Quantifying the Parallelism in BPMN Processes using Model Checking

Radu Mateescu, Gwen Salaün, Lina Ye

Quantifying the Parallelism in BPMN Processes using Model Checking

Radu Mateescu
Inria, University of Grenoble Alpes, LIG, CNRS, France
Radu.Mateescu@inria.fr

Gwen Salaün
University of Grenoble Alpes, Inria, LIG, CNRS, France
Gwen.Salaun@imag.fr

Lina Ye
Inria, University of Grenoble Alpes, LIG, CNRS, France
Lina.Ye@inria.fr

ABSTRACT

A business process is a set of structured, related activities that aims at fulfilling a specific organizational goal for a customer or market. An important metric when developing a business process is its degree of parallelism, *i.e.*, the maximum number of tasks that are executable in parallel in that process. The degree of parallelism determines the peak demand on tasks, providing a valuable guide for the problem of resource allocation in business processes. In this paper, we investigate how to automatically measure the degree of parallelism for business processes, described using the BPMN standard notation. We first present a formal model for BPMN processes in terms of Labelled Transition Systems, which are obtained through process algebra encodings. We then propose an approach for automatically computing the degree of parallelism by using model checking techniques and dichotomic search. We implemented a tool for automating this check and we applied it successfully to more than one hundred BPMN processes.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Model checking*; D.2.8 [**Software Engineering**]: Metrics—*Process metrics*

General Terms

Design, Languages, Verification

Keywords

BPMN Process, Degree of Parallelism, Process Algebra, Labelled Transition System, Model Checking

1. INTRODUCTION

A business process is a set of structured, related activities or tasks designed to produce a specific output for a customer or market. It defines a specific ordering of activities across time and space, with a beginning and an end [5]. Business process modelling is an important area in software engineering. The idea is to represent processes so that they can be analyzed in order to improve process efficiency and quality. More precisely, it is a stage to model activities, their causal and temporal relationships, and specific business rules that process executions have to comply with.

An important metric when modelling and developing a business process is its degree of parallelism, which is the maximum number of tasks that are executable in parallel in the process. Parallel job processing is known as a solution for improving the efficiency of business processes [5]. Hence, when the performance of a process is an important criterion, one may seek to augment the parallelism degree as much as possible. However, the number of parallel branches in a process also complicates process design, development, and maintenance. This means that when performance is not crucial, one may prefer to reduce the parallelism degree. Note that these design tradeoffs between high performance and simplified development, and the refactorings they impose on the corresponding process, can be both guided by the degree of parallelism.

Furthermore, the degree of parallelism, which determines the peak demand on tasks, provides a valuable guide for the problem of resource allocation in business processes [25]. Examples of such resources include physical objects, time, budget as well as human beings. A typical example of resource allocation constraint is that a same agent cannot be assigned to parallel branches of the same workflow and execute different tasks simultaneously.

In this paper, we study how to compute the degree of parallelism, simply also called degree in the sequel, for business processes modelled in Business Process Modelling Notation 2.0 (BPMN for short in the rest of the paper), which is now published as ISO/IEC standard [13]. This is not a trivial task because a BPMN process may be large and may involve intricate structures, such as infinite loops or nested gateways, which impede its degree computation.

In this context, we propose a new framework to automatically compute this degree. We first present a formal model in terms of Labelled Transition Systems (LTSs) for BPMN processes. This model is obtained by encoding a BPMN process into process algebra and then using classical enumerative exploration techniques for generating the corresponding LTS. Such a low-level formal model makes much easier the definition of formal analysis techniques and algorithms, what would have been more complicated if defined directly at the BPMN level. We describe how to extend

.

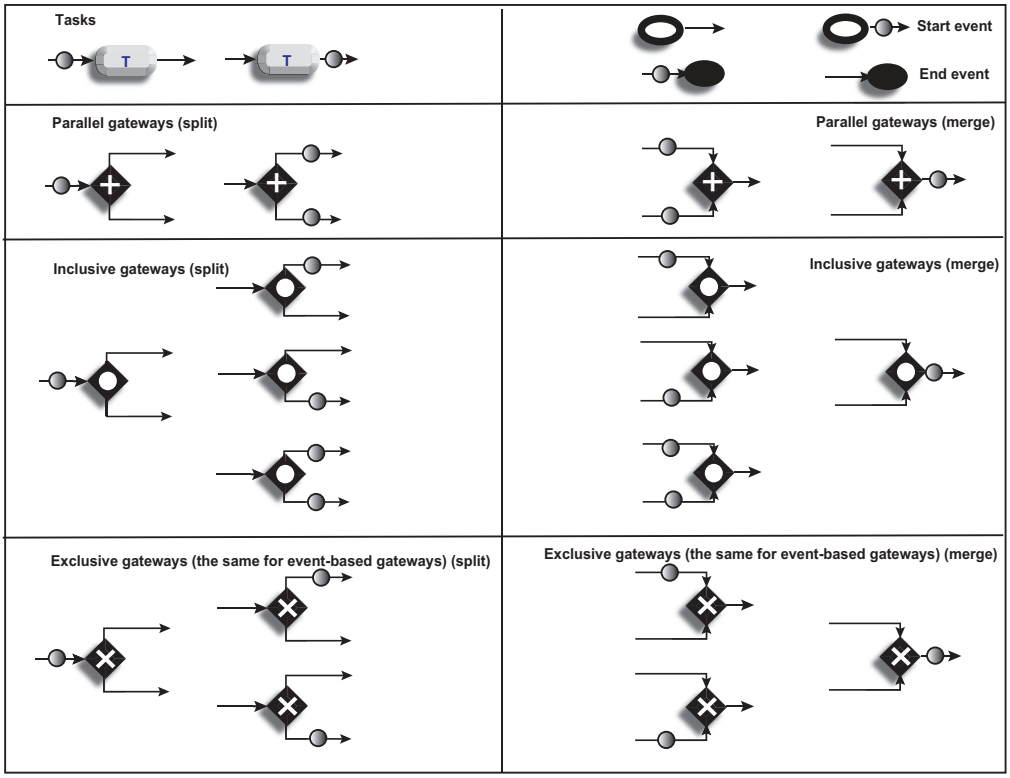

Figure 1: Execution semantics for BPMN elements.

such LTSs with special transitions whose labels contain the information about the number of parallel branches present in the initial BPMN process. These extended LTSs are used in a next step for computing the degree of parallelism. To do so, we propose an algorithm that encodes the degree computation as a model checking problem and applies this verification following a dichotomic search until obtaining the final result. We chose model checking because it is a well-established technique, supported by powerful tools.

To evaluate our approach, we implemented and applied it on more than one hundred BPMN processes that contain combinations of different BPMN gateways as well as cycles. Let us note that even if the degree measurement is illustrated with BPMN, the approach can be applied to other kinds of notations that can be expressed by control flows with parallel tasks, such as UML activity diagrams [14] or YAWL workflows [26].

To sum up, the four major contributions of this work with respect to existing results are as follows:

- We present an (extended) LTS formal model obtained via process algebra encodings for a representative subset of BPMN processes including typical gateways.
- We propose an approach relying on model checking techniques and dichotomic search to measure the degree of parallelism for BPMN processes.
- We implemented our approach as a tool that efficiently automates the degree computation.
- We applied our tool on more than one hundred BPMN processes, in particular on some real-world ones borrowed from the literature on the subject.

The rest of the paper is organized as follows. Section 2 briefly introduces BPMN and formally defines the degree of parallelism for BPMN processes. In Section 3, we present formal models for BPMN using process algebra and LTSs. Section 4 shows how to compute the degree of parallelism on such LTSs. We introduce tool support and experimental results in Section 5. Section 6 reviews related work and Section 7 concludes the paper.

2. DEGREE OF PARALLELISM FOR BPMN PROCESSES

BPMN [13] is a graphical notation for modeling business processes, which can be defined as a collection of related tasks that produce specific services or products for particular clients. BPMN is now an ISO/IEC standard [13]. Its semantics is only described informally in official documents [20, 13], but some attempts have been made for giving a formal semantics to BPMN, see *e.g.*, [8, 27, 22].

2.1 BPMN Processes

In this paper, we focus on the BPMN elements related to control-flow modelling that have an impact on the degree of parallelism [25], and we do not take data objects into account. Hence, three types of nodes named *event*, *task*, and *gateway*, are considered, as well as one type of edges called *sequence flow*. More precisely, we consider start and end

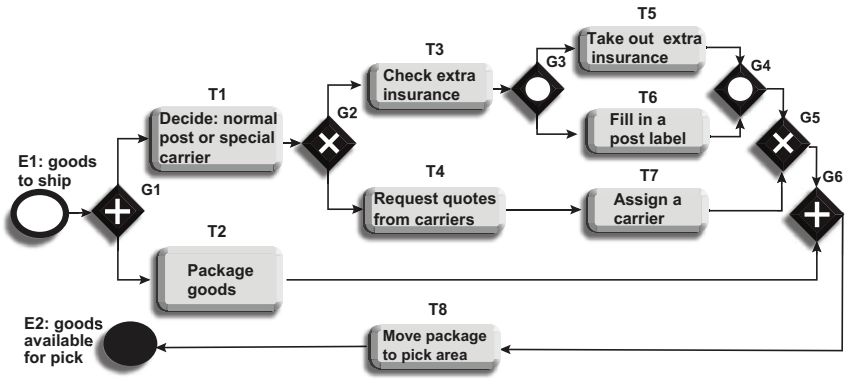

Figure 2: An example of BPMN process: Shipment Process of a Hardware Retailer.

events that are used to initialize and to terminate a process, respectively. A task represents an atomic activity that has exactly one incoming and one outgoing branch. A gateway is used to control the divergence and convergence of execution flow. A sequence flow describes two nodes executed one after the other, *i.e.*, imposing the execution order. We denote $n \rightarrowtriangle m$ when there is a sequence flow from node n to m in a BPMN process.

Gateways are crucial since they are used to model control flow branching in BPMN and therefore influence the parallelism degree. There are five types of gateways in BPMN: *exclusive, inclusive, parallel, event-based* and *complex gateways*. We take into account all of them except complex gateways, which are used to model complex synchronization behaviors especially based on data control.

For each type of gateways, if it has one incoming branch and multiple outgoing branches, we call it a *split*, *e.g.,* split inclusive gateway. Otherwise, if it has one outgoing branch and multiple incoming branches, we call it a *merge*, *e.g.,* merge inclusive gateway. An exclusive gateway chooses one out of a set of mutually exclusive alternative incoming or outgoing branches. For an inclusive gateway, any number of branches among all its incoming or outgoing branches may be taken. A parallel gateway creates concurrent flows for all its outgoing branches or synchronizes concurrent flows for all its incoming branches. For an event-based gateway, it takes one of its outgoing branches or accepts one of its incoming branches based on events. From the parallelism point of view, it can be handled in the same way as an exclusive gateway. In the following, we call the branches that can be taken by a gateway during an execution as *active* branches.

The execution semantics of BPMN elements under our consideration is depicted in Figure 1, where the execution order is from left to right. If a token is on one sequence flow, then the destination node for this sequence flow is ready to be triggered. Besides this, there is no time constraint on when it should start. Particularly, the start event node can be triggered at any moment, which creates a token on its outgoing sequence flow. As for an end event node, whenever a token arrives on its incoming sequence flow, this process can be terminated by firing it.

Now we present and explain the following BPMN process, which serves as an illustrative example in one of the standard documents for BPMN [21].

Example 1. A BPMN process called Shipment Process of a Hardware Retailer is shown in Figure 2. It begins from the start event, *i.e.,* goods are ready to ship ($E1$), and immediately splits into two parallel branches with the parallel gateway $G1$. Thus, two tasks after $G1$ are executable in parallel: deciding normal post or special carrier ($T1$) and packaging goods ($T2$). The split exclusive gateway $G2$ after $T1$ means that only one of its outgoing branches is taken, *i.e.,* either normal post or special carrier is selected. In the former case, the task of checking if extra insurance is necessary ($T3$) is carried out. In the latter case, requesting quotes from carriers ($T4$) is executed. After completing $T3$, two tasks are independently executable due to the split inclusive gateway $G3$: taking out extra insurance ($T5$) and filling in a post label ($T6$). The corresponding merge inclusive gateway for $T5$ and $T6$ is $G4$. The merge exclusive gateway $G5$ has two incoming branches: either the completion of assigning a carrier ($T7$) or the outgoing flow of $G4$. To execute the final task that moves package to pick area ($T8$), one has to wait for both incoming branches of the merge parallel gateway $G6$, *i.e.,* the completion of $T2$ and the branch outgoing from $G5$.

2.2 Parallelism Definition for BPMN Processes

In our BPMN processes, gateways are the only constructs that can change the number of parallel branches, and thus have an impact on the degree of parallelism. To facilitate the computation of this degree, we define the parallel impact of the different gateways on BPMN processes. Precisely, if the number of parallel branches is increased (decreased, resp.) by m after a gateway n_g, then the parallel impact of n_g is m ($-m$, resp.). Particularly, an exclusive (event-based, resp.) gateway does not influence the parallelism since it only takes one of the incoming or outgoing branches. Hence, its parallel impact is 0. We formally define such parallel impacts as follows.

Definition 1. (Parallel Impact of Gateways) Given a gateway n_g, we denote its impact on the maximum number of parallel branches by $\Lambda(n_g)$, which is defined for each gateway as follows:

- if n_g is a split parallel (inclusive, resp.) gateway with $m+1$ outgoing branches, then $\Lambda(n_g) = m$;
- if n_g is a merge parallel (inclusive, resp.) gateway with $m+1$ incoming branches, then $\Lambda(n_g) = -m$;

- if n_g is a split or merge exclusive (event-based, resp.) gateway, then $\Lambda(n_g) = 0$.

Given a BPMN process B, we call an *enactment* of B a set of nodes involved in one actual execution, whose ordering should respect the execution semantics defined by B. For example, $p = E1.G1.T1.T2.G2.T4.T7.G5.G6.T8.E2$ is one enactment for Example 1 but $p\prime = E1.G1.T1.T2.G2.T3.G3.T5.G4.T4.T7.G5.G6.T8.E2$ is not since $T3.G3.T5.G4$ and $T4.T7$ are exclusive alternatives due to exclusive gateway $G2$. We denote the set of all enactments for a BPMN process B by $\Phi(B)$, simply Φ if there is no ambiguity. Note that any non-empty prefix of an enactment is also considered as an enactment.

Definition 2. (Degree of parallelism for enactments) Given an enactment $e \in \Phi$ of a BPMN process, let $\{n_g^1, ..., n_g^n\}$ be the set of all gateways involved in e. The degree of parallelism for e is $D(e) = 1 + \sum_{i=1}^{n} \Lambda(n_g^i)$.

The degree of parallelism for an enactment denotes the maximum number of active parallel branches at the end of this enactment.

Definition 3. (Degree of parallelism for BPMN processes) Given a BPMN process B and its set of all enactments Φ, the degree of parallelism for B is the maximum degree of its enactments, *i.e.,* $D(B) = Max_{e \in \Phi} D(e)$.

Given a BPMN process B, if $\forall k > 0$, we have $D(B) > k$, then B has an infinite degree of parallelism. Since BPMN processes have a finite number of nodes, this situation can be caused only by a cycle containing more added parallel branches than reduced ones. In a BPMN process, if we have a sequence $n_0 \rightarrowtriangle n_1 ... \rightarrowtriangle n_m$ such that $n_0 = n_m$, then we say that it is a process cycle. Only such cycles can generate infinite enactments considering that a BPMN process has a finite set of nodes.

Definition 4. (Parallel Augmented Cycle). Given a cycle ϕ in a BPMN process, $\phi = n_0 \rightarrowtriangle n_1 \rightarrowtriangle ...n_m \rightarrowtriangle n_0$, where $n_i, i \in \{0, ..., m\}$ are BPMN nodes, let $\{n_g^1, ..., n_g^k\}$ be the set of all gateways involved in ϕ. Then ϕ is called a Parallel Augmented Cycle (PAC) if $\sum_{i=1}^{k} \Lambda(n_g^i) > 0$.

LEMMA 1. *Given a BPMN process, its degree is infinite iff it contains a PAC.*

PROOF. Suppose that the degree of a BPMN process B is infinite and that it does not contain a PAC. Non existence of PAC implies that if B has cycles, for each cycle ϕ, we have $\sum_{i=1}^{n} \Lambda(n_g^i) \leq 0$, where n_g^i is a gateway in ϕ. Any possible infinite enactments in B can only come from such cycles, *i.e.*, ϕ executes infinitely. Since the impact of ϕ on the parallelism is not positive, it follows that $\exists k \in N, k \geq D(B)$, which contradicts the assumption of infinite degree for B. Now suppose that the degree of B is finite and it contains a PAC. This means that $\exists k \in N, k \geq D(B)$ and there exists a cycle for which we have $\sum_{i=1}^{n} \Lambda(n_g^i) > 0$, where n_g^i is a gateway in ϕ. From the latter, we know that for each $k \in N$, we can always find a number p such that the cycle is executed p times, leading to an enactment e with $D(e) > k$, which contradicts the assumption that $D(B)$ is finite. □

Our approach for computing the degree of parallelism described in the following sections only concerns the BPMN processes with finite degree, *i.e.,* processes without PAC. The presence of PAC in BPMN processes can be detected by static analysis, as it is done in our tool chain described in Section 5.

3. FORMAL MODELS FOR BPMN

In this section, we show how to transform BPMN processes into LTS models. This is achieved through process algebra encodings and enumerative state space exploration. These models are equipped with model checking features, which facilitate the computation of parallelism degree.

3.1 Process Algebra Encoding

Process algebras are used to formally model concurrent systems, which include a family of related approaches, *e.g.,* CSP, CCS, or ACP. We chose LNT [2] for three reasons. First, it is a value-passing process algebra that provides expressive operators for translating rather complex BPMN constructs, *e.g.,* nested gateways can be modeled by nested choice and/or parallel compositions. Second, a translation between two high-level languages is much simpler since both share similar operators, *e.g.*, sequence, choice, or interleaving. Third, LNT is supported by CADP, a state-of-the-art verification toolbox [10], that can be used to compile the LNT specification into an LTS, enumerating all the possible behaviors. Furthermore, CADP provides various analysis techniques and tools such as model checking, compositional verification, or performance evaluation.

We suppose that BPMN processes are syntactically correct. This can be achieved using existing tools (*e.g.,* the Bonita studio [7]). Now we briefly describe the principles for translating BPMN to LNT (more details are available in [11, 22]). For each BPMN process node, we generate an LNT process as follows.

- Start/End event node: for the start event node n, suppose $n \rightarrowtriangle m$, *i.e., m* is the next node of n. The process for n calls the process for m. For an end event node, its process does nothing but terminates by using the empty statement (**null**).

- Split: we have three types of gateways relative to split structure.

  1. split parallel gateway n: suppose that n has k outgoing branches, *i.e.,* $n \rightarrowtriangle m_i, i \in \{1, ..., k\}$. The LNT process models the parallel execution of all outgoing branches using the LNT parallel operator (**par**). Each branch $m_i, i \in \{1, ..., k\}$ is translated by a call to the LNT process encoding the node m_i. In addition, if there exists a corresponding merge parallel gateway node *mp*, we need to generate an additional parallel branch to realize the synchronization point among the different branches. To do so, for this merge, we create a synchronization action *sync* at the beginning of the additional branch and at the end of all other branches. In this way, the additional branch synchronizes with all other branches on *sync* before calling the process for the next node after *mp*.

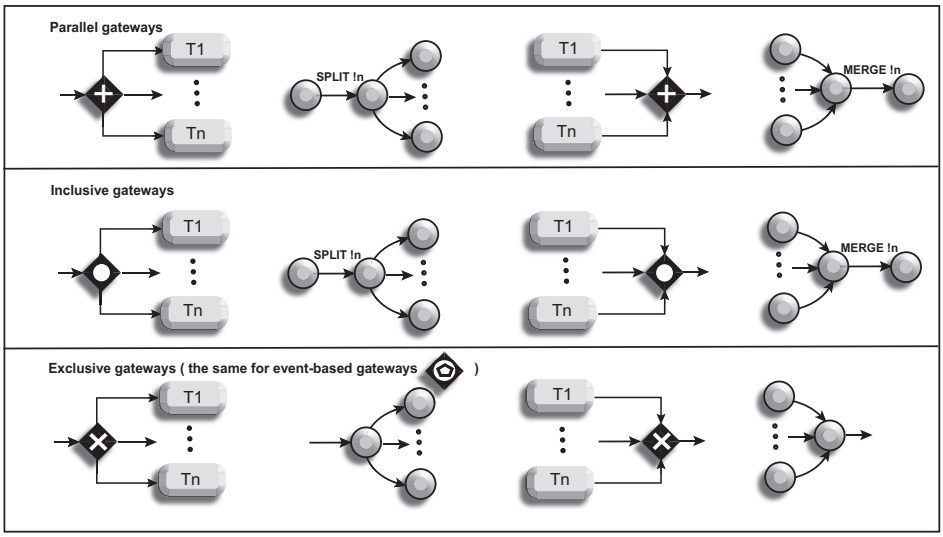

Figure 3: Extra transitions with parallelism information for each gateway considered.

2. split inclusive gateway n: suppose $n \twoheadrightarrow m_i, i \in \{1, ..., k\}$. Any combination of the branches m_i can be executed. For each subset $\{m_{i_1}, ..., m_{i_n}\}, 1 \leq n \leq k, \forall j \in \{1, ..., n\}, i_j \in \{1, ..., k\}$, we obtain all combinations of this subset by using the LNT parallel operator between $m_{i_j}, j \in \{1, ..., n\}$. Then the LNT choice operator (**select**) is used between all combinations of all subsets. If there exists a corresponding merge inclusive gateway, we generate an additional branch for synchronization purpose as above.
3. split exclusive gateway n: suppose $n \twoheadrightarrow m_i, i \in \{1, ..., k\}$. The process models the choice execution of all outgoing branches m_i using the LNT choice operator (**select**). Each branch calls the LNT process encoding the corresponding node.

- Merge: We have three types of merge gateways. A merge gateway has only one outgoing branch.
  1. merge parallel (inclusive, resp.) gateway n: suppose $n \twoheadrightarrow m$. The process for n first synchronizes with all corresponding incoming active branches on the synchronization action *sync* before calling the process for m, *i.e.,* it corresponds to the additional parallel branch produced by translating the split (parallel or inclusive) gateway.
  2. merge exclusive gateway n: suppose $n \twoheadrightarrow m$, then the process for n calls the process for m. Recall that there is no synchronization point for this gateway.
- Sequence: only task nodes are considered as sequential structure since they have exactly one incoming and one outgoing branch. For a task node n, if $n \twoheadrightarrow m$, its process first executes this task that is defined as an action. If m is not a merge (parallel or inclusive) gateway, the process for n calls the process for m. Otherwise, we synchronize the corresponding branch with merge gateway on the synchronization action *sync*, as described in split (parallel or inclusive) gateway.

Given a BPMN process B, the LNT specification obtained using the encoding sketched in this section preserves its execution semantics [11, 22]. Hence, by using classical enumerative exploration techniques, *e.g.*, the LNT compilers of CADP, the LTS generated from this LNT specification corresponds to all possible enactments of B.

3.2 LTS Models

LTS is a low-level formal model, which is defined as follows:

Definition 5. (LTS) An LTS is a tuple $L = (S, s^0, \Sigma, T)$ where S is a finite set of states; $s^0 \in S$ is the initial state; Σ is a finite set of labels; $T \subseteq S \times \Sigma \times S$ is a finite set of transitions.

In an LTS, we denote $s \xrightarrow{\sigma} s\prime$ for $(s, \sigma, s\prime) \in T$. A finite sequence $s_0 \xrightarrow{\sigma_0} s_1 \xrightarrow{\sigma_1} ...s_{n-1} \xrightarrow{\sigma_{n-1}} s_n$ is called a *path*, and the sequence of labels $\sigma_0\sigma_1...\sigma_{n-1}$ is called a *trace*. The LTS obtained from the LNT specification encoded from a given BPMN process represents exactly its execution semantics. We call such an LTS a BPMN LTS or simply LTS if there is no ambiguity.

LEMMA 2. *Each enactment of a BPMN process is represented by a trace in the corresponding LTS. Each trace of a BPMN LTS describes an enactment of its corresponding process.*

PROOF. Suppose that one enactment of a BPMN process is not represented by a trace in its BPMN LTS or one trace in the BPMN LTS does not describe an enactment of the process. This means that either the BPMN LTS does not represent the total execution semantics of the process or it encodes more than the execution semantics. It follows that the LNT specification does not strictly preserve the execution semantics of its BPMN process. However, this is not true according to [11, 22], and thus we get a contradiction. □

3.3 Extended LTS Models

In a BPMN LTS, parallelism is flattened into sequential interleaved executions, which casts away the parallelism information. To compute the degree of parallelism, it is necessary to retrieve such information from the corresponding BPMN process. More precisely, we have to know the number of active split or merge branches for all gateways. In the following, a state s in a BPMN LTS is called an *enabling* state for a BPMN gateway n_g if n_g is ready to be launched from s. Now we show how to extend a BPMN LTS by adding extra transitions with parallelism information for each type of gateways, as shown in Figure 3. This is done by adding extra transitions with corresponding parallelism labels in the LNT specification.

- For a split parallel gateway, its *enabling* state in the LTS is s, suppose $(s\prime \xrightarrow{\sigma})$ s. One extra transition with label *SPLIT* $!n$ is added before s, *i.e.,* $(s\prime \xrightarrow{\sigma})$ $s1 \xrightarrow{SPLIT\ !n} s$, where n represents the number of split branches from this parallel gateway. In a similar way, consider a merge parallel gateway whose *enabling* state is s, suppose s $(\xrightarrow{\sigma} s\prime)$. We add one transition with label *MERGE* $!n$ after s, *i.e.,* $s \xrightarrow{MERGE\ !n} s1$ $(\xrightarrow{\sigma} s\prime)$, where n is the number of merge branches to this parallel gateway.

- For split and merge inclusive gateways, we add extra transitions in the same way as parallel gateways. The difference is that in the label *SPLIT* $!n$ or *MERGE* $!n$, the value n represents the maximum number of split branches or merge branches taken by the gateways. This is because we aim at computing the maximum number of parallel branches.

- For split and merge exclusive gateways, only one branch is active at a time. Thus, we keep the corresponding LTS the same without adding any extra transition. Event-based gateways are treated exactly in the same way as exclusive gateways since they also execute only one branch.

4. DEGREE COMPUTATION

In this section, we show how to compute the degree of parallelism for a given BPMN process B directly on its corresponding Extended LTS, which is called ELTS in the following. We formally define the degree of ELTS, prove that it is equal to that of B, and propose model checking techniques to compute this degree.

4.1 Degree of ELTS

Given a trace π in a BPMN ELTS, the maximum number of active parallel branches at the end of π is called its degree of parallelism. Recall that in the ELTS, the labels of extra transitions contain the information about the maximum number of active split and merge branches for all gateways. Such information can be used to compute the degree of a trace. We consider the default trace degree as 1. After each transition whose label is *SPLIT* $!n$, we will have $n-1$ more parallel branches. In a similar way, for a transition label *MERGE* $!n$, the number of parallel branches will be reduced by $n-1$. In the following, the transition labels added for control flow are called control flow labels.

Definition 6. (Trace degree) Given a trace in an ELTS, denoted $\pi = \sigma_0\sigma_1...\sigma_t$, suppose the set of control flow labels in π is $\{SPLIT\ !i_1, SPLIT\ !i_2, ..., SPLIT\ !i_m, MERGE\ !j_1, MERGE\ !j_2, ..., MERGE\ !j_n\} \subseteq \{\sigma_0, \sigma_1, ..., \sigma_t\}$, the degree of parallelism for the trace π is

$$D(\pi) = 1 + \sum_{k=1}^{m}(i_k - 1) - \sum_{k=1}^{n}(j_k - 1).$$

Definition 7. (ELTS degree) Given a BPMN ELTS L, the set of all traces beginning from the initial state in L is denoted by $\Psi(L)$. The degree of parallelism for this ELTS is the maximum trace degree, $D(L) = Max_{\pi\in\Psi(L)}D(\pi)$.

THEOREM 1. *The degree of parallelism for a BPMN ELTS is equal to the degree of parallelism for the corresponding BPMN process.*

PROOF. Suppose that the degree for a BPMN ELTS, denoted by $D\prime$, does not equal to that for its corresponding BPMN process, denoted by D, *i.e.,* $D\prime \neq D$. Recall that $D\prime$ denotes the maximum trace degree in the ELTS and D equals to the maximum enactment degree in the process. Hence, the inequality of $D\prime$ and D implies that either the trace with maximum degree in ELTS does not represent an enactment of the process or the enactment with maximum degree in the process is not described by a trace in the ELTS. Both cases contradict with Lemma 2. □

Example 2. Consider the trace in Figure 4, which is contained in the ELTS for Example 1. In this trace, we have four control flow labels: two *SPLIT* $!2$ and two *MERGE* $!2$. From Definition 6, the degree for this trace is $1 + (2 * (2 - 1)) - (2*(2-1)) = 1$. In other words, at the end of this trace, the number of active parallel branches is 1. Now consider its (sub) trace from beginning until $T5$, we have two control flow labels, both being *SPLIT* $!2$. The degree for this (sub) trace is $1 + ((2-1) + (2-1)) = 3$. This means that at the end of $T5$, the maximum number of active parallel branches is 3. To calculate the degree of the BPMN process, *i.e.,* the maximum trace degree from Theorem 1 and Definition 7, we must explore all traces of the corresponding ELTS.

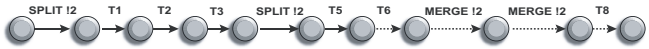

Figure 4: One trace of the ELTS for the BPMN example given in Figure 2.

4.2 Degree Computation by Model Checking

Given a BPMN process with finite degree, we propose model checking techniques to compute this degree on its corresponding ELTS. Recall that this degree is the maximum trace degree in the ELTS. Formula 1, written in MCL (*Model Checking Language*) [17], defines the property *SPLIT_MERGE(N)*, which expresses the existence of a trace whose degree is greater than or equal to N:

Formula 1.

$$SPLIT\_MERGE(N) = \quad (1)$$
$$mu\ X\ (c : Nat := 1). \quad (2)$$
$$((c >= N)\ or \quad (3)$$
$$< SPLIT\ ?\ fan\_out : Nat > X(c + (fan\_out - 1))\ or \quad (4)$$
$$< MERGE\ ?\ fan\_in : Nat > X(c - (fan\_in - 1))\ or \quad (5)$$
$$< not(SPLIT...\ or\ MERGE...) > X(c)) \quad (6)$$

More precisely, given a natural number N, there are two stop conditions for Formula 1. One is that after exploring all traces, there is no trace whose degree is greater than or equal to N, which means that the degree of the corresponding process is smaller than N. In this case, the returned value is false. The other one is that the degree of the trace currently explored becomes greater than or equal to N, in which case the process degree must be greater than or equal to N. Under this condition, the formula returns true. The functions used in Formula 1 are described as follows:

- Line 2 creates a parameter c of type natural number representing the degree of the currently explored trace. It is initialized as 1 because we consider the default degree for any trace as 1.
- Line 3 gives the positive stop condition, under which the value true is returned, *i.e.,* the degree is not smaller than N.
- Line 4 indicates how to deal with a transition with label *SPLIT* !*fan_out*, where *fan_out* is the number of active outgoing branches from the corresponding gateway. During the exploration of an ELTS, when arriving at such a transition, the degree of the current trace should be increased by $fan\_out - 1$.
- In a similar way, line 5 shows how to handle a transition with label *MERGE* !*fan_in*, where *fan_in* is the maximum number of active incoming branches to the corresponding gateway. Whenever reaching such a transition, the degree of the current trace should be subtracted by $fan\_in - 1$.
- Line 6 means that when meeting a transition whose label is not a control flow one, we continue to explore the ELTS without changing the degree.

COROLLARY 1. *Given a BPMN ELTS, if the value returned for SPLIT_MERGE(N) is true, then the degree of the corresponding process is greater than or equal to the number N. Otherwise, the degree is smaller than N.*

For the sake of efficiency, we apply Formula 1 to compute the degree for a given BPMN ELTS by adopting a dichotomic search. More precisely, we check the degree by verifying *SPLIT_MERGE*(2^1) for the first time since the default degree is 1. If the returned value is true, the degree is not smaller than 2. We continue to check the next number, denoted by *numFuture*, which is 2^2 for the second time and 2^m for the m^{th} time until the returned value is false. We keep the last checked number for which the formula returns false, denoted by *negRecent*, as well as that for which the formula returns true, denoted by *posRecent*. The next number to be checked is in the middle of *negRecent* and *posRecent*, *i.e.,* $numFuture = (negRecent + posRecent)/2$. The stop condition is that *negRecent* is greater than *posRecent* by only 1. Algorithm 1 shows the pseudo code for this dichotomic search procedure, which is a variant of the classic binary search algorithm existing in the literature [24].

THEOREM 2. *If the value of $(negRecent - posRecent)$ is 1, then the number posRecent is exactly the degree of parallelism for the given process.*

Algorithm 1 Computing the degree of parallelism using a dichotomic search

```
 1: Input: BPMN ELTS
 2: Initializations: posRecent:=1; negRecent:=1;
    numFuture:=2
 3: while ((negRecent-posRecent) ≠ 1) do
 4:   if SPLIT_MERGE(numFuture) then
 5:     posRecent:=numFuture
 6:     if negRecent=1 then
 7:       numFuture:=posRecent * 2
 8:     else
 9:       numFuture:=(posRecent+negRecent)/2
10:   else
11:     negRecent:=numFuture
12:     numFuture:=(posRecent+negRecent)/2
13: return posRecent
```

PROOF. We know that *posRecent* denotes the last checked number that is smaller than or equal to the degree of the given process and *negRecent* represents the last checked number that is greater than this degree. If the value of $(negRecent - posRecent)$ is equal to 1, then *posRecent* must be equal to the degree since there is no other natural number between *negRecent* and *posRecent*. □

As for the complexity of the degree computation on the ELTS, since we adopt a dichotomic search strategy, the number of times the formula is verified is proportional to the base 2 logarithm of the degree. Furthermore, the complexity of checking Formula 1 (alternation-free) with parameter N is linear *w.r.t.* the size of the formula (proportional to N) and the size of the ELTS (number of states and transitions). Note that the degree is bounded by the total number of outgoing branches for all split parallel and inclusive gateways, which is denoted by N^{out}_{p+i}.

THEOREM 3. *Given a BPMN ELTS L, its degree of parallelism can be computed in $O((|S| + |T|) \cdot N^{out}_{p+i} \cdot log(N^{out}_{p+i}))$ time, where $|S|$ ($|T|$, resp.) is the number of states (transitions, resp.) of L.*

5. EVALUATION

In this section, we first present the architecture of our tool, which is integrated with the CADP toolbox [10]. CADP is a verification toolbox dedicated to the design, analysis, and verification of asynchronous systems consisting of concurrent processes interacting via message passing. Then, we present our experimental results obtained by applying it to a set of BPMN processes.

5.1 Tool Support

To evaluate our approach, we implemented a prototype tool (DeCa) connected to CADP. Figure 5 shows our tool architecture. Given a BPMN process, our tool first encodes it into the LNT value-passing process algebra, one of the input languages accepted by CADP, to capture the behavioral semantics of the process. This LNT encoding is an extension of the VerChor platform [11, 12, 22]. VerChor uses Eclipse Indigo and the BPMN 2.0 modeler as front-end for BPMN, and transforms BPMN into the LNT process algebra by using several intermediate translation steps. Afterwards, the CADP compilers are invoked on the corresponding LNT specification. Recall that a process with

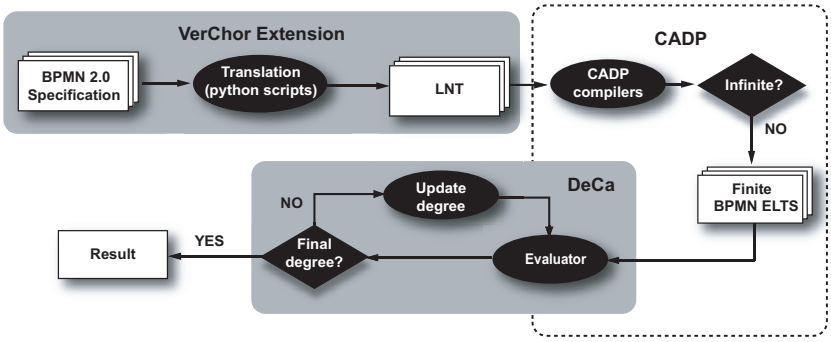

Figure 5: Tool architecture.

infinite degree corresponds to the existence of PAC, which is detected by CADP compilers by checking the finite control property [4]. Hence, only ELTSs for the processes with finite degree are generated. Then, the tool DeCa takes a BPMN ELTS as input and repeatedly invokes the Evaluator 4.0 model checker [16, 17] of CADP to verify Formula 1 encoded in MCL, until the final degree is obtained. This is done by adopting a dichotomic search to minimize the number of model checking invocations. It is worth noticing that, since the LTSs obtained by our tool encode the semantics of the initial BPMN processes, they can be analyzed using other tools available in CADP, such as interactive simulators or testing tools.

5.2 Experimental Results

All our experiments were conducted by using a (server) machine that has six 3.07 GHz processors and 11.7 GB of RAM. Our DeCa tool is implemented in C, using gcc with version 3.4.3. The version of CADP used in our evaluation is BETA-VERSION 2014-a "Amsterdam".

In our approach, one costly computation is the ELTS construction, which includes two phases. The first one is the transformation from BPMN to LNT specification, which has only linear complexity since the specification is generated directly from each node of the BPMN process (see Section 3.1). The second is about the ELTS generation from the LNT specification, which may suffer from state explosion. This can be greatly improved through abstraction.

From Formula 1 and Algorithm 1, we know that the only necessary information to compute the degree is contained in the control flow labels. To reduce the size of BPMN ELTSs without affecting the computation of the degree, we replace all LNT actions representing BPMN tasks by the empty statement (**null**), which keeps the same branching structure with only control flow labels. In the following, we call such an ELTS *abstracted* ELTS and the original ELTS containing all tasks *raw* ELTS. The complexity of degree computation, which depends on the size of the input ELTS (Theorem 3), is greatly reduced if the dichotomic search is applied on the abstracted ELTS instead of the raw one. This improvement becomes clearer for the BPMN processes with a larger number of tasks, and is also demonstrated by the analysis of our experimental results.

We applied our tool on more than one hundred BPMN processes, 90% of which being handcrafted. The experimental results of some examples are shown in Table 1, including several ones extracted from the literature. For instance, Shipment is the example depicted in Figure 1. ChoreOs1 is a variant used in the ChoreOs project [1] and ChoreOs2 is the one obtained from ChoreOs1 by replacing parallel gateways with inclusive gateways. We also present in the second part of the table a few handcrafted examples to show the scalability of our approach considering that those from the literature are quite small.

In this table, for each BPMN process, we give its size (number of tasks and gateways), the size of raw and abstracted ELTS (number of states and transitions), its degree of parallelism as well as the total required time, including the generation of abstracted ELTS from LNT specification as well as the degree computation on the abstracted ELTS. The number of each type of gateways includes split and merge ones.

BPMN processes may correspond to very large raw ELTSs because the parallel operators from parallel and inclusive gateways are expanded in all the possible interleaved behaviors. However, their abstracted ELTSs have very small size, which only depend on the structure of gateways. For example, P111 has more tasks than P110. Hence, the size of the raw ELTS for P111 is much bigger than that for P110. However, they have exactly the same structure in terms of gateways. This is why they have the same abstracted ELTS and total computation time. In our experiments, calculating the degree on abstracted ELTSs reduces drastically the required time, both for ELTS generation and degree computation, especially for the processes whose raw ELTS has large size. For example, if we compute the degree directly on the raw ELTS for P111, it requires about 36 minutes, whereas it takes only 8 seconds on the abstracted ELTS.

With Formula 1 and the ELTS, we can also obtain some important information about the enactment(s) of the BPMN process whose degree is the maximum one. To do this, we add gateway identifiers to all control labels of extra transitions in ELTS, *i.e.,* $MERGE\,!n$ becomes now $MERGE\,!n\,!id_g$, where id_g is the identifier of the gateway associated to this extra transition. The Evaluator 4.0 model checker of CADP provides diagnostics (witnesses and counterexamples) ex-

Table 1: Experimental results for computing the degree of parallelism, where |TS|, |P|, |I|, |E|, and |D| are the number of tasks, parallel, inclusive, exclusive gateways, and the degree of parallelism, respectively.

BPMN	|TS|	|P|	|I|	|E|	raw ELTS($\|S\|/\|T\|$)	abs. ELTS($\|S\|/\|T\|$)	|D|	time
Shipment [21]	8	2	2	2	56/96	9/11	3	2s01
PizzaOrder [21]	9	2	0	0	30/51	3/2	3	1s53
ChoreOs1 [1]	6	4	0	0	20/26	6/5	3	1s56
ChoreOs2	6	0	4	0	22/37	7/8	3	2s03
BookingSystem [22]	6	1	0	1	12/12	2/1	2	1s23
P010	4		4	2	134/275	30/63	3	3s05
P050	5	6			95/220	8/7	4	2s21
P060	8	4	2		576/1596	40/68	5	3s31
P070	30	2		4	20,745/100,234	3/2	6	3s27
P080	40	2			746,505/4,852,234	3/2	8	3s58
P110	25	4	2	2	32,849/245,932	24/36	15	8s15
P111	45	4	2	2	33,554,513/335,544,492	24/36	15	8s18
P120	31	3	2	4	983,188/9,847,132	14/19	17	7s72

plaining the verification result. Precisely, if the verdict is true, then the diagnostic generated is the shortest trace corresponding to the enactment of the BPMN process whose degree is equal to or greater than the input number N. Otherwise, if the verdict is false, then the diagnostic is the whole BPMN LTS to explain that it is impossible to find a trace whose degree is equal to or greater than N. Suppose that the degree of parallelism for a BPMN process is computed as D. If we apply Formula 1 on the number D, then the diagnostic returned actually corresponds to the shortest enactment(s) with the maximum number of parallel branches, which provides valuable information for resource allocation planning.

6. RELATED WORK

First of all, some metrics have been developed for business processes that are regarded as predictors in terms of errors estimation, such as McCabe cyclomatic, CFC and CC metrics [15, 19]. All these complexity metrics were analyzed from an empirical point of view. Differently, the degree of parallelism is a metric that represents the peak demand on tasks to be performed. It can be considered as a worst case metric for business processes, that is a valuable guidance on process modelling and execution.

Several works have focused on providing formal semantics and verification techniques for BPMN, *e.g.*, [23, 27, 28, 8, 6, 22]. The authors of [27] present a formal semantics for BPMN by encoding it into the CSP process algebra. Thus, formal verifications can be carried out against some properties like consistency and compatibility [28]. Decker and Weske present in [6] an extension of BPMN 1.0 (iBPMN) in the direction of interaction modeling. They also propose a formal semantics for iBPMN in terms of interaction Petri nets. [8] presents a mapping from BPMN to Petri nets that enables the static analysis of BPMN models. [23] presents a double transformation from BPMN to Petri nets and from Petri nets to mCRL2. This allows one to use both Petri nets based tools and the mCRL2 toolset for searching deadlocks, livelocks, or checking temporal properties. Our solution, based on a direct translation from BPMN to the LNT process algebra, allows the analysis of business processes using model checking and equivalence checking.

As far as the degree of parallelism for BPMN is concerned, theoretically, it can also be computed by reasoning on Petri net models and determining the bound of a Petri net, which is the maximum number of tokens in a marking of the net. However, to do so, the reachability graph for the net should be constructed entirely. The reachability problem for some specific Petri nets, such as conflict-free Petri nets and 1 safe live free-choice nets [9], is NP-complete. Note that for arbitrary Petri nets, this problem is much harder [18]. This is probably one reason why there is no work on degree computation with Petri nets in the literature. In our paper, we present a very efficient solution for computing the degree of parallelism since we handle LTSs containing only control flow labels by replacing tasks with empty statement.

[25] proposes several algorithms for directly calculating the degree of parallelism of a BPMN process without establishing its behavior model. In this work, a duration constraint is associated to each task, *i.e.,* a task is obliged to be completed within a certain period of time. Furthermore, a task must begin immediately after the completion of its precedent task. Without considering inclusive gateways, they deal with three special cases of BPMN processes: with only one type of gateways; without split exclusive gateway nor cycles; with only two types of gateways. Each case is treated with a different algorithm. Our work focuses on BPMN processes without time constraints because it is not always possible to associate duration to tasks and ensure they can start upon demand, *e.g.*, one task may not start immediately after the precedent one. In this context, we propose a uniform approach, which can automatically analyze the degree of parallelism using model checking techniques for complex BPMN structures, *i.e.,* combining different gateways and cycles.

7. CONCLUSION

In this paper, we show how to represent BPMN processes in terms of ELTSs that contain the information about the maximum number of split and merge branches. This is achieved by an encoding into the LNT value-passing process algebra. It is worth noting that BPMN processes handled by our approach are general enough to be able to contain nested gateways as well as infinite loops. The control-flow informa-

tion is then used for computing the degree of parallelism with model checking techniques. Our approach is fully supported by a chain of tools, including the encoding of BPMN into process algebra, the ELTS generator, and the degree computation. Furthermore, we show how to construct an abstracted ELTS by keeping only control flow labels and replacing tasks with empty statements, which greatly improves performance, as shown in our experimental results. Finally, with the CADP model checker, we produce the enactments of BPMN processes that possess the maximum number of parallel branches, which is a valuable guide for resource allocation planning.

As far as future work is concerned, we first plan to consider a larger subset of BPMN including data objects. Indeed, we currently compute an upper bound of the parallelism degree, which may be actually lower if gateway decisions are controlled by data. Such an extension would allow us to compute (by reusing the same tool chain) the exact degree for a given BPMN process with data control. A second perspective is to investigate other forms of quantitative analysis, such as the throughput and latency of tasks in a BPMN process, which can be computed by extending LTS models with Markovian information and carrying out steady-state analysis [3].

Acknowledgements. This work has been partially supported by the OpenCloudware project (2012-2015), which is funded by the French "*Fonds national pour la Société Numérique*" (FSN), and is supported by *Pôles* Minalogic, Systematic, and SCS.

8. REFERENCES

[1] M. Autili, P. Inverardi, and M. Tivoli. CHOREOS: Large Scale Choreographies for the Future Internet. In *Proc. of CSMR-WCRE'14*, pages 391–394. IEEE, 2014.

[2] D. Champelovier, X. Clerc, H. Garavel, Y. Guerte, C. McKinty, V. Powazny, F. Lang, W. Serwe, and G. Smeding. Reference Manual of the LOTOS NT to LOTOS Translator, Version 5.4. INRIA/VASY, 2011.

[3] N. Coste, H. Garavel, H. Hermanns, F. Lang, R. Mateescu, and W. Serwe. Ten Years of Performance Evaluation for Concurrent Systems using CADP. In *Proc. of ISoLA'10*, volume 6416 of *LNCS*, pages 128–142. Springer, 2010.

[4] M. Dam. Model Checking Mobile Processes. *Information and Computation*, 129(1):35–51, 1996.

[5] T.H. Davenport. *Process Innovation: Reengineering Work Through Information Technology*. Harvard Business School Press, 1993.

[6] G. Decker and M. Weske. Interaction-centric Modeling of Process Choreographies. *Information Systems*, 36(2):292–312, 2011.

[7] Bonita designer (online). Available: http://www.bonitasoft.com/.

[8] R.M. Dijkman, M. Dumas, and C. Ouyang. Semantics and Analysis of Business Process Models in BPMN. *Inf. Softw. Technol.*, 50(12):1281–1294, 2008.

[9] J. Esparza. Reachability in Live and Safe Free-choice Petri Nets is NP-complete . *Theoretical Computer Science*, 198:211–224, 1998.

[10] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. *STTT*, 2(15):89–107, 2013.

[11] M. Güdemann, P. Poizat, G. Salaün, and A. Dumont. VerChor: A Framework for Verifying Choreographies. In *Proc. of FASE'13*, volume 7793 of *LNCS*, pages 226–230. Springer, 2013.

[12] M. Güdemann, G. Salaün, and M. Ouederni. Counterexample Guided Synthesis of Monitors for Realizability Enforcement. In *Proc. of ATVA'12*, volume 7561 of *LNCS*, pages 238–253. Springer, 2012.

[13] ISO/IEC. International Standard 19510, Information technology – Business Process Model and Notation. 2013.

[14] C. Larman. *Applying UML And Patterns: An Introduction To Object-Oriented Analysis And Design And Iterative Development*. Prentice Hall, 2005.

[15] K. Bisgaard Lassen and W. Van der Aalst. Complexity Metrics for Workflow Nets. *Inf. Softw. Technol.*, 51(3):610–626, 2009.

[16] R. Mateescu and M. Sighireanu. Efficient On-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus. *Sci. Comput. Program.*, 46(3):255–281, 2003.

[17] R. Mateescu and D. Thivolle. A Model Checking Language for Concurrent Value-Passing Systems. In *Proc. of FM'08*, volume 5014 of *LNCS*, pages 148–164. Springer, 2008.

[18] E.W. Mayr. An Algorithm for the General Petri Net Reachability Problem. *SIAM J. Comput.*, 13(3):441–460, 1984.

[19] T. McCabe. A Complexity Measure. In *IEEE Transactions on Software Engineering*, volume 2, pages 308–320. IEEE, 1976.

[20] OMG. *Business Process Model and Notation (BPMN) – Version 2.0*. January 2011.

[21] OMG. BPMN 2.0 by Example. 2010.

[22] P. Poizat and G. Salaün. Checking the Realizability of BPMN 2.0 Choreographies. In *Proc. of SAC'12*, pages 1927–1934. ACM Press, 2012.

[23] I. Raedts, M. Petkovic, Y. S. Usenko, J. M. van der Werf, J. F. Groote, and L. Somers. Transformation of BPMN Models for Behaviour Analysis. In *Proc. of MSVVEIS'07*, pages 126–137, 2007.

[24] T. J. Rolfe. Analytic Derivation of Comparisons in Binary Search. *SIGNUM Newsletter*, 32(4):15–19, 1997.

[25] Y. Sun and J. Su. Computing Degree of Parallelism for BPMN Processes. In *Proc. of ICSOC'11*, volume 7084 of *LNCS*, pages 1–15. Springer, 2011.

[26] W. M. P. van der Aalst and A. H. M. Ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30:245–275, 2003.

[27] P.Y.H. Wong and J. Gibbons. A Process Semantics for BPMN. In *Proc. of ICFEM'08*, volume 5256 of *LNCS*, pages 355–374. Springer, 2008.

[28] P.Y.H. Wong and J. Gibbons. Verifying Business Process Compatibility. In *Proc. of QSIC'08*, pages 126–131. IEEE, 2008.