Enhancing the Correctness of BPMN Models

Anacleto Correia[a,b]*, Fernando Brito e Abreu[a,c]

[a]*QUASAR, CITI, FCT/UNL, 2829-516 Caparica, Portugal*
[b]*IPS/EST, 2910-761 Setúbal, Portugal*
[c]*DCTI, ISCTE-IUL, 1649-026 Lisboa, Portugal*

Abstract

BPMN is becoming the *de facto* standard for process description, and analysis, in IT and many other business domains. BPMN supports different levels of abstraction, from high-level process models, to detailed models capable of being executed.

Several tools now support, at least partly, OMG's BPMN metamodel specification. However, while several other OMG's metamodels include a formal specification of well-formedness rules, using OCL, the BPMN metamodel specification only includes those rules in natural language, scattered across several hundred pages of that document. Not surprisingly, we found that all mainstream BPMN tools do not enforce those well-formedness rules, while checking the correctness of process models.

Model correctness enforcement is important to mitigate ambiguity. The latter hampers the achievement of a shared meaning among process stakeholders, is detrimental to process reuse and is unacceptable if we look for executable processes. To enforce model correctness we propose to supplement the OMG BPMN metamodel with well-formedness rules expressed as OCL invariants.

The verification of BPMN process models publicly available, against well-formedness rules appended to the BPMN metamodel showed that a relevant percentage of those BPMN process models fail in complying with all the well-formedness rules.

Keywords: business process modeling; BPMN; metamodel; model correctness; model checking; OCL

1. Introduction

BPMN (Business Process Modeling and Notation) (BPMN2, 2011) is one of the most recent process modeling languages, so it is grounded on the experience of earlier ones, which ontologically makes it one of the most complete process modeling languages available (J.C. Recker, Indulska, Rosemann, & Green, 2005) (J.C. Recker, Rosemann, Indulska, & Green, 2009). BPMN is also nowadays the business process notation most used among BPM practitioners (Harmon & Wolf, 2011), and the process modeling language with more modeling tools available[†]. BPMN has also transformations to other notations available, such as CSP (Wong & Gibbons, 2008) and Petri-Nets (Dijkman, Dumas, & Ouyang, 2007), which allow the use of accessible tools for formal verification.

Version 2 of the BPMN standard, is a step forward in the alignment of process modeling with OMG's initiative of Model Driven Architecture (MDA) (MDA, 2001). The BPMN language definition is based upon a metamodel built with the UML (UML, 2007a) (UML, 2007b), the standard *de facto* for software engineering modeling. Therefore, the BPMN standard formalization of the process modeling concepts and their relationships is accomplished by means of a metamodel. The specification defines different types of conformance that tool implementers can adhere to, namely regarding process modeling (elements that are part of the orchestration in a single process, as well as elements that participate in the collaboration among

* Corresponding author. Tel.: +351-21-294-8536; fax: +351-21-294-8541.
E-mail address: accorreia@campus.fct.unl.pt.

[†] See a detailed list in http://www.bpmn.org/

processes), BPMN process execution (the operational semantics support and interpretation of activity life-cycle), BPEL process execution (mapping of a BPMN model to WS-BPEL), and choreography modeling (a set of elements that puts modeling emphasis in the interaction among participants).

The BPMN standard specification can be referred, for the definition and meaning of each element, as well as for the rules about how they can be connected and for the connections meaning, but it is a too complex technical document to be suitable to normal business modelers. Besides, that standard does not provide guidance on how the modeling notation should be used to attain a comprehensible and expressive BPMN model. Moreover, a great deal of definitions and rules are only informally presented in plain English. To fulfill this gap, best modeling practices and complementary well-formedness rules for BPMN models, have been proposed by academics (Becker, Rosemann, & Von Uthmann, 2000) (Jan Mendling, Reijers, & Cardoso, 2007) (Vanderfeesten, Reijers, Mendling, van der Aalst, & Cardoso, 2008) (J. Mendling, Reijers, & van der Aalst, 2010) (Correia & Brito e Abreu, 2012) and practitioners (White & Miers, 2008) (Silver, 2009) (Allweyer, 2010).

BPMN is a semantically rich modeling language. While, for instance, a UML Activity Diagram has around 20 different modeling constructs, a BPMN process model diagram (the more complex of the 3 available ones) has around 100 different modeling constructs, including 51 event types, 8 gateway types, 7 data types, 4 types of activities, 6 activity markers, 7 task types, 4 flow types, pools, lanes, etc. If BPMN modelers are given the freedom to combine such a large plethora of modeling constructs in the absence of a powerful validation / recommendation facility embedded in the used modeling tool, inconsistent and/or even invalid models are easily produced.

A metamodel (M2) describes the abstract syntax of a language by means of meta-classes, meta-associations and cardinality constraints. When UML is adopted for expressing metamodels, Object Constraint Language (OCL) (OCL, 2006) clauses can be used in a declarative way, similar to 1st order predicate logic, to strengthen metamodel syntax and semantics, namely by imposing well-formedness rules and best practices that reduce the sources of modeling malformation.

Adding preciseness to OMG's BPMN metamodel, by using such OCL clauses, is the first objective of the work presented herein. The second objective is to validate BPMN models. The USE tool (UML based Specification Environment) (Gogolla, Buttner, & Richters, 2007) was used to embed OCL clauses on the BPMN metamodel and to instantiate it with process models.

This work contributes to enhance the correctness of produced business process models, by providing a set of static semantic rules‡ (Aaby, 1996) and best-practices design rules for business process models. Since the rules were embedded in the BPMN metamodel, business process models' correctness became intrinsically verified by the language and not ensured by rules implemented in other languages, external tools or checkers.

Some of those rules, were withdrawn from the process modeling language specification (BPMN2, 2011), scattered by the text and tables of a document with more than five hundred pages. They were expressed in the standard, in natural language yielding sometimes, a dubious interpretation. Other rules came from disseminated best-practices both from academics and practitioners. The Object Constraint Language (OCL), a declarative and predicate logic like language that supplements the UML, was used to rigorously specify and implement the mentioned rules by means of invariants. With OCL we were able to improve the static semantics of BPMN within the UML metalanguage context, the same that was used by OMG to derive the BPMN metamodel.

‡ The static semantics defines restrictions on the structure of valid texts that are hard or impossible to express in standard syntactic formalisms, i.e., exclusively through the elements and relationships of the metamodel.

The rules' empirical validation was done with 56 process models from downloaded two sources, and transformed for data analysis.

This paper is structured as follows. Section 2 provides an overview of the BPMN metamodel. Section 3 describes our metamodel-based approach that allows checking BPMN well-formedness rules upon BPMN models. Some of those rules are illustrated in section 4. In section 5 we describe the empirical validation. Results are presented in section 6. Related work is described in section 7 and finally, in section 8, some conclusions are drawn and future work is outlined.

2. BPMN Metamodel Overview

BPMN has three notations: (i) one for modeling processes' orchestration and collaboration; (ii) another called "conversation", which is a simplified version of collaboration diagrams; and (iii) a last one called "choreography" for modeling participant interactions. The full metamodel includes 151 meta-classes and 200 meta-associations.

In this paper we will only consider the first notation, the only one already existing in BPMN version 1, since it is, by far, the most well-known and used by practitioners. Next we will introduce its corresponding main concepts and connections, as described in OMG's BPMN metamodel.

The metaclass *Process* (**Figure 1**) describes a sequence of Activities carried out in an organization with some specific objective. If a process interacts with other processes, it must participate in a *Collaboration*. A collaboration groups several participants. Each *Participant* (aka Pool) must address only one process. Since a participant is an *InteractionNode*, it can send or receive *MessageFlows*.

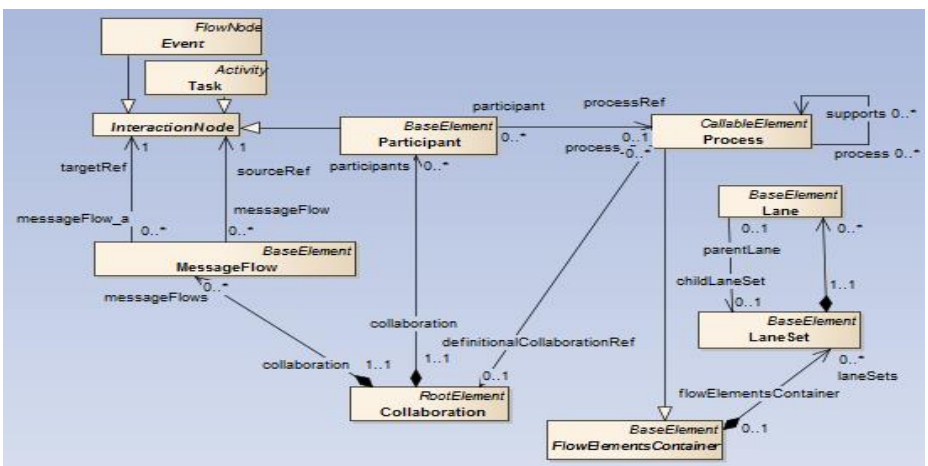

Figure 1*- Process meta-class connections*

Figure 1 depicts some of most instantiated meta-classes when a BPMN class diagram is drawn. A *FlowElementsContainer* (which can be a *Process* or a *SubProcess*) is a container of *FlowElement*. A flow element can be *FlowNode*, *SequenceFlow* or *DataObject*. A sequence flow link the various kind of flow node. The metaclass *ItemAwareElement* is the abstract class of the several kind of meta-classes, representing transient (*DataObject*), persistent (*DataStore*), input data or output data to/from *Activity* by means of subclasses of *DataAssociation*.

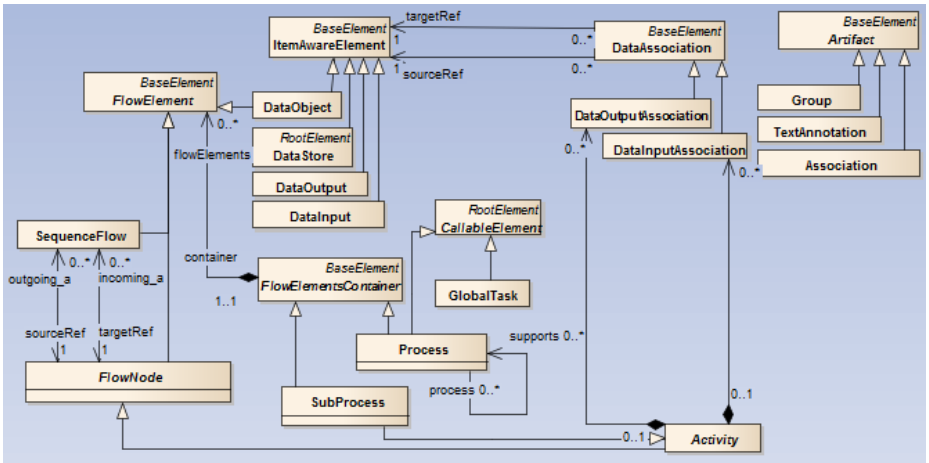

Figure 2*- Main meta-classes in a process orchestration*

While analyzing the BPMN metamodel, we found the following issues:

- In the specification it is considered a visual shortcut that uses the non-directional data association connected to a sequence flow (page 225). However, the metamodel only allows links among instances of subclasses *DataAssociation* and *Activity* (see **Figure 2**). So, a sequence flow cannot be directly linked to a *DataObject* via an instance of type *DataAssociation*. Tools that implement this visual shortcut, should instantiate the same meta-classes as the regular solution (a *DataOutputAssociation* going out an activity to a *DataObject* and *DataInputAssociation* coming from the same *DataObject* instance to other activity);
- The metamodel does not allow a *Subprocess* to receive/send a message flow. This constraint introduces a huge limitation in the modularization of a process in *Subprocesses* when there are interactions among participants. The elements that participate in an interaction must appear at top level in the process. Modelers tend to ignore this constraint, therefore violating the metamodel.

3. BPMN Syntax and Semantics

The OMG BPMN metamodel describes the abstract syntax of the BPMN language by means of meta-classes, meta-associations and cardinality constraints. We started by checking BPMN model syntax by instantiating the BPMN metamodel in the USE validation environment (Gogolla et al., 2007). The latter allows checking if a set of objects and their links match the corresponding model structural constraints, namely in what regards cardinality and type conformance.

To operationalize the aforementioned objective, our first step was to transform the BPMN metamodel definition, available in XMI format in the OMG site, in the USE concrete syntax. We did it by developing a transformation from the BPMN metamodel and BPMN process models to the USE concrete syntax, a human-readable textual format. The transformation (depicted in **Figure 3**), was attained by importing the XMI file into a CASE tool (*Enterprise Architect*) repository and then using the Java API of the CASE tool, to generate a file with the BPMN metamodel in the USE concrete syntax. At the same time, another transformation, regarding BPMN process models, was done using the *Eclipse* environment. It started by converting the XMI file into an Ecore BPMN metamodel and also by creating an Ecore USE metamodel. Using the two Ecore metamodels an ATL (Eclipse, 2011) transformation (BPMN2USE in **Figure 4**) was generated to convert BPMN process models into the equivalent USE syntax for process instances. These transformation had to match USE language conventions, thus requiring some minor changes in meta-association names such as appending as suffix an underscore plus the alphabetic character ‘a’ to identifiers which are reserved keywords in USE (e.g.,

operations, *from*), or an underscore plus an alphabetic character (a, b, or c) to the target/source to the role identifiers of associations between the same meta-classes.

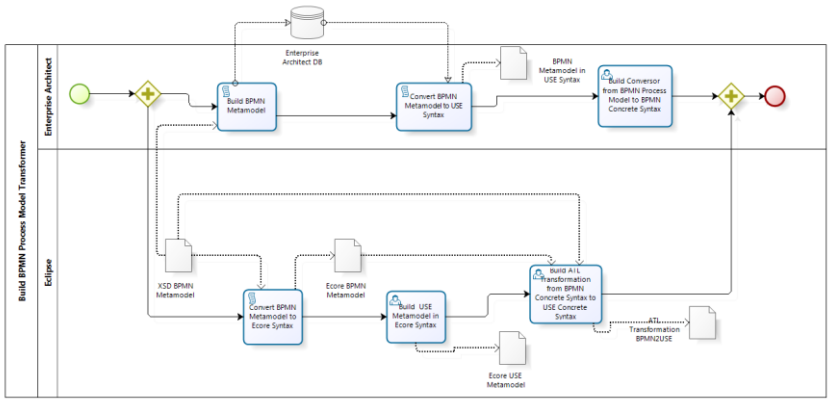

***Figure 3**- A business process model depicting (1) the transformation of the BPMN metamodel into the USE concrete syntax, and (2) the generation of BPMN2USE transformation*

After the BPMN metamodel transformation to the USE concrete syntax has been accomplished, the file with the transformed metamodel could be loaded by the USE environment. **Figure 6** shows the USE tool loaded with the 151 meta-classes and 200 meta-associations (see the "Log" window) of BPMN. The "Class diagram" window shows a cluttered snapshot of the corresponding class diagram.

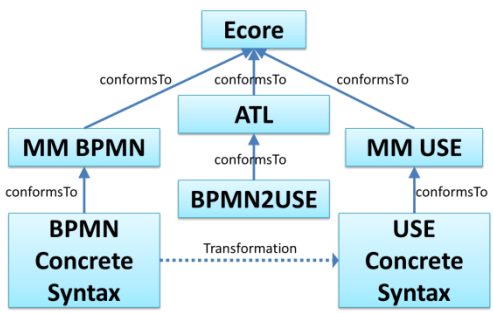

***Figure 4** - The BPMN2USE transformation*

Subsequently, BPMN models were built with the mentioned CASE tool. The depicted elements' definitions were exported to a file, and the ATL transformation BPMN2USE was used to get the instances definitions equivalent in the USE concrete syntax (see lanes *Enterprise Architect* and *Eclipse* in **Figure 5**).

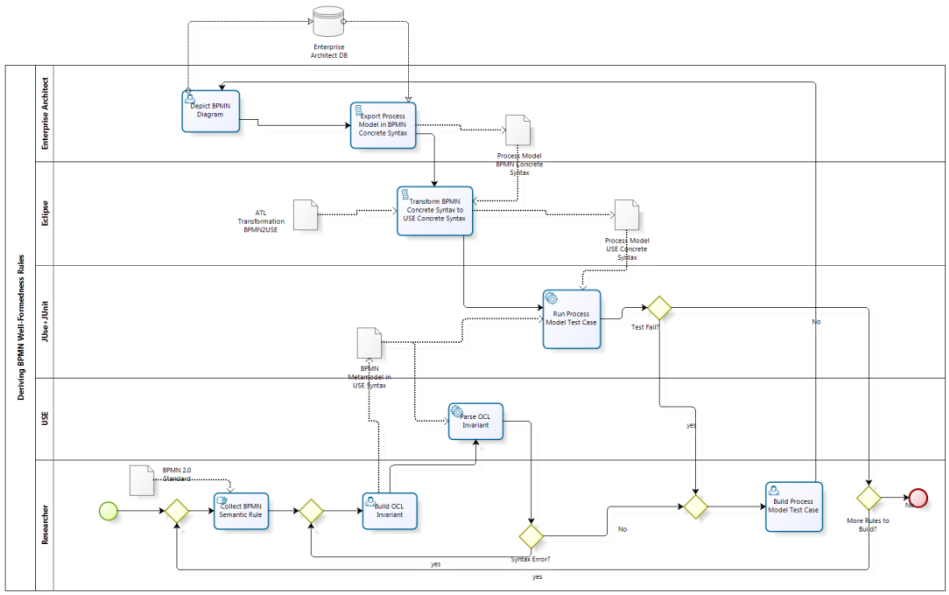

Figure 5*- A business process model depicting the building and verification of actual BPMN process models*

We were then able to instantiate the BPMN metamodel with instances corresponding to process models, as it can be ascertained in the "Object diagram" window in **Figure 6** where one can see a cluttered snapshot of the meta-object diagram corresponding to the BPMN model extract in **Figure 7**. The "Command list" window shows the commands issued to create the instances of meta-classes and meta-associations, as well as to set their state. "Object count" and "Link count" windows display the number of instances of elements and connections created, by type.

By this time, syntactical errors were already caught by the USE tool. Examples include typeless instances and connections among elements not allowed in the metamodel, such as a *DataInputAssociation* linking two instances of *Task*, an instance of *MessageFlow* linking an instance of *Gateway* to an instance of *Task*.

The next step to build an environment to validate BPMN process models was to enrich the BPMN metamodel by adding well-formedness rules as OCL invariants, corresponding to the informally conveyed rules throughout the OMG specification, complemented with best practices from the field. In **Figure 5** we depicted the business process model with all the activities taken place to enhance the BPMN with the mentioned rules. The JUSE-JUnit lane refers the role of a Java facade[§] and code generator for USE tool used for rules debugging. After each rule was codified, added to the BPMN metamodel and syntactically validated and (lanes *Researcher* and *USE* in **Figure 5**), a snipped BPMN model was generated (lanes *Enterprise Architect* and *Eclipse* in **Figure 5**) to test the correctness of the rule.

We elicited 145 invariants[**] and implemented 610 operations, resulting in a total of 755 OCL expressions (see log window in **Figure 6**), classified as follows:

- *Flow Control Well-formedness Rules*: rules related with the interaction among modeling elements;
- *Data Flow Well-formedness Rules*: rules related with sharing of data by activities;

[§] Available in http://code.google.com/p/j-use/

[**] Covering all the rules, claimed by practitioners as essential to be followed in the BPMN process modelling, such as the one at http://www.brsilver.com/2010/09/28/the-rules-of-bpmn/ (accessed in April, 16th 2012)

- *Best-Practices Recommendations*: optional rules related with advised usage of BPMN elements in diagrams.

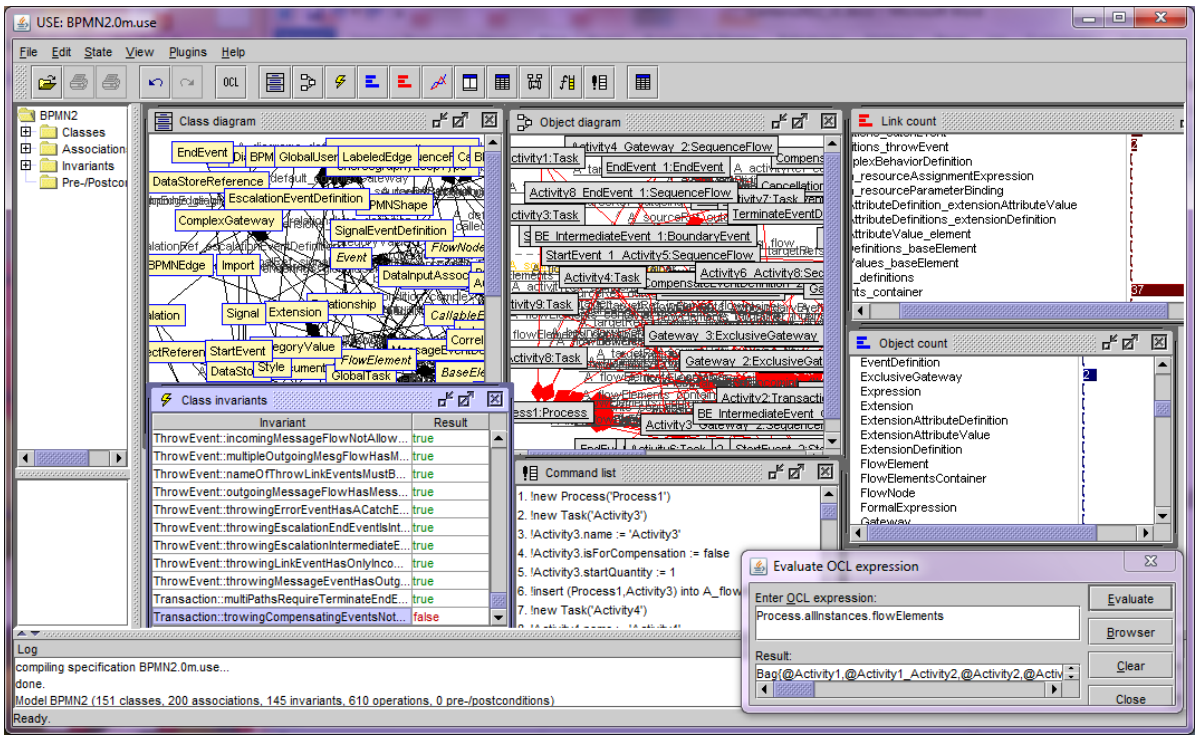

Figure 6**- The USE environment loaded with BPMN metamodel and the BPMN diagram presented in **Figure 7

The "Class invariants" window in **Figure 6** shows the results of the model check performed upon the BPMN model of **Figure 7**. One can also see that at least one well-formedness rule was broken (denoted by the Boolean value false). By querying the broken rule we can understand its semantics: a throwing compensate event is not allowed in a transitional sub-process.

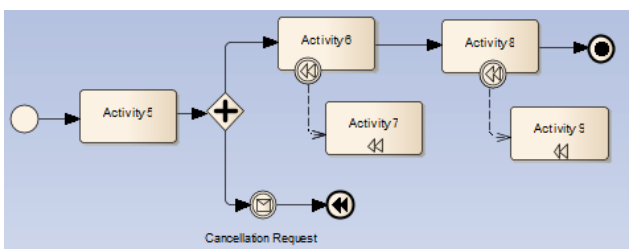

***Figure 7**- A BPMN simple diagram of a transactional sub-process*

4. Well-formedness rules

BPMN is intended for modelers with different levels of modeling expertise and technical backgrounds (business analysts or process implementers) (BPMN2, 2011). However, the available Integrated Modeling Environments (IME) for BPMN do not provide mechanisms for in depth model verification.

Any modeler would appreciate the chance of syntactically and semantically checking a produced model, according to different kinds of rules, i.e., enforced by the specification or recommended by best-practices. This need increased in the case of BPMN, given the large available number and type of constructs in the language, which allows several complete disparate correct solutions of a specific problem. The need would even become a requirement, if the process model to check became large or complex.

We present in this section, due to space restrictions, just a subset of the rules that we have defined. Each rule will be presented: (1) in textual form; (2) with model snippets illustrating its correct usage and exemplifying its violation; and (3) in a formal form using OCL syntax.

To avoid disruption in the description of these examples, we only included in this section the first order calls to OCL functions.

None of the five commercial BPMN modeling tools, which we tried for benchmarking purposes, was able to identify the violation of all these rules. This is a simple indicator that the BPMN tool market is still immature regarding well-formedness rules implementation.

4.1. A start event has no incoming sequence flows

The Start Event indicates where a particular process will start. In terms of sequence flows, the start event starts the flow of the process, and thus, should not have any incoming sequence flows (BPMN2, 2011) (page 238). Moreover, it is not allowed to have a start event without an outgoing sequence flow.

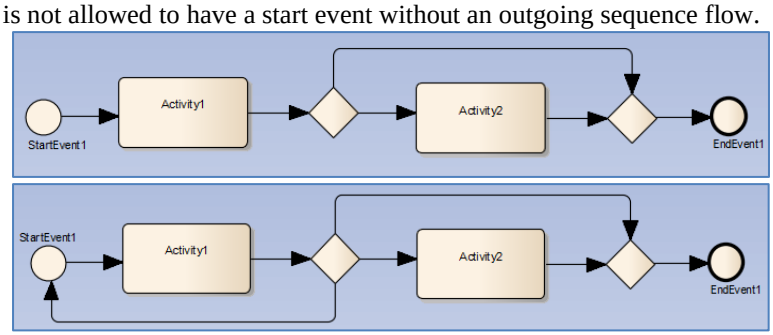

Figure 8- *Correct: Start event has no incoming sequence flows (top). Wrong: Start event has an incoming sequence flow (bottom).*

The well-formedness rule regarding start events can be enforced by attaching the following invariant to the *StartEvent* element of the BPMN 2 metamodel.

```
context StartEvent
  inv startEventCannotHaveInputSequenceFlow:
    self.inputSequenceFlows()->isEmpty() and
    self.outputSequenceFlows()->notEmpty()
```

4.2. If exists a join gateway after a parallel gateway, it must be a parallel gateway

This invariant states that for merging parallel sequence flows, originated from previous splitting with parallel gateways, a merging parallel gateway should be used.

The well-formedness rule regarding parallel gateways can be enforced by attaching the following invariant to the *Gateway* element of the BPMN 2 metamodel.

```
context Gateway
  inv mergingParalGatewayIsPrecededBySplitWithParalGateway:
    (self.isJoin() and self.oclIsTypeOf(ParallelGateway))
    implies
    precedentSplitElementIsNonExclusive()
```

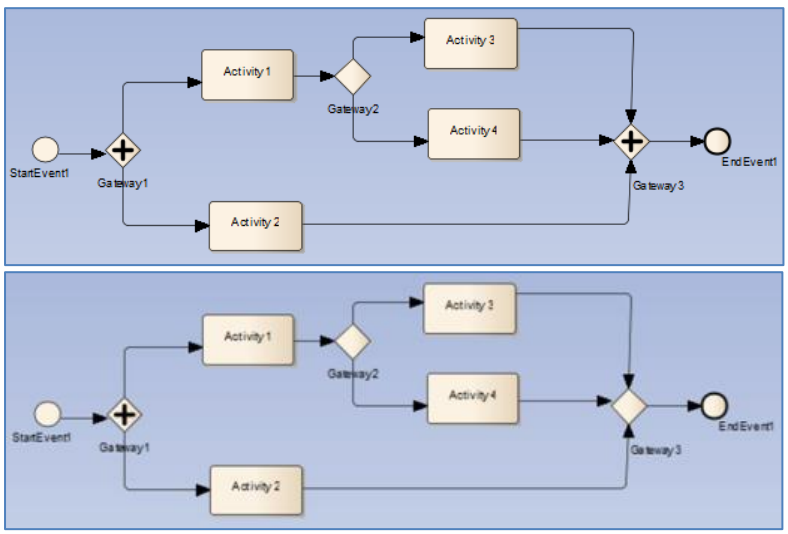

Figure 9- *Correct: A parallel gateway (Gateway3) must be used to join non-exclusive sequence flows previously split from an event based parallel gateway (Gateway1) (top). Wrong: A parallel gateway (Gateway1) precedes an exclusive gateway (Gateway3) that cannot handle non-exclusive sequence flows (bottom).*

4.3. Implicit start events require implicit end events, and vice versa

Explicit start and end events can be omitted. Implicit start (end) events require implicit end (start) events. In this case, all activities, gateways, etc. without outgoing sequence flows have implicit end events which have the same behavior as none end events.

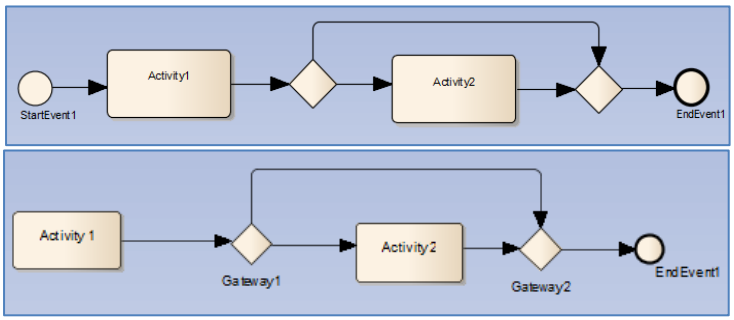

Figure 10- *Correct: Explicit start and end events (top). Wrong: Implicit start event and explicit end event (bottom).*

The corresponding well-formedness rule can be enforced by attaching the following invariant to the *FlowElementsContainer* element of the BPMN 2 metamodel.

```
context FlowElementsContainer
```

```
inv explicitStartAndEndEventsCanBeOmitted:
  (self.countAllStartEvents()=0 implies self.countAllEndEvents()=0)
  and
  (self.countAllEndEvents()=0 implies self.countAllStartEvents()=0)
```

4.4. Non-interrupting start events are only allowed in event sub-processes

When using interrupting start events in an event sub-process, the occurrence of the start event results in an interruption of the containing process. If, despite the start event occurrence, it is desirable to proceed with the containing process, we should use non-interrupting start events. However non-interrupting start events are only allowed inside an event sub-processes.

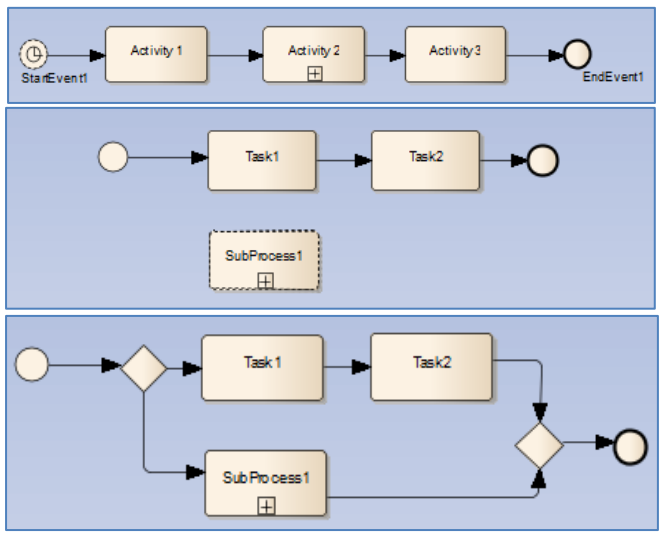

Figure 11- *Correct: Non-interrupting start event (top) only allowed inside an event sub-process (middle). Wrong: Non-interrupting start event (top) not allowed in an embedded sub-process (bottom).*

The well-formedness rule can be enforced by attaching the following invariant to the *FlowElementsContainer* element of the BPMN 2 metamodel.

```
context FlowElementsContainer
  inv nonInterruptingStartEventsHostedOnlyByEventSubProcess:
    (self.allStartEvents()
        ->select(isNonInterruptingEvent())->notEmpty())
    implies
    (self.oclIsKindOf(SubProcess)
      and self.oclAsType(SubProcess).isEventSubProcess())
```

5. Empirical Validation

An empirical study was conducted using available BPMN models stored in public repositories to determine the conformance of these BPMN models with the BPMN specification. At the same time, we tried to evaluate the effectiveness of rules for checking the correctness of these BPMN process models. The *convenience sample* used was based upon the repositories managed by the two BPMN tool providers:

BizAgi[††] - a Business Process Management (BPM) solution provider, positioned in the 2010 Gartner's BPMS Magic Quadrant (Hill, Cantara, Kerremans, & Plummer, 2009), which made available online 19 customizable templates of Business process models;

Trisotech[‡‡] - a provider of consulting services and BPM solutions, which runs an online resource repository, the Business Process Incubator, with almost 50 BPMN business process models collected from several sources.

The models in the sample were submitted to a transformation that allowed to instantiate OMG's BPMN metamodel. Then, they load into the USE environment, where the OCL evaluator returned possible rule violations resulting from the instantiation. **Figure 12** depict the activities of the business process that was took place in this study, for data collection and analysis. They are next succinctly described:

- Each of the business process models was downloaded from the respective site: (1) BizAgi models were in a proprietary format used by the tool (BizAgi Process Modeler v.2.3) of repository owner; (2) business process models from Business Process Incubator were in Visio format. These models were converted to BizAgi format since BizAgi Process Modeler can import Visio files and save them in BizAgi own format;
- After having all the files in BizAgi format, it was possible to convert them, using a functionality available in the BizAgi tool, to XPDL 2.2 format[§§], a standard from WfMC, which allows the serialization of business process models and the exchange of process definitions;
- Having business process models samples serialized into XPDL format, in order to make their verification for possible standard or best-practices violations, we needed to convert the XPDL concrete syntax to USE concrete syntax. We made that using a transformation tool. It was derived an ATL transformation (XPDL2USE) to convert the XPDL concrete syntax to USE concrete syntax (see **Figure 13**). The used metamodels were expressed using the semantics of the Ecore metametamodel. XPDL2USE transformation enables to generate a set of commands to instantiate a process model in the USE environment, and therefore conforming to the USE metamodel, from a XPDL file exported from the Bizagi Process Modeler, which serializes a BPMN process model, and conforms to the XPDL metamodel.
- The business process models now expressed in the USE concrete syntax is verified against syntactic and semantic static rules and invariants present in the BPMN metamodel read into the USE tool. Any syntactic or semantic violations, as well as the value of the metrics calculated upon the model, were outputted to a file. At the end of the models verification, the statistics were consolidated into a file that was imported by the IBM-SPSS statistical tool for data analysis.

[††] http://www.bizagi.com/

[‡‡] http://www.businessprocessincubator.com/

[§§] http://www.xpdl.org/

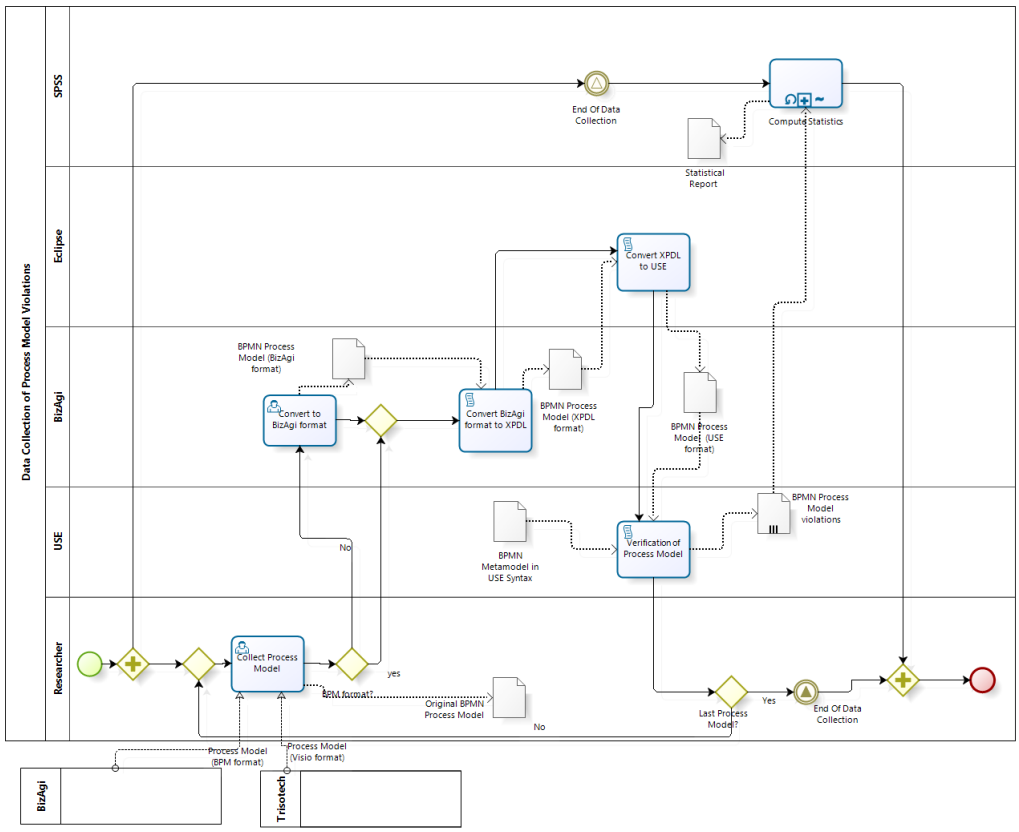

***Figure 12**- A business process model depicting the data collection of BPMN process models for empirical validation*

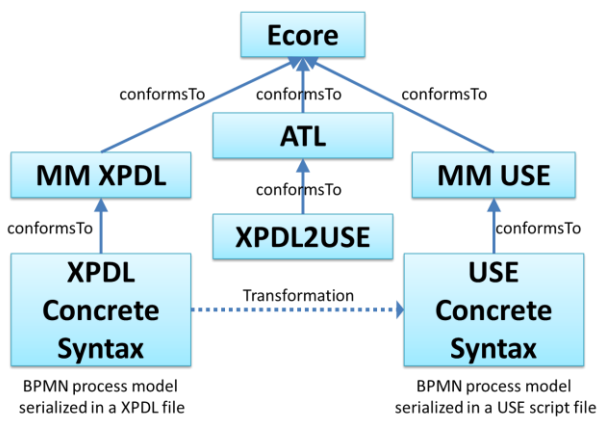

***Figure 13**- The transformation XPDL2USE*

6. Empirical Results

Table 1 summarizes the results attained by checking the business process models publicly available. As can be seen, only 53,6% of the models were in conformance with specification rules that are part of the BPMN standard. If furthermore, we had more strict requirements, by imposing the conformance with best-practices modeling rules, the percentage of models that would comply with these rules, would be drastically reduce to only 3,6%. These results underline the effectiveness and importance of OCL rules embedded in the BPMN metamodel to attain correctness in business process models.

***Table 1**- Number of Rule violations, by type, in BPMN Models*

# Errors	Standard Violation	Best-Practices Violation
0	53,6%	3,6%
1	26,8%	7,1%
2	12,5%	10,7%
3	3,6%	19,6%
4	1,8%	16,1%
5	1,8%	25,0%
6		7,1%
7		5,4%
8		3,6%
9		1,8%

We have also noticed a cumulative distribution Pareto's curve shape, regarding the modeling elements' usage in the analyzed BPMN models. Hence, most of the BPMN models (80%) only made use of a small subset (20%) of all the BPMN elements made available by the language. In **Figure 14** the shaded shape denotes the number of times a modeling element was used in the BPMN models, starting with the most used elements closest to the origin till the scarcely used, on the right side. As can be seen, the dotted vertical line representing 20% of the modeling elements intersects the cumulative curve of BPMN models in which the elements appear, near the 80%. One can conclude that even models using a small subset of elements of the BPMN specification, are highly prone to errors. So, the enforcement of well-formedness rules by BPMN tools, would probably mitigate the problem.

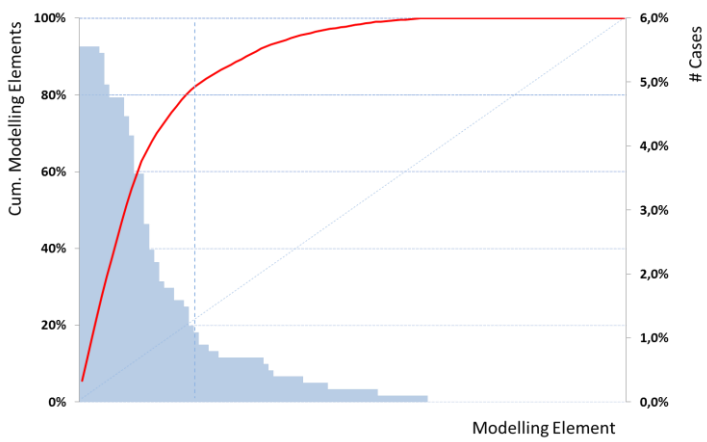

***Figure 14**- The Pareto Curve Of Modeling Elements Usage*

Furthermore, tools that implement the BPMN metamodel with embedded well-formedness rules, would contribute to a reduced learning curve of the modeling language for users, since these tools would assist the

building of business process models. However, empirical studies should also be done to corroborate this conjecture.

7. Related work

There is no general standard established for evaluating conceptual modeling (D. L. Moody, 2005), and particularly the quality characteristics, such as correctness, of business process models. However, due to the several BPMLs (Business Process Modeling Languages) available, there are some dispersed research works about modeling guidelines (Becker et al., 2000) (Jan Mendling et al., 2007) (Vanderfeesten et al., 2008) (J. Mendling et al., 2010) (Correia & Brito e Abreu, 2012), and metrics (J. Cardoso, Mendling, Neumann, & Reijers, 2006) (Jorge Cardoso, 2007) (Vanderfeesten, Cardoso, Mendling, Reijers, & Aalst, 2007) regarding the measurement of characteristics of business process models. Practitioners have also produced contributions, namely by promoting business process models modeling best-practices (White & Miers, 2008) (Silver, 2009) (Allweyer, 2010).

Meanwhile, a range of quality frameworks for conceptual modeling have been proposed in the literature, although none of them has reached a wide acceptance, thus becoming a de facto standard. For instance, the SEQUAL framework (Lindland, Sindre, & Solvberg, 1994) provides a sound theoretical basis for understanding quality in conceptual modeling. SEQUAL takes the semiotic theory (Morris, 1971) point of view, and has five components: the model, language, domain, audience participation, and perceived knowledge. Model quality is defined by relationships between the model and the other four framework components in terms of the following models' qualities: syntactic (model conformance to the language), semantic (model conformance to the domain) and pragmatic (model conformance to the audience interpretation) (J. C. Recker, 2007). The quality framework was empirically validated regarding process modeling (D. Moody, Sindre, Brasethvik, & Sølvberg, 2003). The results collected raised questions about reliability of the framework to be applied in practice in its current form. Based upon the initial approach, an enhancement regarding process modeling was proposed (Krogstie, Sindre, & Jørgensen, 2006).

Although the framework addresses quality in a systematic and comprehensive way, the drawback pointed out is to be too abstract to be used by practitioners (Shanks & Darke, 1997).

Model checking concerning business process models correctness has been also a matter of intense research. Some of the work has been done on the verification using modeling languages with formal semantics (e.g. Petri-Nets, graph-based). Due to the mathematical ground of those languages, they allow several formal verification methods, such as the verification of different classes of workflow definitions (W. van Der Aalst, 2000). However, for BPMLs, which do not have a formal semantics and allow only business processes informal representation, a different approach to verification was required. Since business process models have to be translated into a specification to be executed by a machine, a general consensus was that business process models had also to be formalized. Therefore, approaches for checking business process models for semantic errors came to light, aiming business process models mapping to languages with formal semantics, such as the checking of EPC diagrams, using transformations to Petri-Nets, proposed in (Langner, Schneider, & Wehler, 1998) (W. M. P. van der Aalst, 1999) (Dehnert & Van Der Aalst, 2004) (van Dongen, van der Aalst, & Verbeek, 2005) (J. Mendling, 2007). A modeling tool that applies graph-based rules for identifying problems in EPC business process models is provided by (Kühne, Kern, Gruhn, & Laue, 2010).

There are other approaches targeting error checking for business process models expressed in BPMN. In (Wong & Gibbons, 2008) business process models are formally analyzed using Z schemas for expressing the abstract syntax of a subset of the BPML and the CSP for expressing behavioral semantics. The approach in (E. Börger & Sörensen, 2011) uses a rule system, the Abstract State Machines (ASM) (Egon Börger & Thalheim, 2008), which can be viewed as a rigorous form of pseudo-code that follows the inheritance steps in the BPML class hierarchy.

The abstract model of the dynamic semantics of the language is attained, by inserting rules as behavioral elements at appropriate places in the class hierarchy, defining therefore, the language's execution semantics.

The enhanced BPML model can be used to check the conformance of business process models. Another approach is an ontology defined in (Natschläger, 2011) (BPMN 2.0 Ontology) that formally represents the BPMN specification. The ontology can be used as a knowledge base and as a syntax checker to validate Business process models. A common property of ontologies and the Web Ontology Language (OWL) semantics is the so-called open-world assumption (Horrocks, Patel-Schneider, & Van Harmelen, 2003), a form of partial description or under-specification as a means of abstraction, i.e., from the absence of statements, a deductive reasoner must not infer that the statement is false.

The USE validation tool has been applied to models from several domains. In (Richters & Gogolla, 2000) for instance, it was applied to the Core package of the UML 1.3 metamodel and its well-formedness rules.

We highlight the following main limitations of above mentioned proposals regarding business process models error checking:

1. Most of the verification methods rules are only applicable, after business process models are at valid state to be mapped from a specific BPML into the language and environment of the model checker.
2. The approaches presented, are technically demanding w.r.t. the formalisms, which are assumed to be known by business process models modelers (business process analysts or implementers, from whom one can hardly expect to be acquaint with above mentioned formalisms), otherwise a blended environment using a BPML tool and a transformation tool must be made available.
3. The ontological approach proposed uses an opposite approach of the closed-world assumption (Reiter, 1978) used in metamodelling through OCL invariants, which advocates that what is not currently known to be true, is false, and therefore assumes that the model has complete information to restrict arbitrary extensions of the system that could lead to inconsistencies.

So, our approach to get over the mentioned limitations was to supplement the standard process modeling language with rules that could enhance business process models. Being an integral part of the BPMN, the rules could be implemented by the same tools that already support the modeling language. Therefore modelers would have in the business process models process design, real-time notification, in addition to syntax error warnings, already available from tools, notices about static semantic violations (e.g. a throw event without a corresponding catch event; a mismatch between flows from a split parallel gateway and the incomings on the corresponding joint element, a possible cause of a deadlock situation), or even violations to the organization's established best-practices regarding business process models design. The rules appended to the metamodel could greatly enhance the quality of business process models resulting from the design phase of business process's life cycle.

8. Conclusions and future work

This paper provides a brief overview of our approach to add preciseness to OMG's BPMN metamodel specification, by formalizing as OCL invariants the well-formedness rules described informally (in natural language) within that specification. Due to space constraints, only a few rules were presented herein, along with their (incomplete) specification and some model snippets illustrating correct and incorrect situations. We have also briefly described how we have operationalized our approach by developing several transformations to allow checking rules conformance in BPMN models available from public repositories. Our metamodel-based checking facility is developed in Java and ATL. To make it more robust, we also used a JUnit test-suite where each test case checks the validity of a model snippet, such as those presented in this paper.

Using the same metamodel-based approach, we formalized a considerable set of best practices for BPMN modelers, based on published recommendations produced by BPMN experts in tutoring books.

We analyzed a considerable number of BPMN models from public repositories, and conclude for the existence of a relevant percentage of process models violating the specification rules, and even more, well accepted best-practices rules.

We plan in the future to search empirical evidence to allow corroborate (or refute) the occurrence of recurrent BPMN model malformations (aka process model anti-patterns). Along with it, we will analyze if rules violation somehow cluster. In other words, is there a high probability that breaking a modeling rule can be used as a predictor for other violations? If such is the case, then we could use that information as warnings in a recommendation system for BPMN modelers that would act by preventing modeling errors, while speeding up the learning curve.

As future work we also intend to build an open source test-driven tool that allows, for BPMN models, checking the compliance of well-formedness and best-practices rules, as well as provide the justification for each non-compliance found.

Acknowledgements

The authors would like to acknowledge CITI - PEst - OE/EEI/UI0527/2011, Centro de Informática e Tecnologias da Informação (CITI/FCT/UNL) - 2011-2012) - for the financial support for this work, and the anonymous reviewers for their valuable feedback.

References

Aaby, A. A. (1996). Introduction to Programming Languages. Retrieved 2012-01-12, from http://www.emu.edu.tr/aelci/Courses/D-318/D-318-Files/plbook/

Allweyer, T. (2010). *BPMN 2.0*. Norderstedt: Herstellung and Verlag: Books on Demand GmbH.

Becker, J., Rosemann, M., & Von Uthmann, C. (2000). Guidelines of business process modeling. *Business Process Management*, 241-262.

Börger, E., & Sörensen, O. (2011). BPMN core modeling concepts: Inheritance-based execution semantics. *Handbook of Conceptual Modeling*, 287-332.

Börger, E., & Thalheim, B. (2008). A Method for Verifiable and Validatable Business Process Modeling

Advances in Software Engineering. In E. Börger & A. Cisternino (Eds.), (Vol. 5316, pp. 59-115): Springer Berlin / Heidelberg.

BPMN2, OMG. (2011). Business Process Model and Notation (BPMN)

Cardoso, J. (2007). *Business process quality metrics: log-based complexity of workflow patterns*. Paper presented at the Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part I, Vilamoura, Portugal.

Cardoso, J., Mendling, J., Neumann, G., & Reijers, H. A. (2006). *A Discourse on Complexity of Process Models.* Paper presented at the BPM 2006 Workshops.

Correia, A., & Brito e Abreu, F. (2012, 3-5 October 2012). *Adding preciseness to BPMN Models.* Paper presented at the 4th Conference on ENTERprise Information Systems (CENTERIS'2012), Algarve, Portugal.

Dehnert, J., & Van Der Aalst, W. M. P. (2004). Bridging the gap between business models and workflow specifications. *International Journal of Cooperative Information Systems, 13*(03), 289-332.

Dijkman, R. M., Dumas, M., & Ouyang, C. (2007). Formal semantics and analysis of BPMN process models.

Eclipse. (2011, 2011-06-23). Atlas Transformation Language (ATL) v3.2.0. 2012-01-31, from http://www.eclipse.org/atl/

Gogolla, M., Buttner, F., & Richters, M. (2007). USE: A UML-based specification environment for validating UML and OCL. *Science of Computer Programming*, 69:27-34.

Harmon, P., & Wolf, C. (2011). Business Process Modeling Survey *Business Process Trends* (pp. 36): BPTrends.

Hill, J. B., Cantara, M., Kerremans, M., & Plummer, D. C. (2009). Magic quadrant for business process management suites. *Gartner Research, 164485*.

Horrocks, I., Patel-Schneider, P. F., & Van Harmelen, F. (2003). From SHIQ and RDF to OWL: The making of a web ontology language. *Web semantics: science, services and agents on the World Wide Web, 1*(1), 7-26.

Krogstie, J., Sindre, G., & Jørgensen, H. (2006). Process models representing knowledge for action: a revised quality framework. *European Journal of Information Systems, 15*(1), 91-102.

Kühne, S., Kern, H., Gruhn, V., & Laue, R. (2010). Business process modeling with continuous validation. *Journal of Software Maintenance and Evolution: Research and Practice, 22*(6-7), 547-566.

Langner, P., Schneider, C., & Wehler, J. (1998). Petri Net Based Certification of Event-Driven Process Chains

Application and Theory of Petri Nets 1998. In J. Desel & M. Silva (Eds.), (Vol. 1420, pp. 286-305): Springer Berlin / Heidelberg.

Lindland, O. I., Sindre, G., & Solvberg, A. (1994). Understanding Quality in Conceptual Modeling. *IEEE Softw., 11*(2), 42-49. doi: 10.1109/52.268955

MDA, OMG. (2001). Model Driven Architecture (MDA).

Mendling, J. (2007). *Detection and Prediction of Errors in EPC Business Process Models.* (PhD), Vienna University of Economics and Business Administration.

Mendling, J., Reijers, H., & Cardoso, J. (2007). What Makes Process Models Understandable?

Business Process Management. In G. Alonso, P. Dadam & M. Rosemann (Eds.), (Vol. 4714, pp. 48-63): Springer Berlin / Heidelberg.

Mendling, J., Reijers, H. A., & van der Aalst, W. M. P. (2010). Seven process modeling guidelines (7PMG). *Information and Software Technology, 52*(2), 127-136.

Moody, D., Sindre, G., Brasethvik, T., & Sølvberg, A. (2003). Evaluating the quality of process models: Empirical testing of a quality framework. *Conceptual Modeling—ER 2002*, 380-396.

Moody, D. L. (2005). Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. *Data & Knowledge Engineering, 55*(3), 243-276.

Morris, C. W. (1971). *Writings on the General Theory of Signs*. The Hague, The Netherlands: Mouton de Gruyter.

Natschläger, C. (2011). Towards a BPMN 2.0 Ontology. *Business Process Model and Notation*, 1-15.

OCL, OMG- Object Management Group. (2006). *OCL - Object Constraint Language Version 2.0*.

Recker, J. C. (2007). Understanding Quality in Process Modelling: towards a holistic perspective. *Australasian Journal of Information Systems, 14*(2), 43-63.

Recker, J. C., Indulska, M., Rosemann, M., & Green, P. (2005). Do process modelling techniques get better? A comparative ontological analysis of BPMN.

Recker, J. C., Rosemann, M., Indulska, M., & Green, P. (2009). Business process modeling: a comparative analysis. *Journal of the Association for Information Systems, 10*(4), 333-363.

Reiter, R. (1978). *On closed world data bases*. New York: Plenum Press.

Richters, M., & Gogolla, M. (2000). *Validating UML Models and OCL Constraints*.

Shanks, G., & Darke, P. (1997). *Quality in Conceptual Modelling: Linking Theory and Practice.* Paper presented at the PACIS 1997.

Silver, B. (2009). *BPMN Method and Style* (1st ed.). Aptos: Cody-Cassidy Press.

UML. (2007a). UML-Unified Modeling Language (OMG UML), Infrastructure, V2.1.2. In O. M. Group (Ed.): OMG-Object Management Group.

UML. (2007b). UML-Unified Modeling Language (OMG UML), Superstructure, V2.1.2. In O. M. Group (Ed.): OMG-Object Management Group.

van Der Aalst, W. (2000). Workflow verification: Finding control-flow errors using petri-net-based techniques. *Business Process Management*, 19-128.

van der Aalst, W. M. P. (1999). Formalization and verification of event-driven process chains. *Information and Software Technology, 41*(10), 639-650. doi: 10.1016/s0950-5849(99)00016-6

van Dongen, B., van der Aalst, W., & Verbeek, H. (2005). *Verification of EPCs: Using reduction rules and Petri nets.* Paper presented at the Advanced Information Systems Engineering.

Vanderfeesten, I., Cardoso, J., Mendling, J., Reijers, H. A., & Aalst, W. v. d. (2007). Quality Metrics for Business Process Models In L. Point (Ed.), *Workflow Handbook 2007* (pp. 179-190). FL, USA: Future Strategies Inc.

Vanderfeesten, I., Reijers, H., Mendling, J., van der Aalst, W., & Cardoso, J. (2008). On a Quest for Good Process Models: The Cross-Connectivity Metric

Advanced Information Systems Engineering. In Z. Bellahsène & M. Léonard (Eds.), (Vol. 5074, pp. 480-494): Springer Berlin / Heidelberg.

White, S. A., & Miers, D. (2008). *BPMN Modeling and Reference Guide: Understanding and Using BPMN*. Lighthouse Point, Florida, USA: Future Strategies, Inc.

Wong, P., & Gibbons, J. (2008). A process semantics for BPMN. *Formal Methods and Software Engineering*, 355-374.