OPBUS: Automating Structural Fault Diagnosis for Graphical Models in the Design of Business Processes

A. J. Varela-Vaca[1], R. Martínez Gasca[1], L. Parody[1]

[1]Departamento de Lenguajes y Sistemas Informáticos,
Universidad de Sevilla, Sevilla, España, 41012, España
{ajvarela, gasca, lparody}@us.es

ABSTRACT

When quality is considered in the design stages of business processes, fault diagnosis must be taken into account. In the business process modeling area there exist several proposed graphical languages. These languages have many different graphical elements that could contain structural fault modes in accordance with the corresponding standard used. It is essential to aid designers to diagnose faults within the graphical models for the business processes before they are put into execution; this is a major factor in risk avoidance. The graphical models must satisfy their associated graphical structural constraints. For organizations compliance is of major importance. Automation of the diagnosis in the design stage is necessary in order to diagnose the non-compliance to the graphical structural constraints of a business process as soon as possible. This article presents a framework with automatic diagnosing capabilities. It provides an early diagnosis of badly designed business processes. This paper describes the proposed general framework and focuses on fault diagnosis where the compliance to graphical structural constraints can be analyzed using various approaches of business processes.

1 Introduction

Business process analysis is a critical step which must be taken into account, since faults in the design of processes could result in profit losses or system failures, thereby rendering it essential to diagnose faults in the early design phase before the processes are put into the production phase.

Business process modeling remains an important factor in the areas of information systems development and business process management. The most widely known modeling techniques are: *Flowcharts*, *Petri Nets* (Salimifard, 2001), *Event-driven Process Chain* (EPC) (Aalst, 1999), *UML Activity Diagram* (Dumas, 2001), *Data Flow Diagram* (*DFD*) (Penny, 1996), *Business Process Management Notation* (BPMN) (BPMN, 2009), and *IDEF3* (Bosilj, 2001).

Once the modeling technique is selected then graphical business process specification can be defined. The designer can then subject the designed process to analysis, whereby automatic approaches to comply to the different standards and logical properties are desirable. Our proposal must be able to represent graphical business process entities that are relevant within management domains. The selection of an adequate graphical method has become an important issue for both academic researchers and business professionals, since each individual process modeling method features has its own characteristics. As a consequence, there are many research efforts dedicated to comparing and transforming these process modeling methods. In (Huang, 2008), a comparison of these major graphical process modeling methods is presented in accordance with a review of the literature review. Extensive literature research regarding another type of business process compliance has been presented (Sadiq, 2007; Namiri 2007).

In various studies, process models have been analyzed in order to prevent structural faults in the form of: *Petri Nets* (Aaslst, 2000); a set of graph reduction rules to indentify structural conflicts in process models (Sadiq, 2000); and an improved version of the latter method (Lin, 2002). Several process analysis strategies are then used to detect the syntactical, semantic and performance conflict which are discussed in (Huang, 2008), in order to guarantee executable properties.

Validation analysis methods are focused on discovering whether the designed business processes can be automatically enacted as expected. Through this analysis, any possible constraints violations should be detected. Verification analysis methods are concerned with semantic or logical conflicts, hidden in the design processes, which and could lead to unsuccessful execution. These kinds of analysis methods have been widely discussed in various reviews of the literature (Sadiq, 2000; Aalst, 1999; Marjanovic, 2000; Huang. 2008).

The automation of validation analysis is a very desirable capability from the point of view of the different stakeholders involved in the life cycle of business processes. Validation in the form of fault diagnosis provides a mechanism to identify where and what components in the business process model are failing. The main aims of this article are: (1) the formalization of key concepts of graphical elements of a business process; (2) the analysis of the different graphical elements and possible fault modes using structural constraints; (3) the design of a framework that, according to the previous items, is able to carry out the automatic validation analysis of the graphical specification of a determined instance of a business process and to report its fault modes. The contributions of this article include: automatic fault mode diagnosis using a meta-model of business process and a more efficient algorithm (linear with respect to path size) for the diagnosis.

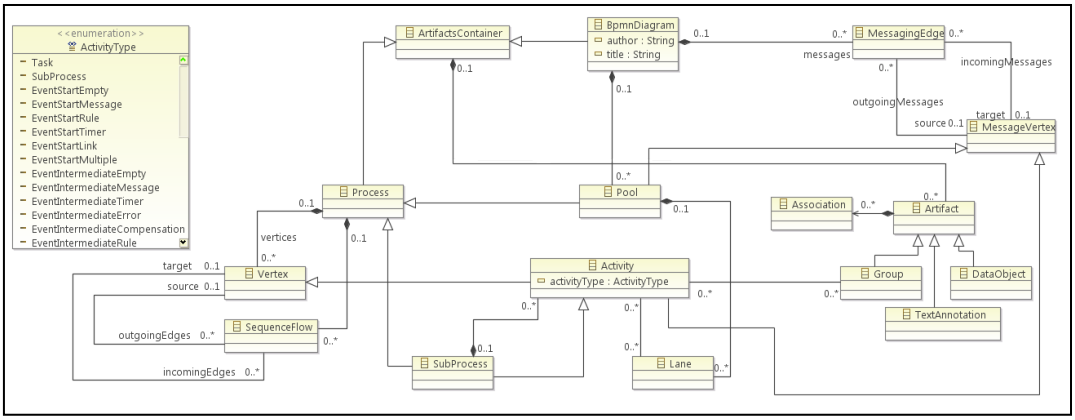

Figure 1: Meta-model for BPMN

The structure of the paper is as follows. In Section 2, the definition of the main concepts in fault diagnosis and business processes are introduced. Section 3 presents the types of structural constraints. Section 4 describes an illustrative example. Section 5 presents the framework and focuses on the validation algorithms and their implementation. As a conclusion, the paper finalised by discussing related work and future lines of research.

2 Fault Mode Diagnosis of graphical business process models

BPMN presents a graphical representation, Figure 2, for the specification of business processes in a BPM paradigm. A meta-model for BPMN is defined based on the BPMN Standard (BPMN, 2009), Figure 1.

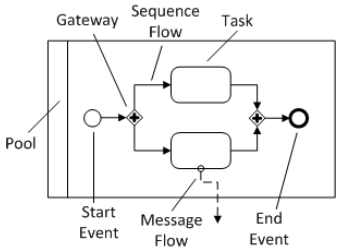

Figure 2: BPMN introduction of main elements

A meta-model is a representation which gathers the concepts and the relations between these concepts to a very high level of abstraction. This meta-model provides a finite set of structural constraints. These constraints referred to implicit constraints imposed by the construction of the meta-model. For instance, one structural constraint is: a *Sequence flow* must link two *Activities* (see in Figure 1 the association *target* and *source* from *SequenceFlow* to *Vertex*). A *Vertex* is an *Activity* and it could be a *Start Event* type, *Task* type, and so. But it is not possible to link a *Data Object* and an *Activity* with a *Sequence flow* due to the lack of any relation in the meta-model which includes this action. The following definitions have been adapted from the conflict based approach to model-based diagnosis (Hamscher, 1992).

Definition 1. Meta-model Structural Constraints. The set of the constraints which imposes the structure of the meta-model.

$$SC_m = \{C_1, C_2, C_3, \dots, C_n\}$$

Although the meta-model introduces structural constraints, there are other constraints that cannot be expressed within the meta-model due to the diagrammatic form of the meta-model and its own lack of expressivity itself. For instance, in a business process model which integrates two participants (two *Pool*), the communication between them must be carried out with a *Message* and never with a *Sequence flow*. That constraint is not implicit in the meta-model shown in Figure 1. For this reason, it is necessary to use a specific language to describe other kind of constraints. An example of these is Object Constraint Language (OCL) used to define constraints in models Unified Model Language (UML).

Definition 2. Non-Meta-Model Structural Constraints. The set of the structural constraint that are not in the meta-model.

$$SC_n = \{K_1, K_2, K_3, \dots, K_n\}$$

Definition 3. System of Structural Constraints. This is a set of constraints compounded by SC_m and SC_n.

$$SC = SC_m \cup SC_n$$

Definition 4. Business Process Model. A business process model is a tuple *(BPG, SC)* where: *BPG* is an instance of the meta-model and represents the graphical model of the business process.

For instance, in Figure 8, we can observe a specific instance (from here BPG_1) of the meta-model of Figure 1, and for example, this instance is compounded of two pools (*Tutor* and *Student*). Those are specific instances of the meta-class *Pool* of the meta-model.

Definition 5.Graphical Context. *gC* is an element of a graphical business model that contains an identification name and an finite set of associated structural constraints. This concept could be considered as a component in the model-based diagnosis.

BPG_1 is compounded of two pools (*Tutor* and *Student*) taking these as example, they represent two different graphical contexts gC_1(*Tutor*) and gC_2(*Student*).

Definition 6. Abnormal Graphical Context. A graphical context is abnormal when the $AB(gC)$ literal is hold since an associated constraint of a determined graphical context gC is not satisfied.

Definition 7. Fault Mode of Abnormal Graphical Context. This is the label associated with the constraints of a graphical context. The $AB(gC, mode_i, \ldots mode_j)$ literal represents the graphical context gC which has fault modes labeled as $(mode_i \ldots mode_j)$ since the corresponding associated constraints of these modes are not satisfied. The fault modes could also be scopes that are labels enriched with information about possible faulty graphical contexts.

For instance, in BPG_1 a *Message Flow* is used to link the activities *Inform Dept*. and *Begin Project*, a constraint is not satisfied and it produces a fault mode $mode_1$ that is related with the label: *"Message flow only link activities from different pools"*. Therefore, in gC_2(*Student*) has associated the fault modes $mode_1$ corresponding to the constraints that is not satisfied. The literal of abnormal graphical context is $AB(gC_1, mode_1)$. In case of gC_1 has n different fault modes, then $AB(gC_1, mode_1 \ldots mode_n)$.

A diagnosis specifies every fault modes of the abnormal graphical context of the graphical model and every normal graphical context. A diagnosis could be defined as follows:

Definition 8. Fault Diagnosis. Given two sets of graphical contexts gC_p and gC_n of a graphical diagram, define $D(gC_p, gC_n)$ is defined as the conjunction:

$$\bigwedge_{c \in gC_p} AB(c, mode_i \ldots mode_j) \wedge \bigwedge_{c \in gC_n} \neg AB(c)$$

3 Types of structural constraints

Before automating the fault diagnosis, we have applied a selection procedure on the structural constraints that have clear semantics. The lack of well-defined semantics of structural constraints affects their ability to achieve good automatic fault diagnosis of the graphical components. In the following section the most significant constraints to be satisfied while modeling a business process model are satisfied. The BPMN components referred to this section are described in more detail in (BPMN, 2009).

3.1 Connecting objects

Connecting objects represent different edges which could be used in the graphical model. There are three kinds of edge:

1. *Sequence Flow* defines the order of flow objects (activities, events and gateways) in a process. A sequence flow cannot cross the boundary of one pool to connect different objects of different pools. Let *SF* be a set of sequence flows which belong to a determined pool p of a business process diagram.

$$C1 \equiv \forall s \in SF \mid (p.source(s) = p.target(s))$$

2. *Message Flow* defines the order of a flow communication between two participants. In our case, participants are represented by means of pools. Therefore, a message flow can only connect two different objects from different participants, and hence it is not possible to connect an object within a single pool. Let *MF* be a set of message flows belong to a business process diagram, and p_1 and p_2 be two different pools belonging to the same business process diagram.

$$C2 \equiv \forall m \in MF\ ;\ p_1, p_2 \in P \mid (Activity.source(m) \in p_1 \wedge Activity.target(m) \in p_2 \wedge p_1 \neq p_2)$$

3. *Association Flow* is mainly used to link tasks and artifacts. The most important and useful association is between a data object and a task. This link shows that the data object is an input and output from the task. This kind of object does not affect the structure and flow of processes. Association is supported mainly between data objects and tasks, pools, sequence flows and message flows, with some restrictions. A data object is linked with a task for only one association connector at a time. Let *AF* be a set of association flows belonging to a determined business process diagram. Let *OB* be a set of all objects of the diagram minus the set of Artifacts.

$$C3 \equiv \forall a \in AF \mid \exists o \in OB(\#a.target(o) = 1 \wedge \#a.source(o) \geq 1)$$

3.2 Swim lanes

These represent major participants in a process, and typically separate different organizations. In this case, we can suppose that there is at least one participant (one pool) in a diagram and the different participants always take different names each other. We also suppose every pool contains at least one process and begins with at least one *Start Event* and finishes with at least one *End Event.* Let *P* be the set of pools of a business process diagram, and *vertices* be a property of a pool. A swim lane contains the set of objects that can belong to a business process.

$$C4 \equiv \forall p1, p2 \in P \mid (P.name(p1) \neq P.name(p2))$$
$$C5 \equiv \forall p \in P \mid (\exists s \in Start\ Event \mid (p.vertices\ (s)))$$
$$C6 \equiv \forall p \in P \mid (\exists e \in End\ Event \mid (p.vertices\ (e)))$$

3.3 Activity

A task describes the kind of work which must be carried out. *None Task Activity* is supported by our framework, although all types of activities can easily be adapted or simulated. Other special activities such as Sub-processes have not yet included in our framework, although they may easily be simulated by means of other elements already included in the editor. In our editor the None Task does not introduce any conflicts, but no Activity can be isolated. Let *A* be the set of activities which belong to a determined business process diagram, and SF the set of sequence flows belonging to the same diagram.

$$C7 \equiv \forall a \in A \mid (\exists s \in SF \mid (s.souce(a) \vee s.target(a))$$

3.4 Events

An event is something that happens during the course of a business process. These events affect the flow of the process, and they can start, delay, interrupt or end the flow of the process. There are three categories of event: (1) *Start Events* show where a process can begin; (2) *End*

Events mark where a process, o more specifically, a path (referred to as execution thread) within the process ends; (3) *Intermediate Events* indicates where something happens after a process has started and before it has ended.

For each category, various different kinds of events exist: *None Start, Message Start, Message End*, etc. A set of possible faults have been identified:

1. *Start Event* only supports outgoing sequence flows. Let *SE* be the set of Start Events which belong to a business process diagram, and *incomings* is a specific property which provides a set of sequence flows that arrive to the event.

$$C8 \equiv \forall s \in SE \mid (\#s.incomings() == 0)$$

2. *End Events* only support incoming sequence flows. Let *EE* be the set of End Events belonging to a business process diagram, and $outcomings$ is a specific property which provides a set of sequence flows that leave the event.

$$C9 \equiv \forall e \in EE \mid (\#s.outcomings() == 0)$$

3. *Start/End/Intermediate Timer Event* is triggered through a time condition, in our proposal this condition (*time*) is specified by a positive integer greater than zero. Let *STE* be the set of Start Timer Events, *ETE* the set of End Timer Events, and *ITE* the set of Intermediate Timer Events belonging to a business process diagram.

$$C10 \equiv \forall s \in STE \mid (s.time \neq null \rightarrow s.time > 0)$$
$$C11 \equiv \forall e \in ETE \mid (e.time \neq null \rightarrow e.time > 0)$$
$$C12 \equiv \forall i \in ITE \mid (i.time \neq null \rightarrow i.time > 0)$$

4. *Start/End Message Event* is triggered through a message arriving/sending to/from another pool. Let *SME* be the set of Start Message Events, *EME* the set of End Message Events, and *IME* the set of Intermediate Message Events belonging to a business process.

$$C13 \equiv \forall s \in SME \mid (\exists m \in Message \mid (m.target(s)))$$
$$C14 \equiv \forall e \in EME \mid (\exists m \in Message \mid (m.source(e)))$$
$$C15 \equiv \forall i \in IME \mid (\exists m \in Message \mid (m.target(i)))$$

5. *End/Intermediate Compensation Event*. Compensation does not just happen automatically. Another *Activity* is required to undo the work of the original *Activity* (to compensate). Therefore, this event either has to be linked to a specific activity or it can be left as general in which case it applies globally. Let *ICE* be the set of Intermediate Compensation Events, and *ECE* the set of End Compensation Events belonging to a business process diagram.

$$C16 \equiv \forall i \in ICE \mid (\exists t \in Task \mid (t.name = i.name))$$
$$C17 \equiv \forall e \in ECE \mid (\exists t \in Task \mid (t.name = e.name))$$

3.5 Gateways

Gateways are modeling elements which control how the process is executed. They can split or merge the flow of a process. There are various ways of controlling the process flow, of which the most important of them are: *(1) Exclusive Gateway,* which splits the path only for one outgoing paths depending on the evaluation of a *condition*. Acting as merge gateway, it waits for one path to finish and then continues for any incoming paths; *(2) Inclusive Gateway,* which splits the path for one or more outgoing path depending on the evaluation of a condition. Acting as a merge gateway, it waits to all incoming paths to finish; (*3) Event Gateway,* which splits the path for only one outgoing path depending on a specified Event. Acting as a merge gateway, it waits for one path to finish and continues for any incoming paths; *(4) Parallel Gateway,* which splits the path for all outgoing paths (in parallel). Acting as a merge gateway, it has to wait for all incoming paths to finish. We differentiate between a gateway for splitting and that for merging. In the case of the splitter gateway, it has only one incoming flow and two or more outgoing flows. A merge gateway works the opposite way, since it has two or more incoming flows and only one outgoing flows. Let *G* be the set of Gateways belonging to a business process diagram; *incomings* is a specific property which provides a set of sequence flows that enter the gateway, and $outcomings$ is a specific property which provides a set of sequence flows that leave the gateway.

$$C18 \equiv \forall g \in G \mid ((\#g.incomings() = 1 \rightarrow \#g.outcomings() > 1) \vee (\#g.incomings() > 1 \rightarrow \#g.outcomings() == 1))$$

Another interesting constraint for a parallel gateway is that the number of the outgoing paths has to be closed by another parallel gateway, in this case a merge gateway, with the same number of incoming paths. An example of this fault is shown in Figure 3.

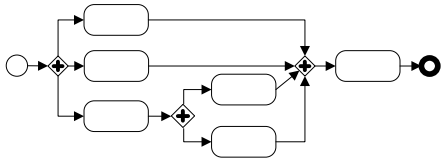

Figure 3: Example of parallels error

Let *PG* be the set of Parallel Gateways belonging to a business process diagram; *pair* is a predication that indicates if given g_1 and g_2 parallel gateways, then g_2 closes the paths (threads) opened for g_1.

$$C19 \equiv \forall g_1, g_2 \in PG \mid (pair(g_1, g_2) \wedge \#g_1.outcomings() = \#g2.incomings())$$

3.6 Non-meta-model constraints

Lack of synchronous activities produces a fault is when there is an inclusive gateway such as that which splits paths ending with an exclusive gateway, Figure 4.

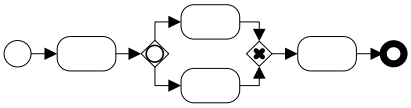

Figure 4: Example of lack in synchronization fault

Let *G* be the set of Gateways belonging to a business process diagram; *pair* is a predicate that indicates if given g_1 and g_2 parallel gateways, g_2 closes the paths (threads) that were previously opened by g_1; and *isExclusiveGateway / isInclusiveGateway* is a predicate that indicates if a particular gateway is of the type of Exclusive/Inclusive Gateway.

$$C20 \equiv \forall g_1, g_2 \in G \mid (pair(g_1, g_2) \wedge g_1.isExclusiveGateway() \wedge g_2.isInclusiveGateway())$$

Starvation (Huang, 2008) is the same case of lack synchronization but with an exclusive gateway as splitting which the path of the process and an inclusive gateway as ending of the process. In this case, the splitting gateway divides the execution in two parallel threads but the inclusive gateway is only waiting for the first thread finish to continue the execution.

$$C21 \equiv \forall g_1, g_2 \in G \mid (pair(g_1, g_2) \wedge g_2.isExclusiveGateway() \wedge g_1.isInclusiveGateway())$$

Livelock, without an appropriate end condition in the design process, can contain loop logic which would fall into an infinite state, Figure 5.

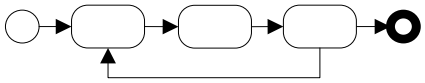

Figure 5: Example of a livelock fault

Let *cycle* be a predicate that indicates whether, given c_1 an activity, there is a cycle between c_1 and c_1.

$$C22 \equiv \forall c_1 \in Activity \mid (cycle(c_1, c_1))$$

4 Illustrative example

In this section, we present a practical example of the design of a process, Figure 8. The process shows the procedure where a Student begins to search for a final degree project until the project starts. This process involves two participants a Student and a University Tutor. The tutor's department has decided to define the process using BPMN notation. The design contains a set of deliberate structural faults.

5 OPBUS: Framework description

The proposed framework has been developed based on the main ideas of Model-Driven Development (MDD) (MDD, 2006) and Model-Driven Architecture (MDA) (MDA, 2009). Model-Driven Development is a software engineering approach where models become key elements in software development. One of the main goals of MDA is to improve the software adaptation to several different technological scenarios, thereby providing a structural separation in the architecture. The solution proposed is an architecture composed of various different levels of modeling. This separation enables the specification of models to a very high level for a particular domain but with non-specific information about the platform where the model will be deployed. MDA introduces the concept of transformation which allows one model to be obtained in one level (target model) from another model or a set of models from another level (source model). Transformations are a conflictive point because if a model is not validated in one level, then when this model is transformed into another of the different level; any faults could be dragged into the different models. For this reason it is crucial to validate the models in the development of a first step in order to prevent further faults in the subsequent steps in the framework.

5.1 OPBUS: Framework architecture

The framework has been structured (Figure 6) in several layers: Presentation, Modeling, Validation and Application.

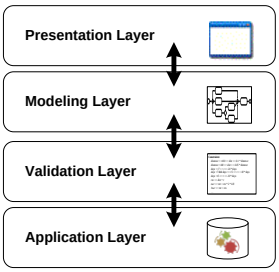

Figure 6: Framework architecture

1. The *Presentation Layer* provides the user with an integrated development environment for BPMN design. The user interface is composed of four main parts: an edition zone, a palette, properties and problem tabs, and a project workspace zone with the basic menus.

2. The *Modeling layer* integrates the power of graphically designing BPMN models together with the model-driven ideas. Thus, while modeling a BPMN in a diagram at the same time a XML-based model is constructed in background. The editor provides two different zones: the edition zone and the palette. The palette provides a graphical definition of BPMN elements which can be then selected and dropped to the edition zone. The editor automatically checks the structural constraints of the meta-model. For example, the possibility of inserting some elements inside others, whereby if this is impossible, and then the editor automatically forbids this option. This is based on the implicit structural constraints of the meta-model, and hence the model is being built while conforming to a meta-model.

3. The *Validation layer* integrates the framework with other tools that provide model validation capabilities; Epsilon framework (Epsilon, 2010). This framework is verified with a specific language for models validation, EVL. This language allows specifying constraints to be checked throughout the model in live mode. These constraints can be defined for a specific set of elements (contexts) in the model. The constraints defined in this layer correspond with the non-Meta-Model structural constraints. Thus, when the model is saved it is checked throughout the validation model. A scheme of the validation process is shown in the Figure 8.

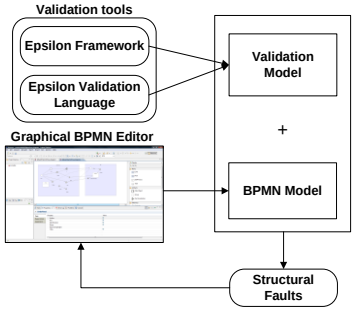

Figure 7: Validation process

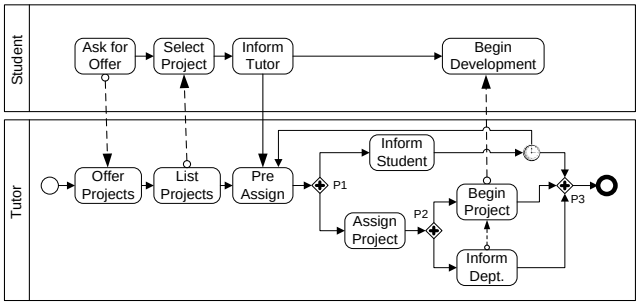

Figure 8: Example of business process with several structural faults

Constraints are composed of three main parts: (1) the '*context'* where the constraints will be applied; (2) the '*check'* part where the specific code for the constraint is specified, and where a constraint is satisfied if a *true* value is returned, *false*, in otherwise; (3) a '*message'* which, in the case where the constraint remains unsatisfied, is presented in the editor. The section message has been introduced in the definitions of *Section 2* as a label. The syntax of validation constraints in EVL is shown as follows:

```
1. // Where the constraint will be applied
2. context BPMN_Diagram {
3.    // Number of pools has to be more than 0
4.    constraint havePools{
5.      check{
6.        return self.pools.size()>0;
7.       }
8.       message: 'There is no pools defined in
9.    }                      the diagram'
10.}
```

5.2 Validation of Non-Meta-Model structural constraints

In this section, a set of Non-Meta-Model structural validation constraints are presented. The algorithms presented are associated with the context of *Pool*. A previous step to apply the algorithms is necessary to construct an auxiliary structure of the business process diagram. This structure represents an adjacency list of the business process diagram. The construction of this structure is considered as pre-processed work.

- Livelock algorithm

The main idea, subjacent in this algorithm is the detection of cycles in a model. The algorithm makes an in depth of traverse of the business process diagram, as if there is re-visited node detection. A path in reverse order is constructed from the parent of this node. If this path includes the node in question, then there is a cycle, otherwise there is no cycle. In the Figure 9, we show a trace of the algorithm applied to the example.

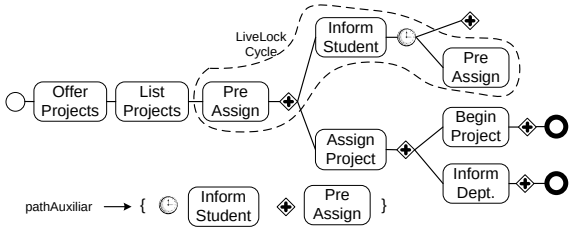

Figure 9: Livelock determination trace

```
1.constraint liveLock do
2.check do
3.  var path:=new OrderedSet;
4.  var visited, neighborings :=new List;
5.  //For each Start Event included in the pool
6.   for(s ∈ Start) do
7.    if(self.vertices.includes(s)) then
8.       // A path is build dynamically
9.       // while the graph is traversed
10.      path.add(s);
11.      i:=0;
12.      // Traversing the path
13.      while(i < #path) do
14.       //Obtain a node
15.       neigh:= path.at(i);
16.       // Obtain node's neighborings
17.       neighborings:= graph.get(neigh);
18.       visited.add(neigh);
19.       // For each new neighboring
20.        for(v ∈ neighborings) do
21.         if(visited.includes(v))then
22.            //update pathAuxiliar with
23.            // reverse path from neigh
24.            if(pathAuxiliar.includes(v))do
25.              return fault
26.            end if
27.          else
28.            path.add(v);
29.          end if
30.        end for
31.        i:=i+1
32.    end while
33.   end if
34.  end for
35. // Otherwise algorithm finishes without fault
36. end check
37.   message: "Livelock fault [Activity]"
38. end constraint
```

- Starvation algorithm

In this case the same idea of traversing of the diagram is implicit in the algorithms of Starvation and Lack of synchronization. The most significant variation between these algorithms is focused on the detection of particular gateways (inclusive or exclusive). Each gateway acting as splitter of the paths has been associated with a 'scope'. This scope saves the gateway (merger gateway) that closed the paths opened before, as shown in Figure 4. Therefore, when if we find two pairs of gateways whereby the first one is a splitter and the second one is a merger, then if the first is an exclusive gateway and the second is an inclusive gateway then Starvation has been located. In the case of an inclusive gateway as the splitter and an exclusive gateway as the merger, then Lack of synchronization is found. Figure 10 shows the trace of how the algorithms work.

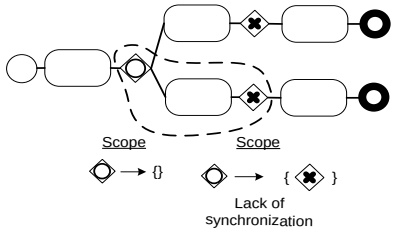

Figure 10: Lack of synchronization determination trace

```
1. constraint starvation do
2. check do
3. var path:=new OrderedSet;
4. var visited, neighborings:=new List;
5. var neigh;
6.  //For each Start Event included in the pool
7.  for(s ∈ Start) do
8.    if(self.vertexs.includes(s)) then
9.       // A path is built dynamically
10.      // while the graph is traversed
11.      path.add(s);
12.      i:=0;
13.      if(isExclusiveGateway(neigh))do
14.        //Initialize scope of neigh
15.      end if
16.      // Traversing the path
17.      while(i < #path) do
18.       //Obtain a node
19.       neigh:= path.at(i);
20.       // Obtain node's neighborings
21.       neighborings:= graph.get(neigh);
22.       visited.add(neigh);
23.       // For each new neighboring
24.       for(v ∈ neighborings) do
25.        if(isInclusiveGateway(v)))then
26.          // update pathAuxiliar with
27.          // reverse path from neigh
28.          if(pathAuxiliar.includes(neigh) and
29.               isExclusiveGateway(neigh)do
30.               return fault
31.          end if
32.         else
33.           path.add(v);
34.         end if
35.        end for
36.        i:=i+1
37.      end while
38.    end if
39. end for
40. //Otherwise the algorithm finishes without fault
41. end check
42.  message: 'Starvation fault {Gateways}'
43. end constraint
```

- Lack of synchronization in closing parallels algorithm

This algorithm is based on the same ideas as those of the starvation algorithm: the traversing of the diagram is similar but it uses the concept of scope. However another intrinsic problem appears in form of the nesting of parallels. In the case of several parallels being opened and then not closed correctly, the detection of the subjacent fault it causes is not a trivial problem. A clear example is shown in Figure 8 where it is possible to observe three faults: (1) the parallel P1 splits the execution of the business process into two threads but this parallel is matched with P3 with three incoming arrows (there should be two to match up with P1); (2) the parallel P2 splits the execution of the business process into two threads but this parallel is matched with P3 with three incoming arrow (there should be two to match up with P1); (3) the parallel P3 closed the threads opened by P1 and P2 and this is not correct way to do it (another gateway should exist to first close P2).

```
1.constraint parallelLackSynchronization do
2.check do
3.  // Same variables as other algorithms
4.  //For each Start Event included in the pool
5.  for(s ∈ Start) do
6.    if(self.vertexs.includes(s))then
7.       // Same from line 9 to 25 as Starvation
8.       // constraint, with the variation
9.       // changing if to isParallelGateway and
10.      // not necessary lines (28-30).
11.      // Different scopes are analyzed
12.      for(s1 ∈ Scope) do
13.        // Where s is a pair <Key,Value>
14.        if(isNotClosed(s.getKey())then
15.          return fault
16         end if
17.        if(#outcomings(s.getKey())<>
18.              #outcomings(s.getValue()))then
19.          return fault
20.        end if
21.        for (s2 ∈ Scope)do
22.          if(s2.getValue())
23.               includes(s1.getValue())then
24.          return fault
25.          end if
26.        end for
27.      end for
28.     end if
29.  end for
30. end check
31.   message: "Parallel fault {Gateways}"
32. end constraint
```

6 Results of validation

The example has been modeled in the editor of our framework. Structural faults are presented as shown in Figure 11. The editor shows the faults over the particular symbols and every fault messages is collected in a specific tab of the IDE called *Problems*. The fault modes identified in the design are:

- The *Student* pool has no *End Event* and *Start Event*.
- The *'Inform Tutor'* task uses a *Sequence flow* to communicate between two different pools.
- The *'Inform Department'* uses a *Message* to communicate two different tasks within the same pool.
- There is an *Intermediate Event* which indicates that time is not specified correctly.
- The first parallel (*P1*) splits the path into two threads, but the number of incoming paths, in the parallel merge (*P3*), do not match. The same occurs with the second parallel split (*P2*).
- A *Livelock* has been introduced between the *Timer Event* and '*Pre Assign'* Task.

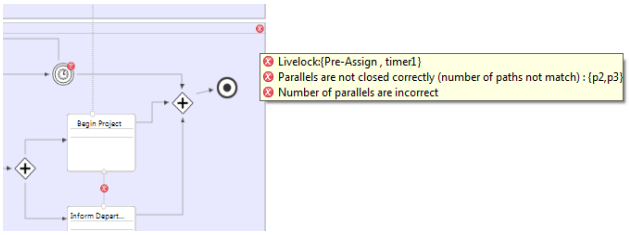

Figure 11: Faults identified in the editor

7 Related Work

In other studies a classification of anomalies is described (Kim, 2009). These anomalies are similar considered in the same way as a fault in this work. No new research contribution is made since the anomalies are already studied (Huang, 2008) and therefore a tool is proposed for EPC.

In the products of the business process arena, most application based on the design of process do not support validation in BPMN standard and in particular do not support automatic validation (in-live). A comparison of commercial products can be found in (Huang, 2008), where validation is a parameter of classification. Most validation methods for faults related to non-meta-model constraints proposed make a transformation of business processes to a graph and then carry out the analysis on this graph (Sadiq, 2000). The solutions provided in most of cases have a very high order of complexity. Other research tries to improve this complexity (Touré, 2008; Mukherjee, 2007).

Although our paper is focused on the validation of BPMN models in design time, most revised literature try to diagnose processes in run-time (Mayer, 2006; Yan, 2009; Li, 2007). (Mayer, 2006) diagnoses a composition of services in sense of the correct coordination of services involved in an orchestration. (Yan, 2009) transforms BPEL models into formal models, as automaton, to analyze them. (Li, 2007) transforms BPEL models to enriched Petri Nets notation (BPEL Petri Nets). These models are used to derivate diagnosis models that could be used to apply some diagnosis algorithm. Other works are focused on the diagnosis of BPEL models in run-time using decentralized diagnosers (Console, 2007; Ardissono, 2009).

8 Conclusion and future work

This paper focuses on the improvement of quality in the design of business processes and the need to provide a validation of business processes at the same time as the design stage. Fault diagnosis has been formalized by means of validating structural constraints. Constraints have been classified into two sets: meta-model structural constraints and non-meta-model structural constraints. Once the formalization has been carried out, it is necessary to consider the need for a tool that provides mechanisms to validate in design time. At the same time, it has to enable the visualization of structural design faults in an automatic way. For this reason, the proposed framework, OPBUS, provides an editor based on the BPMN standard and its main characteristic is the automatic validation (in-live). In our proposal for validation, we apply the constraints directly to the business process diagram and the complexity in all proposed algorithms in all cases is linear with respect to the path size.

For future work we propose an improvement in the framework with other new validation capabilities and the introduction of the validation of models in every layer of the life cycle of MDA in order to validate all different stages until correct business processes are achieved.

ACKNOWLEDGMENT

This work has been partially funded by Consejería de Innovación, Ciencia y Empresa of the Regional Government of Andalusia project under grant *P08-TIC-04095*, and by the Spanish Ministerio de Ciencia e Innovación project under grant *TIN2009-13714*, and by FEDER (under ERDF Program).

REFERENCES

(Ardissono, 2009) L. Ardissono, S. Bocconi L. Console, R. Furnari, A. Goy, G.Petrone, C. Picardi, M. Segnan, D. T. Dupré. Diagnosis of Enhancing Web Services Composition by means of Diagnosis. Business Information Processing, 2009, Volume 17, 468-479, 2009.

(Aalst, 2000) W.M.P. van der Aalst and A.H.M. ter Hofstede. Verification of Workflow Task Structures: A Petri-Net-based approach. BPM Center Report BPM-00-03, BPMcenter.org, 2000.

(Aalst, 1999) W. M. P. van der Aalst, Formalization and verification of Event-driven process chains, Information and Software Technology, Vol. 41, pp. 639-650, 1999.

(Bosilj, 2001) V. Bosilj-Vuksic, V. Hlupic, Petri Nets and IDEF: applicability and efficacy for business process modeling, Informatica 25 (1), pp.123– 133, 2001.

(BPMN, 2009) OMG Standard Business Process Model and Notation. Available: http://www.omg.org/spec/BPMN/1.2

(Console, 2007) L. Console, C. Picardi, D. T. Dupré. A framework for descentralized qualitative Model-Based Diagnosis. 20th International Joint Conference on Artificial Intelligence (IJCAI-07), Hyderabad, India, 2007.

(Dumas, 2001) M. Dumas, M., A.H.M. ter Hofstede, UML Activity Diagrams as a workflow specialization language, in: Proceedings of the 4th International Conference on the Unified Modeling Language, UML, pp. 76–90, 2001.

(Epsilon, 2010) Epsilon, Available: http://www.eclipse.org/gmt/epsilon/

(Hamscher, 1992) W. Hamscher, L. Console, and J. de Kleer. Readings in model-based diagnosis. Morgan Kaufmann Publishers Inc, 1992.

(Huang, 2008) S. Huang, Y. Chu, Shing-Han Li, D. C. Yen, Enhancing conflict detecting mechanism for Web Services composition: A business process flow model transformation approach, Information and Software Technology, Vol. 50, pp. 1069-1087, 2008.

(Kim, 2009) Gun-Woo Kim, Jeong Hwa Lee, Jin Hyun Son, Classification and Analyses of Business Process Anomalies, Communication Software and Networks, International Conference on, pp. 433-437, 2009.

(Li, 2007)Y. Li, T. Melliti, P. Dague. Modeling BPEL Web Services for diagnosis: Towards Self-Healing Web Services. 3rd International Conference on Web Information Systems and Technologies WEBIST'07, Barcelona , Spain , March 2007.

(Marjanovic, 2000) O. Marjanovic, Dynamic Verification of Temporal Constraints in Production Workflows, Agile Development Conference, Australasian Database Conference, pp. 74, 2000.

(Mayer, 2006)W. Mayer and M. Stumptner. Debugging Failures in Web Services Coordination. 17th International Workshop on Principles of Diagnosis (DX'06), 2006.

(MDA, 2009) Model-Driven Architecture, Available: http://www.omg.org/mda/

(MDD, 2006) Model-Driven Software Development, T. Stahl, M. Völter, Ed. Wiley, ISBN 0-470-03570-0, 2006.

(Mukherjee, 2007) A. Mukherjee, Anup Kumar Sen and Amitava Bagchi. The representation, analysis and verification of business processes: a metagraph-based approach. Information Technology and Management, 2007.

(Penny, 1996) A.K. Penny, Introduction to systems analysis & design: a structure approach, IRWIN, 1996.

(Namiri, 2007) K. Namiri, N. Stojanovic Using Control Patterns in Business Processes Compliance, Web Information Systems Engineering – WISE 2007 Workshops, pp. 178-190, 2007.

(Sadiq, 2000) Sadiq, W. and Orlowska, M. E. Analyzing process models using graph reduction techniques. Inf. Syst. 25, 2, pp.117-134, 2000.

(Sadiq, 2007) S. Sadiq, G. Governatori, K. Namiri, Modeling Control Objectives for Business Process, Compliance, Business Process Management, pp. 149-164, 2007.

(Salimifard, 2001) K. Salimifard, M. Wright, Petri net-based modeling of workflow systems: an overview, European Journal of Operational Research, 134, pp. 664–676, 2001.

(Touré, 2008) F. Touré , K. Baïna and K. Benali, An Efficient Algorithm for Workflow Graph Structural Verification. Lecture Notes in Computer Science Springer Volume 5331, 2008.

(Yan, 2009) Y. Yan, P. Dague, Y. Pencole, M-O Cordier. A Model-Based approach for diagnosising faults in Web Service Processes. International Journal of Web Services Research JWSR, 2009.