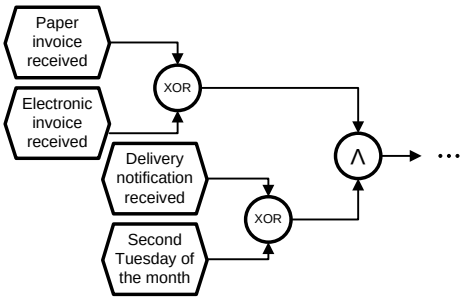

Figure 11. A-4 Occurred events as an EPC

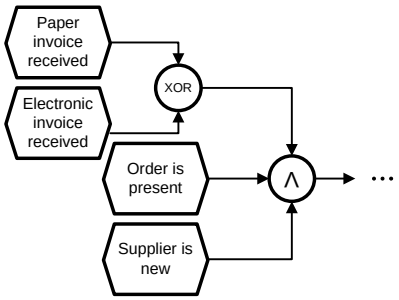

Figure 12. A-5 Occurred events plus conditions as an EPC

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". 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

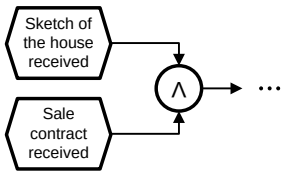

Figure 13. S-1 All subscriptions as an EPC

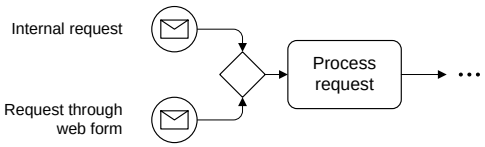

Figure 14. S-2 No subscriptions in a BPMN diagram

"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).

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

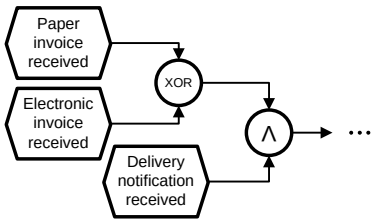

Figure 15. S-3 Reachable subscriptions as an EPC

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 U-1 Until consumption in BPEL

```
<flow>
 <sequence><receive name="rcvRFIDNotification" ..
     createInstance="yes">
   <correlations><correlation set="cont"
       initiate="join"/></correlations>
 </receive>..</sequence>
 <sequence><receive name="rcvRoutingInfo" .. createInstance="yes">
   <correlations><correlation set="cont"
       initiate="join"/></correlations>
 </receive>..</sequence>
</flow>
```

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. This is illustrated in Listing 1.

Listing 2 U-2 Until termination in BPEL

```
<flow>
 <receive name="rcvMsg1" .. createInstance="yes" />
 <sequence><receive name="rcvMsg2" .. createInstance="yes" />
  .. <exit/>
 </sequence>
</flow>
```

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

Example: A BPEL process reaches an exit activity terminating all subscriptions (Listing 2).

Listing 3 U-3 Timer-based unsubscription in BPEL

```
<flow>
 <receive name="rcvMsg1" .. createInstance="yes" />
 <pick createInstance="yes">
  <onMessage name="rcvMsg2" .. >
  .. </onMessage>
  <onAlarm name="timeout" .. >
  .. </onAlarm>
 </pick> ..
</flow>
```

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 (Listing 3).

Listing 4 U-4 Event-based unsubscription in BPEL

```
<flow>
 <receive name="rcvInvoice" .. createInstance="yes" />
 <pick createInstance="yes">
  <onMessage name="rcvPaperDeliveryNotification" .. >
  .. </onMessage>
  <onMessage name="rcvElectronicDeliveryNotification" .. >
  .. </onMessage>
 </pick> ..
</flow>
```

U-4 Event-based If one of alternative events is consumed, the others are no 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. Listing 4 shows the corresponding BPEL code.

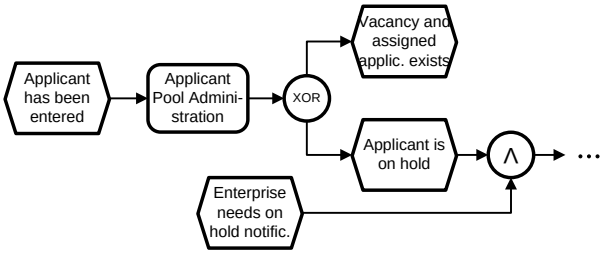

Figure 16. U-5 Proper completion as an EPC

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 contains two start nodes, “applicant has been entered” and “enterprise requires on hold notification”. An activation of the

second start node is not necessary/desired after the decision has been made that there is vacancy. This example originating from the SAP Reference Model is illustrated in Figure 16.

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 [25], 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 unsubscrip-

tion (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 [23], 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 (UML-AD)* 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 UML-AD models. That way UML-AD 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 a token upon instantiation (A-2). That way, YAWL is similar to UML-AD 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 Listings 1 to 4, 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.38]. 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 mul-

Patterns	oWFN	EPCs	UML-AD	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 (+ for supported, – for not supported, ∅ for ambiguous

tiple 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 Multiple Entry Points in Real-World Models

In Table 1 we have seen that EPCs cover the broadest range of instantiation semantics, but also with several ambiguities in subscription and unsubscription semantics. In this section we analyze how modelers in real-world projects define instantiation conditions using entry points. We utilize the SAP Reference Model [27,28] in this analysis for two reasons. First, the SAP Reference Model is publicly available for research and has been used by various researchers in their investigation of modeling practice. In this way, our findings can easily be related to that prior research. Second, and more importantly, EPCs and the SAP Reference Model are both results of a joint research project between the Institute of Information Systems (IWi) in Saarbrücken and SAP [4]. Since the definition of EPCs was directly motivated by the requirements of the SAP system, we assume that the SAP Reference Model should cover an extensive set of instantiation semantics. Against this background, this section is organized as follows. In Section 4.1 we discuss start events versus start conditions in the SAP Reference Model. Section 4.2 relates these patterns to instantiation semantics that lead to sound behavior.

4.1 Start Event versus Start Condition in the SAP Reference Model

The aim of this section is to shed light on the usage of start events and start conditions in the SAP Reference Model. We are interested in respective quantitative results for two reasons. First, we aim to gain insight into the relative importance of start events and start conditions in process mod-

to create	323	to post	102	to perform	58	to reach	31
to carry out	142	to require*	99	to release	45	to complete	28
to exist*	127	to enter	76	to plan	40	to schedule	26
to be*	124	to receive	66	to generate	39	to arise	26
to process	113	to transfer	60	to settle	34	to assign	24

Table 2
Occurrences of 20 most frequent verbs in start event labels (verbs referring to a state marked with asterisk)

els. While most scholars recommend events in EPCs to be used as real-world events, for example [5,25], a significant amount of start conditions would indicate a requirement to distinguish both on the level of the modeling language. Second, and stemming from that, such a requirement would also support our distinction of start events and start conditions in the CASU framework. For this purpose, we analyze the labels of EPC start events in the SAP Reference Model linguistically. We check which verb is used and if it is stated in passive voice in order to classify an entry point as start element or start condition. A similar approach has been used in [29] for identifying activity labeling styles of process models.

In order to make our classification as transparent as possible, we approximate the occurrence of start events and start conditions using two different methods. First, we analyze if the verb being used in the label could potentially point to a real-world event. Second, we consider the grammatical modus of the verb in the label to check whether it is likely to refer to a condition. We also did a manual classification of the start labels, but felt it was difficult to make the classification rules explicit. Since we interested in the tendency and not in the precise number of start events and start conditions, we present only the approximation here.

Table 2 lists the 20 most frequently used verbs in start labels of the SAP

Modus	Passive	To-Be	Active	Modal Verb	No Verb	Sum
Amount	943	811	347	174	52	2327
Percent	41%	35%	15%	7%	2%	100%

Table 3
Grammatical modus of verb in start event labels

Reference Model. The foundation for our first analysis builds on the insight that there are two classes of verbs: those which can be used for describing the occurrence of an event and those which refer to a state. Consider the verb *to create*. It is used, for example, in a passive voice label *demand program created* in the SAP model which indicates the occurrence of an event. In contrast to that, the verb *to exist* refers to a state. A typical example from the SAP model is *claim for support exists* where it is used to express a condition. Altogether, the latter group accounts for 396 occurrences of the five verbs *to exist, to be, to require, to need* and *to desire*. This is about 17% of all start labels.

Table 3 presents the findings from the analysis of the grammatical modus. The greatest share accounts for verbs that are used in passive voice in the start label. This is the grammatical figure that has to be expected for real-world events. Examples from the SAP Reference model are *invoice received from vendor* and *purchase order commitment created*. The second category includes those verbs that are used in a *to be* construction. Examples are *internal order is to be created* and *program structure is to be determined*. This grammatical style typically refers to a pre-condition for the subsequent activity, which could be *create internal order* and *determine program structure* for the two mentioned examples. Clearly, such a construct does not refer to an event but rather to a condition. Similar observations can be made for active voice sentences like *quotation is valid*, modal verb statements like *notification should be entered*, and labels without a verb. Therefore, only the 41% passive voice labels can be identified as real-world events using this analysis. It must be noted that several of these none passive voice labels can be regarded as so-called "trivial events". A trivial event is introduced by the modeler to meet the syntax requirements

of EPCs which demand an alternation of functions and events [25]. Several languages like YAWL and BPMN are designed to avoid such unnecessary elements.

Given these figures, we can expect a percentage between 41% and 83% of real-world start events in the SAP Reference Model and 17% to 59% start conditions. While these intervals are rather large, they clearly point to the significance of start conditions for business process modeling. The figures support a distinction of start events and conditions already on the language level. This suggestion also implies that our distinction between start condition creation patterns and start event creation patterns is required for describing the full range of instantiation semantics which are used in modeling practice.

4.2 Multiple Start Events and Correctness

In this section we discuss the issue of correctness in presence of multiple start events. We approach this question by considering the structural combination of start events in the process model. We illustrate the problem of correctness and instantiation semantics by referring back to the S-3 example process model of Figure 15. In this model there is a pair of start events s_1 and s_2 that are merged with an XOR-join followed by an AND-join from a third start event s_3. The challenge for this model is to identify instantiation semantics such that there will be no deadlock and no lack of synchronization, no matter which events occur in which order.

- Consider the case that s_3 triggers the instantiation. In order to avoid a deadlock, it is required to issue subscriptions for both s_1 and s_2. If s_1 occurs then, we have to unsubscribe the s_2 event subscription to avoid a lack of synchronization at the XOR-join.
- Consider s_1 to occur first. Now, we need only the subscription on s_3 such that the AND-join does not deadlock. Therefore, no unsubscription is required.

As can be seen, appropriate subscription and unsubscription semantics might depend on which event triggers the instantiation. In the following, we identify classes of process models for which appropriate instantiation semantics can be defined. We analyze these classes in the SAP Reference Model. Our structural analysis requires the assumption that all entry points in an EPC are start events (we have seen in the previous section that this assumption has some weaknesses). Furthermore, we assume multiple event trigger (C-5) and occurred events (A-4) semantics in order to make statements about EPCs' vague CASU part of subscription and unsubscription. Clearly, the set of classes is not complete. Still, we get insight into the relevance of subscription and unsubscription semantics in process modeling practice.

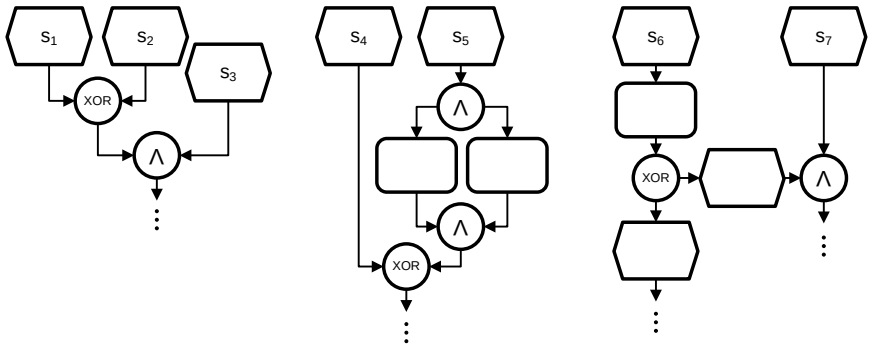

Figure 17. Start joins

The process models for which we apply our structural analysis is the class of EPCs from the SAP Reference Model that include a node that we refer to as a start join (see Figure 17). A *start join* is a join connector such that for every other node n in the EPC there is either a path from n to the start join or a path from it to n. In this way, a full order of start joins is implied. Therefore, there also exists a *minimal start join*, that is a start join for which there is no path from it to an end event such that there is another start join on the path. In Figure 17 (a) the AND is a minimal start join and in (b) the XOR. EPCs having a start join, and in particular the structure between the start events and the minimal start join, are the subject of our analysis. Furthermore, we consider start bundles. A *start bundle* is a join connector such that for two

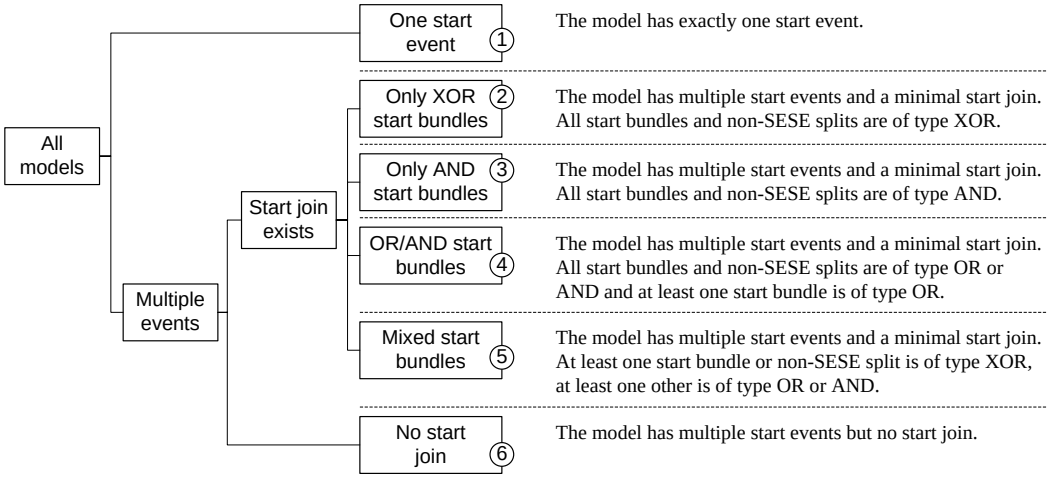

Figure 18. Structural classification of models

of its input nodes n_1 and n_2 there is no node in the EPC that has a path to both. More illustrative, there are isolated branches from at least two start events that lead to a start bundle. A join that closes a single-entry-single-exit component (SESE) in such a branch is not a start bundle such as the AND-join in Figure 17 (c). The minimal start join is a special start bundle that collects branches from all start events. Finally, we involve also those split connectors in the analysis that are not an entry to a SESE component as non-SESE splits. Based on these definitions, we partition the set of EPC models into the six categories described in Figure 18.

In the following we assume that the start join is reached exactly once per process instance. In this way, we exclude the problem of control flow correctness (deadlock and lack of synchronization) from issues that stem from instantiation semantics. In case (1) (cf. Figure 18) subscription and unsubscription are not applicable, therefore no subscription (S-2) leads to correct behavior. If all start bundles are of type XOR (2) there is also no need for a subscription (S-2). All start events are mutually exclusive. In case (3) all start events have to occur in order to provide the required tokens at the AND start bundles. This means all subscription S-1 needs to be combined with no unsubscription U-1. Once there are also OR start bundles involved (4), this approach can be considered, too. Alternatively, the reachable subscriptions could be made

(S-3) with different unsubscription strategies. In this way, less events would be needed to complete the process. Most interesting is U-5 that cancels all those subscriptions that are not needed anymore for proper completion. The latter approach can also be used for start bundles that combine all three connector types.

Pattern U-2 does not appear in the list for different reasons. An obvious realization of U-2 are explicit termination points in a process, e.g. using exit in BPEL or a terminate end event in BPMN. A construct canceling a whole process instance is not present in EPC. In the context of EPC, the assumption of U-2 semantics for start events most likely leads to modeling errors or is based on a misinterpretation of control flow constructs. The latter is the cases, when modelers assume discriminator semantics (cf. [8]): the control flow is assumed to continue as soon as one incoming branch is activated and all other activations are ignored later on. Modeling errors most likely occur as not properly handled multi-merges would be the possible result of assuming U-2 semantics. For these reasons we excluded U-2 from the considerations in this section.

Our quantitative analysis is summarized in Table 4. We only consider control flow errors that stem from flawed start event combinations. It must be noted that cases (1) to (3) are trivially correct, errors can only occur from (4) to (6) when (X)OR splits permit navigation away from a branch that runs into an AND-join that might be activated by an external event. We analyze the SAP Reference Model with its 604 EPC models. Some of these models are not coherent and we have to split them up. In this way, we get 714 models as input to our structural analysis. 265 of these models (37%) have exactly one start event. Together with the group of Only XOR (83) and Only AND start bundles (80), they are trivially correct. Together, these three groups account for 60% of all EPC models.

We also combined this structural information with the EPC soundness analysis of [14]. It is interesting to see in Table 4 that none of the five categories of EPCs

Category	Occurrence	S-patterns	U-patterns	Start Event Error	Error Rate
One start event	265	S-2	–	0	0%
Only XOR start bundles	83	S-2	–	0	0%
Only AND start bundles	80	S-1	U-1	0	0%
OR-/AND start bundles	60	S-1,S-3	U-1,U-3,U-4,U-5	0	0%
Mixed start bundles	57	S-3	U-1,U-3,U-4,U-5	0	0%
No start join	169	S-3	U-5	49	29%

Table 4

Occurrence of subscription and unsubscription patterns

with start joins had behavioral problems with start events. There are 49 in the set of 169 models without a start join that are not EPC sound. We further analyzed these models and found that all of them followed a pattern similar to Figure 16 which we used to illustrate the U-5 pattern. In these models there are AND-joins that synchronize with a start event and these AND-joins are not necessarily reached because there is an (X)OR-split upstream. Obviously, the modelers of these EPCs seem to assume U-5 instantiation semantics that would automatically unsubscribe the events.

5 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. [30,31,24].

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 UML-AD 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 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 [32]. The property of *EPC soundness* is stricter: it demands that for every start element there exists an initial marking that guarantees proper completion [24]. 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 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 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 [24], would be required. While this solution would prevent some deadlocks

at AND-joins that merge paths from start events (see Table 4), 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.

Supporting the start condition patterns (C-2 and C-3) would require either (a) the introduction of a new element or (b) the redefinition of the conditional events. Both options affect the metamodel and the current semantics, which is undesirable. However, option (b) would be less invasive. The *condition* attribute of conditional events could be reused and an additional attribute *isTriggered* could distinguish start events from start conditions. The support for activation patterns (A-1 to A-5) depends on the creation support (C-1 to C-5). By introducing start conditions A-3 and A-5 would be supported as well. Introducing the concept of start conditions would be a major semantic change for BPMN. Therefore, we do not recommend it.

As an alternative, preconditions for activities or start events could be used as substitute for start conditions. The main difference between preconditions and start conditions is that all preconditions have to be fulfilled, while start conditions might not be fulfilled, leading to non-activation of the respective branch (cf. A-3). Preconditions could be realized through data objects in BPMN. While they are underspecified in the current specification, sharpening of the semantics could easily be done. Data objects can be defined to be the required

input for activities and events. Furthermore, required states can be defined for such objects, e.g. “invoice” in state “open”. In addition to the notion of using the invoice data as input, they could indeed be interpreted as precondition for a process, meaning that process instantiation is not allowed unless there exists a corresponding open invoice. We recommend to use this strategy, which would lead to support for C-2 and C-3. However, A-3 and A-5 would still not be supported. Figure 19 illustrates the use of data objects as preconditions for instantiation.

With respect to the subscriptions (S-1 and S-3) there are three options: either (a) changing the instantiation semantics of BPMN, (b) adding a subscription attribute to pools or (c) adding a subscription attribute to start events. Options (b) and (c) seem more attractive from a consistency perspective. Regarding (b), a *keepStartEventSubscriptions* attribute could be used to specify whether S-1 semantics (value *all*) or S-2 semantics (value *no*) apply. Similar support for S-3, as it is the case for BPEL, could be achieved by using event-based gateways as start nodes. Regarding (c), there are several realization options. Using a *keepSubscription* attribute for start events, one would be able to specify S-1 (all subscriptions) and S-2 (no subscriptions). By using an attribute *subscriptionGroup* one would again be able to specify instantiation behavior as in 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. Unsubscriptions could equally be captured by additional attributes for start events, 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.

We would recommend option (b), as it requires minimal additional modeling effort and is closely aligned with BPEL semantics. All concepts for realizing S-1, S-2 and the U-patterns require correlation mechanisms such as identified

in [33]. As BPEL has a more sophisticated profile in terms of subscription and unsubscription patterns (cf. Table 1), an extension of BPMN with these aspects could simplify the automatic transformation between the languages.

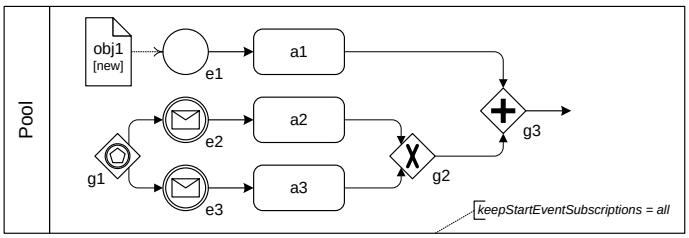

Figure 19. Recommended BPMN extensions

Figure 19 illustrates the recommended extensions. A data object *obj1* in state *new* is required for the upper start event to happen, realizing C-2. In order to realize C-3, two data objects could be specified for the same start event. The attribute *keepStartEventSubscriptions* realizes S-1, while the event-based gateway *g1* realizes U-4.

6 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 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 different workflow engines and process execution platforms. In particular, it will be interesting to compare tools of different paradigms such as activity-based approaches like in the YAWL system [34], data-driven implementations like COREPRO [35], and case-handling systems such as Flower [36].

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 & Knowledge Engineering **64** (2008) 38–54

[3] Aalst, W.: Verification of Workflow Nets. In Azéma, P., Balbo, G., eds.: Application and Theory of Petri Nets 1997. Volume 1248 of Lecture Notes in Computer Science., Springer Verlag (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. Springer (2001)

[6] Object Management Group: Unified modeling language: Superstructure. Technical Report Version 2.1.1, Object Management Group (2007)

[7] Aalst, W., Hofstede, A.: YAWL: Yet Another Workflow Language. Information Systems **30** (2005) 245–275

[8] Aalst, W., Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases **14** (2003) 5–51

[9] Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guizar, A., Kartha, N., Liu, C., Khalaf, R., Koenig, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A.: Web services business process execution language version version 2.0. Committee specification 31 january 2007, OASIS (2007)

[10] Object Management Group (OMG): Business Process Modeling Notation, V1.1. Technical report (2008)

[11] Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer-Verlag (2007)

[12] Mendling, J., Verbeek, H., Dongen, B., 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., 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. Volume 4803 of Lecture Notes in Computer Science., Springer (2007) 113–130

[14] Mendling, J.: Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness. Volume 6 of Lecture Notes in Business Information Processing. Springer (2008)

[15] Russell, N., Hofstede, A., Aalst, W., Mulyar, N.: Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22, BPMcenter.org (2006)

[16] Ouyang, C., Dumas, M., Breutel, S., ter Hofstede, A.: Translating standard process models to bpel. In Dubois, E., Pohl, K., eds.: Advanced Information Systems Engineering, 18th International Conference, CAiSE 2006, Luxembourg, Luxembourg, June 5-9, 2006, Proceedings. Volume 4001 of Lecture Notes in Computer Science., Springer (2006) 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] Aalst, W., Lassen, K.: Translating unstructured workflow processes to readable BPEL: Theory and implementation. Information and Software Technology **50** (2008) 131–159

[19] Decker, G., Mendling, J.: Instantiation semantics for process models. In Dumas, M., Reichert, M., Shan, M.C., eds.: BPM. Volume 5240 of Lecture Notes in Computer Science., Springer (2008) 164–179

[20] Kühne, T.: Matters of (meta-) modeling. Software and Systems Modeling **5** (2006) 369–385

[21] Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE **77** (1989) 541–580

[22] Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley (2001)

[23] Rump, F.: Geschäftsprozessmanagement auf der Basis ereignisgesteuerter Prozessketten - Formalisierung, Analyse und Ausführung von EPKs. Teubner Verlag (1999)

[24] Mendling, J., Aalst, W.: Formalization and Verification of EPCs with OR-Joins Based on State and Context. In Krogstie, J., Opdahl, A., Sindre, G., eds.: Proceedings of the 19th Conference on Advanced Information Systems Engineering (CAiSE 2007). Volume 4495 of Lecture Notes in Computer Science., Trondheim, Norway, Springer-Verlag (2007) 439–453

[25] Scheer, A.W., Thomas, O., Adam, O.: Process Modeling Using Event-Driven Process Chains. In: Process Aware Information Systems: Bridging People and Software Through Process Technology. Wiley Publishing (2005) 119–146

[26] Carzaniga, A., Rosenblum, D., Wolf, A.: Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems **19** (2001) 332–383

[27] Curran, T., Keller, G., Ladd, A.: SAP R/3 Business Blueprint: Understanding the Business Process Reference Model. Enterprise Resource Planning Series. Prentice Hall PTR, Upper Saddle River (1997)

[28] Keller, G., Teufel, T.: SAP(R) R/3 Process Oriented Implementation: Iterative Process Prototyping. Addison-Wesley (1998)

[29] Mendling, J., Reijers, H.: How to define activity labels for business process models? In Oberweis, A., Hesse, W., eds.: Proc. of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems (SIGSAND Europe 2008), Marburg, Germany. Lecture Notes in Informatics (2008)

[30] Eshuis, R., Wieringa, R.: Tool support for verifying uml activity diagrams. IEEE Trans. Software Eng. **30** (2004) 437–447

[31] Puhlmann, F., Weske, M.: Investigations on soundness regarding lazy activities. In Dustdar, S., Fiadeiro, J., Sheth, A., eds.: Business Process Management, 4th International Conference, BPM 2006. Volume 4102 of Lecture Notes in Computer Science., Springer-Verlag (2006) 145–160

[32] Dehnert, J., Aalst, W.: Bridging The Gap Between Business Models And Workflow Specifications. International J. Cooperative Inf. Syst. **13** (2004) 289–332

[33] Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation patterns in service-oriented architectures. (2007) 245–259

[34] van der Aalst, W.M.P., Aldred, L., Dumas, M., ter Hofstede, A.H.M.: Design and implementation of the yawl system. In Persson, A., Stirna, J., eds.: CAiSE. Volume 3084 of Lecture Notes in Computer Science., Springer (2004) 142–159

[35] Müller, D., Reichert, M., Herbst, J.: A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In Bellahsene, Z., Léonard, M., eds.: CAiSE. Volume 5074 of Lecture Notes in Computer Science., Springer (2008) 48–63

[36] Aalst, W., Weske, M., Grünbauer, D.: Case Handling: A New Paradigm for Business Process Support. Data and Knowledge Engineering **53** (2005) 129–162