Complexity Metrics for Workflow Nets

Kristian Bisgaard Lassen[a,*], Wil M.P. van der Aalst[b]

[a] *Department of Computer Science, University of Aarhus, IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark.*
[b] *Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands*

Abstract

Process modeling languages such as EPCs, BPMN, flow charts, UML activity diagrams, Petri nets, etc. are used to model business processes and to configure process-aware information systems. It is known that users have problems understanding these diagrams. In fact, even process engineers and system analysts have difficulties grasping the dynamics implied by a process model. Recent empirical studies show that people make numerous errors when modeling complex business processes, e.g., about 20 percent of the EPCs in the SAP reference model have design flaws resulting in potential deadlocks, livelocks, etc. [29]. It seems obvious that the complexity of the model contributes to design errors and a lack of understanding. It is not easy to measure complexity, however. This paper presents three complexity metrics that have been implemented in the process analysis tool ProM. The metrics are defined for a subclass of Petri nets named Workflow nets, but the results can easily be applied to other languages. To demonstrate the applicability of these metrics, we have applied our approach and tool to 262 relatively complex Protos models made in the context of various student projects. This allows us to validate and compare the different metrics. It turns out that our new metric focusing on the structuredness outperforms existing metrics.

Key words: Metrics, Petri nets, Understandability

1. Introduction

No silver bullet exists for building readable process models no matter what process language is used. Often the complexity of a process model is a true reflection of the nature of the problem that needs to be solved. However, the model may also be unnecessarily complex. For example, the same behavior can be represented in different ways. Therefore, it is important to stimulate designers to keep their models as simple as possible. Moreover, if the model is

*Corresponding author

too complex to be understood by end users, then one may choose to decompose or split the model.

Overly complex models cause all kinds of problems. If end-users cannot check the models, this may lead to the implementation of systems that do not adequately support them. Moreover, complex models also lead to all kinds of design flaws. A nice illustration is the empirical work presented in [27, 29, 28]. Using a collection of 2003 Event-driven Process Chain (EPC) [1, 33] models it is shown that at least 10 percent of these models are flawed [27, 29]. 604 of these 2003 models originate from the well-known SAP Reference model [20]. In [28] it was shown that with a rather basic Petri-net-based analysis technique (invariants) already 5.6 percent of these models can be proven to be incorrect (deadlocks, livelocks, dead activities, etc.). Using a more refined technique it was shown that more than 20 percent of the EPCs in the reference model have errors such as deadlocks, livelocks, etc. [29]. Moreover, as shown in [27, 28], it is possible to adequately predict errors based on the complexity. Therefore, we think it is safe to assume that the complexity of the model directly impacts the readability and quality of the model. Therefore, this paper *focuses on measuring the complexity of process models.*

In this paper, we define three metrics. The first two metrics are extensions of existing metrics and the third metric is new. All metrics are defined on a subclass of Petri nets named *WorkFlow nets* (WF-nets) [2, 3, 4]. WF-nets are an abstraction of real-life workflow languages suitable for analysis. However, as we will demonstrate, the results are also directly applicable to other languages such as EPCs, BPMN diagrams, flow charts, and UML activity diagrams.

The first metric extends the metric defined by Cardoso [13] and is purely based on the presence of certain splits and joins in the syntactical process definition. The second metric extends the so-called Cyclomatic metric of McCabe [26]. This metric is based on the state-space. The third metric is a *new metric that better tries to capture the complexity of the model as it is perceived by humans.* It iteratively analyzes the structure of the model and assigns penalties to undesirable constructs from a complexity point of view.

The three complexity metrics are defined differently and are good representatives of the various types of complexity metrics found in the literature. They each try to measure how difficult a process definition is to understand and communicate to others. The scores of the different metrics cannot be compared directly, but by looking at how different processes score with respect to the same metric, it is possible to see if one process is more complicated than the other.

All three metrics have been implemented in the ProM framework [7]. Since ProM supports the mapping of different types of models onto WF-nets, we can apply our results to a wide range of models. To validate our approach and to compare the different metrics, we have applied our approach and tool to 262 relatively complex Protos models. These Protos models have been translated to WF-nets and analyzed using ProM. This case study shows that the new metric based on structuredness provides a major improvement over existing metrics.

The paper is structured as follows. In Section 2 we familiarize the reader with Petri nets and other basic concepts used in this paper. Section 3 presents

the three metrics. A short introduction to the implementation of the metrics is given in Section 4. Section 5 describes a case study where the three metrics are compared and there differences are discussed. Section 6 presents related work and Section 7 concludes the paper.

2. Preliminaries

In this section we would like to familiarize the reader with some key concepts that are important for the understanding of the remainder of this paper.

2.1. Petri nets

This section introduces the basic Petri net terminology and notations. Readers familiar with Petri nets can skip this section.

The classical Petri net is a directed bipartite graph with two node types called *places* and *transitions*. The nodes are connected via directed *arcs*. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles.

Definition 1 (Petri net). A *Petri net* is a triple (P, T, F):

- P is a finite set of *places*,
- T is a finite set of *transitions* ($P \cap T = \emptyset$),
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs* (flow relation)

Note that we do not consider multiple arcs from one node to another. In the context of workflow procedures it makes no sense to have other weights, because places correspond to conditions.

Elements of $P \cup T$ are called *nodes*. A node x is an *input node* of another node y iff there is a directed arc from x to y (i.e., $(x, y) \in F$). Node x is an *output node* of y iff $(y, x) \in F$. For any $x \in P \cup T$, $\overset{N}{\bullet} x = \{y \mid (y, x) \in F\}$ and $x \overset{N}{\bullet} = \{y \mid (x, y) \in F\}$; the superscript N may be omitted if clear from the context.

The *projection* and *union* of a Petri net are defined as follows.

Definition 2 (Projection and union). Let $PN = (P, T, F)$ and $PN' = (P', T', F')$ be Petri nets and $X \subseteq P \cup T$ a set of nodes. $PN|_X = (P \cap X, T \cap X, F \cap (X \times X))$ is the projection of PN onto X. $PN \cup PN' = (P \cup P', T \cup T', F \cup F')$ is the union of PN and PN'.

At any time a place contains zero or more *tokens*, drawn as black dots. The *state*, often referred to as *marking*, is the distribution of tokens over places, i.e., $M \in P \to \mathbb{N}$. We will represent a state as follows: $1p_1 + 2p_2 + 1p_3 + 0p_4$ is the state with one token in place p_1, two tokens in p_2, one token in p_3 and no tokens in p_4. We can also represent this state as follows: $p_1 + 2p_2 + p_3$. To compare states we define a partial ordering. For any two states M_1 and M_2, $M_1 \leq M_2$ iff for all $p \in P$: $M_1(p) \leq M_2(p)$.

The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

(1) A transition t is said to be *enabled* iff each input place p of t contains at least one token.

(2) An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and *produces* one token for each output place p of t.

Given a Petri net (P, T, F) and a state M_1, we have the following notations:

- $M_1 \xrightarrow{t} M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2
- $M_1 \rightarrow M_2$: there is a transition t such that $M_1 \xrightarrow{t} M_2$
- $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \dots t_{n-1}$ leads from state M_1 to state M_n via a (possibly empty) set of intermediate states $M_2, ...M_{n-1}$, i.e., $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} ... \xrightarrow{t_{n-1}} M_n$

A state M_n is called *reachable* from M_1 (notation $M_1 \xrightarrow{*} M_n$) iff there is a firing sequence σ so that $M_1 \xrightarrow{\sigma} M_n$. Note that the empty firing sequence is also allowed, i.e., $M_1 \xrightarrow{*} M_1$.

We use (PN, M) to denote a Petri net PN with an initial state M. A state M' is a *reachable state* of (PN, M) iff $M \xrightarrow{*} M'$.

Let us define some standard properties for Petri nets. First, we define properties related to the dynamics of a Petri net, then we give some structural properties.

Definition 3 (Live). A Petri net (PN, M) is *live* iff, for every reachable state M' and every transition $t \in T$, there is a state M'' reachable from M' which enables t.

A Petri net is *structurally live* if there exists an initial state such that the net is live.

Definition 4 (Bounded, safe). A Petri net (PN, M) is *bounded* iff for each place $p \in P$ there is a natural number n so that for every reachable state the number of tokens in p is less than n. The net is safe iff for each place the maximum number of tokens does not exceed 1.

A Petri net is *structurally bounded* if the net is bounded for any initial state.

For $PN = (P, T, F)$ we also define some standard structural properties.

Definition 5 (Strongly connected). A Petri net is *strongly connected* iff, for every pair of nodes (i.e., places and transitions) x and y, there is a path leading from x to y.

Definition 6 (Free-choice). A Petri net is a *free-choice Petri net* iff, for every two transitions $t_1 \in T$ and $t_2 \in T$, $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ implies $\bullet t_1 = \bullet t_2$.

Definition 7 (State machine). A Petri net is *state machine* iff each transition has at most one input place and at most one output place, i.e., for all $t \in T$: $|\bullet\, t| \leq 1$ and $|t\, \bullet\,| \leq 1$.

Definition 8 (Marked graph). A Petri net is *marked graph* iff each place has at most one input transition and at most one output transition, i.e., for all $p \in P$: $|\bullet\, p| \leq 1$ and $|p\, \bullet\,| \leq 1$.

See [16, 35] for a more elaborate introduction to these standard notions.

2.2. WF-nets

A Petri net which models the control-flow dimension of a workflow, is called a *WorkFlow net* (WF-net, [2]). In WF-net the transitions correspond to activities. Some of the transitions represent "real activities" while others are added for routing purposes (i.e., similar to the structured activities in BPEL). Places correspond to pre- and post-conditions of these activities. It should be noted that a WF-net specifies the dynamic behavior of a single *case* (i.e., *process instance* in BPEL terms) in isolation.

Definition 9 (WF-net). A Petri net $PN = (P, T, F)$ is a WF-net (Workflow net) if and only if:

(i) There is one source place $i \in P$ such that $\bullet i = \emptyset$.

(ii) There is one sink place $o \in P$ such that $o\bullet = \emptyset$.

(iii) Every node $x \in P \cup T$ is on a path from i to o.

A WF-net has one input place (i) and one output place (o) because any case handled by the procedure represented by the WF-net is created when it enters the WFM system and is deleted once it is completely handled by the system, i.e., the WF-net specifies the life-cycle of a case. The third requirement in Definition 9 has been added to avoid "dangling activities and/or conditions", i.e., activities and conditions which do not contribute to the processing of cases.

Given the definition of a WF-net it is easy to derive the following properties [4].

Proposition 1 (Properties of WF-nets). *Let $PN = (P, T, F)$ be Petri net.*

- *If PN is a WF-net with source place i, then for any place $p \in P$: $\bullet p \neq \emptyset$ or $p = i$, i.e., i is the only source place.*
- *If PN is a WF-net with sink place o, then for any place $p \in P$: $p\bullet \neq \emptyset$ or $p = o$, i.e., o is the only sink place.*

- *If PN is a WF-net and we add a transition t^* to PN which connects sink place o with source place i (i.e., $\bullet t^* = \{o\}$ and $t^* \bullet = \{i\}$), then the resulting Petri net is strongly connected.*
- *If PN has a source place i and a sink place o and adding a transition t^* which connects sink place o with source place i yields a strongly connected net, then every node $x \in P \cup T$ is on a path from i to o in PN and PN is a WF-net.*

2.3. Soundness

In this section we summarize some of the basic results for WF-nets presented in [2, 3, 4].

The three requirements stated in Definition 9 can be verified statically, i.e., they only relate to the structure of the Petri net. However, there is another requirement which should be satisfied:

> *For any case, the procedure will terminate eventually and the moment the procedure terminates there is a token in place o and all the other places are empty.*

Moreover, there should be no dead activities, i.e., it should be possible to execute an arbitrary transition by following the appropriate route through the WF-net. These two additional requirements correspond to the so-called *soundness property* [3].

Definition 10 (Sound). A procedure modeled by a WF-net $PN = (P, T, F)$ is sound if and only if:

(i) For every state M reachable from state i, there exists a firing sequence leading from state M to state o. Formally:[1]

$$\forall_M (i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o)$$

(ii) State o is the only state reachable from state i with at least one token in place o. Formally:

$$\forall_M (i \xrightarrow{*} M \ \wedge \ M \geq o) \Rightarrow (M = o)$$

(iii) There are no dead transitions in (PN, i). Formally:

$$\forall_{t \in T} \ \exists_{M, M'} \ i \xrightarrow{*} M \xrightarrow{t} M'$$

[1] Note that there is an overloading of notation: the symbol i is used to denote both the *place* i and the *state* with only one token in place i (see Section 2.1).

Note that the soundness property relates to the dynamics of a WF-net. The first requirement in Definition 10 states that starting from the initial state (state i), it is always possible to reach the state with one token in place o (state o). If we assume a strong notion of fairness, then the first requirement implies that eventually state o is reached. Strong fairness means that in every infinite firing sequence, each transition fires infinitely often. The fairness assumption is reasonable in the context of WFM: All choices are made (implicitly or explicitly) by applications, humans or external actors. Clearly, they should not introduce an infinite loop. Note that the traditional notions of fairness (i.e., weaker forms of fairness with just local conditions, e.g., if a transition is enabled infinitely often, it will fire eventually) are not sufficient. See [3, 23] for more details. The second requirement states that the moment a token is put in place o, all the other places should be empty. The last requirement states that there are no dead transitions (activities) in the initial state i.

Given a WF-net $PN = (P, T, F)$, we want to decide whether PN is sound. In [2] we have shown that soundness corresponds to liveness and boundedness. To link soundness to liveness and boundedness, we define an extended net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$. $\overline{PN}$ is the Petri net obtained by adding an extra transition t^* which connects o and i. The extended Petri net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$ is defined as follows: $\overline{P} = P$, $\overline{T} = T \cup \{t^*\}$, and $\overline{F} = F \cup \{(o, t^*), (t^*, i)\}$. In the remainder we will call such an extended net the *short-circuited* net of PN. The short-circuited net allows for the formulation of the following theorem.

Theorem 1. *A WF-net PN is sound if and only if* $(\overline{PN}, i)$ *is live and bounded.*

PROOF. See [2]. □

This theorem shows that standard Petri-net-based analysis techniques can be used to verify soundness.

Sometimes we require a WF-net to be safe, i.e., no marking reachable from (PN, i) marks a place twice (cf. Definition 4). Although safeness is defined with respect to some initial marking, we extend it to WF-nets and simply state that a WF-net is safe or not.

In literature there exist many variants of the "classical" notion of soundness used here. Juliane Dehnert uses the notion of relaxed soundness where proper termination is possible but not guaranteed [15, 17]. The main idea is that the scheduler of the workflow system should avoid problems like deadlocks etc. In [24] Ekkart Kindler et al. define variants of soundness tailored towards interorganizational workflows. Kees van Hee et al. [19] define a notion of soundness where multiple tokens in the source place are considered. A WF-net is k-sound if it "behaves well" when there are k tokens in place i, i.e., no deadlocks and in the end there are k tokens in place o. Robert van der Toorn uses the same concept in [36]. In [12, 6] stronger notions of soundness are used and places have to be safe. Another notion of soundness is used in [21, 22] where there is not a single sink place but potentially multiple sink transitions. See [36] for the relation between these variants of the same concept. Other references

using (variants of) the soundness property include [18, 25]. For simplicity we restrict ourselves to the classical notion of soundness described in Definition 10. However, the reader is referred to [8] for a definition of the various soundness notions, their relations, and their decidability.

3. Three Complexity Metrics

In this section we will introduce three metrics for determining the readability of a WF-net. The first two metrics are extensions of existing metrics while the third one is a completely new metric. The new metric attempts to capture the real complexity as perceived by the modeler or user. The two extensions are triggered by the translation of the original metrics to WF-nets. The remainder of this section is structured as follows. First of all, we will introduce a version of the Cardoso Metric [13], extended to WF-nets (Section 3.1). Second, we introduce an extended version of the Cyclomatic complexity metric [26] known from procedural programming (Section 3.2). Third, we present our new metric for WF-nets. Finally, we compare the three metrics using an example WF-net.

3.1. Extended Cardoso Metric

Before we introduce the extended Cardoso metric let us first introduce the classic Cardoso metric. To do this it is necessary to define the type of process that Cardoso considers, and we do this in Definition 11.

Definition 11 (Cardoso process). We say that $P = (N, F, J, S)$ is a Cardoso process where

- N is a set of nodes.
- $F \subseteq N \times N$ is a flow relation from between nodes.
- $J : \{n \in N | \bullet n \neq \emptyset\} \rightarrow \{\text{XOR}, \text{OR}, \text{AND}\}$ is the join function.
- $S : \{n \in N | n\bullet \neq \emptyset\} \rightarrow \{\text{XOR}, \text{OR}, \text{AND}\}$ is the split function.

Using the above definition we can define the so-called "Cardoso metric" [13].

Definition 12 (Cardoso metric). Let $P = (N, F, J, S)$ be a Cardoso process. CFC_{XOR}, CFC_{OR}, and CFC_{AND} are three auxiliary functions of type $dom(S) \rightarrow \mathbb{N}$ defined as follows. For $n \in dom(S)$:

- $CFC_{XOR}(n) = |n \bullet|$,
- $CFC_{OR}(n) = 2^{|n\bullet|} - 1$,
- $CFC_{AND}(n) = 1$.

Using these auxiliary functions the Cardoso metric is defined as

$$\begin{aligned} CFC(P) &= \sum_{n \in dom(S): S(n)=XOR} CFC_{XOR}(n) \\ &+ \sum_{n \in dom(S): S(n)=OR} CFC_{OR}(n) \\ &+ \sum_{n \in dom(S): S(n)=AND} CFC_{AND}(n) \end{aligned}$$

As Definition 12 shows, the metric counts the various splits (XOR, OR, and AND) in the net and give each of them a certain penalty. The rationale for each split, is to give a penalty depending on how many different substates it induces when executed. XOR-split $n \in N$ will go to exactly one of the states in $n\bullet$, therefore the penalty is $|n \bullet|$; OR-split $n \in N$ may go to any subset of the succeeding states in $n\bullet$ with the exception of the empty set, so the penalty here is $|\mathbb{P}(n\bullet) \setminus \emptyset| = 2^{|n\bullet|} - 1$; and finally, AND-split $n \in N$ always goes to all subsequent nodes in $n\bullet$ (i.e., one state), so n is given the penalty 1.

The Cardoso metric is applicable for languages with XOR-, OR-, and AND-splits. Petri nets have only two types of nodes: places and transition. However, using places and transition it is possible to model XOR-, OR-, and AND-splits. In fact, using non-free choice nets even more complex choices are possible. Therefore, we have defined a Petri net version of the metric that generalizes and improves the original metric.

Definition 13 (Extended Cardoso Metric). Let $PN = (P, T, F)$ be a WF-net. $ECFC_P : P \to \mathbb{N}$ is an auxiliary function. For any $p \in P$:

- $ECFC_P(p) = |\{t \bullet | t \in p\bullet\}|$.

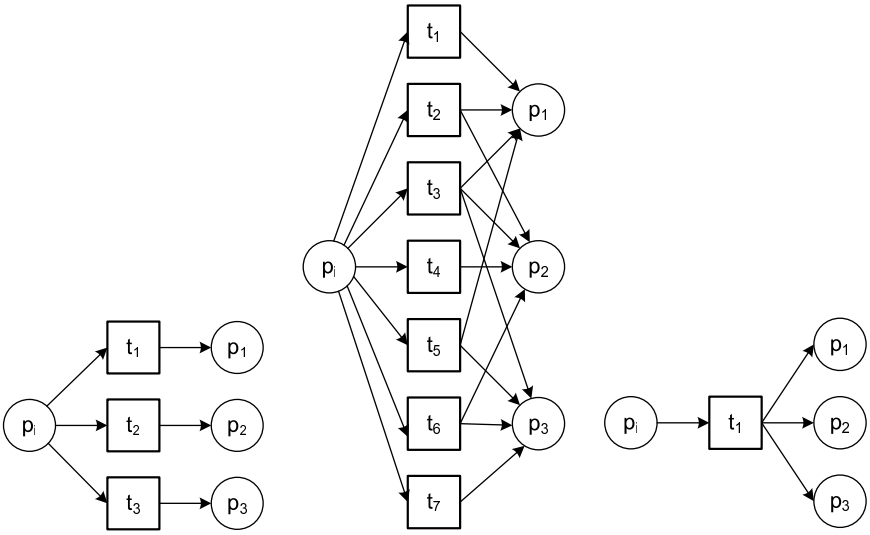
We define the extended Cardoso metric (ECaM) for Petri net as

$$ECaM(PN) = \sum_{p \in P} ECFC_P(p)$$

As for the Cardoso metric, the ECaM metric penalizes each state by how many direct successor states it induces. $ECFC_P$ in Definition 13 says that the penalty for a place p is the number of subsets of places reachable from a place p. This may not be completely obvious why this is a Petri net version of the Cardoso metric, so let us look a some examples of XOR-, OR-, and AND-splits in a free choice Petri net in Figure 1.

In Figure 1(a) we see an XOR-split. Corresponding to Definition 13 it has value $ECaM(PN_{XOR}) = \sum_{p \in \{p_i, p_1, p_2, p_3\}} ECFC_P = ECFC_P(p_i) + ECFC_P(p_1) + ECFC_P(p_2) + ECFC_P(p_3) = |\{\{p_1\}, \{p_2\}, \{p_3\}\}| + |\emptyset| + |\emptyset| + |\emptyset| = 3+0+0+0 = 3$.[2] Note that $ECaM(PN_{XOR})$ yields the same as the Cardoso metric for a XOR-split n with three succeeding states. Likewise we can compute for Figure 1(b) $ECaM(PN_{OR}) = |\{\{p_1\}, \{p_2\}, \{p_3\}, \{p_1, p_2\}, \{p_1, p_3\}, \{p_2, p_3\}, \{p_1, p_2, p_3\}\}| =$

[2]Note that the three Petri nets in Figure 1 are not WF-nets while the definition of *ECaM* is intended for WF-nets. One can see these nets as fragments of some larger WF-net and the metric does not depend on this requirement.



(a) PN_{XOR}: XOR-split. (b) PN_{OR}: OR-split. (c) PN_{AND}: AND-split.

Figure 1: Splits in three different Petri nets: PN_{XOR}, PN_{OR}, and PN_{AND}.

7, the same as the Cardoso metric gives, i.e., $2^3 - 1 = 7$, and for Figure 1(c) $ECaM(PN_{AND}) = |\{\{p_1, p_2, p_3\}\}| = 1$, again the same as for the Cardoso metric.

The examples show that in many situations the Cardoso metric and our extended version coincide. Only for non-free choice Petri nets there are true differences, because this situation is not covered by the original metric, but it is obvious that our extended version is defined in the same spirit as the Cardoso Metric.

The ECaM metric uses a local/syntactical view of process complexity. The metric only looks at the successor nodes for each place. This means that other behavioral issues resulting from combinations of constructs are not taken into account.

If we assume that there are n places in a net and that there is a number k bounding the number of successor places for each place, we then know that the ECaM will score between $n-1$ and $(n-1)\cdot(2^k-1)$. Note that the output place has no output arcs while all the others have between 1 and k output arcs. This explains the bounds. The time complexity of the ECaM metric is linear in the number of nodes in the WF-net (assuming some upperbound k), or $O(|P \cup T|)$, and is therefore very efficient.

3.2. Extended Cyclomatic Metric

The Cyclomatic metric of McCabe [26] is a very well known metric for measuring the control-flow graph of a procedure of a program. It is defined in

Definition 14.

Definition 14 (Cyclomatic metric). Let G = (V,E) be a control-flow graph. The Cyclomatic metric CM is defined as

$$CM(G) = |E| - |V| + p$$

where p is the number of connected components.

Only WF-nets that are state-machine nets behave as a control-flow graph, because the control-flow graph does not allow for concurrency. Therefore, the Cyclomatic metric needs to be extended to address this.

The Cyclomatic metric was intended to reveal complicated pieces of imperative code by measuring the control-flow graph of that code. McCabe did not consider the actual code, only its behavior, and in the same sense we can extend the Cyclomatic metric to WF-nets by not measuring the actual WF-net structure, but its reachability graph instead. In Definition 15 we define this extended Cyclomatic metric (ECyM) for WF-nets.

Definition 15 (Extended Cyclomatic metric). Let $PN = (P, T, F)$ be a WF-net with source place i. $G = (V, E)$ is the corresponding reachability graph of PN, i.e., $V = \{M \mid i \xrightarrow{*} M\}$ and $E = \{(M_1, M_2) \in V \times V \mid M_1 \rightarrow M_2\}$. The extended Cyclomatic metric (ECyM) is defined as follows:

$$ECyM(PN) = |E| - |V| + p$$

where p is the number of strongly connected components in G.

Note that typically we only consider sound WF-nets. For such nets the state space is finite and the metric can be computed.

The ECyM metric clearly attempts to measure one aspect of what we are interested in: How complicated is the behavior that my model exhibit? It does so by considering which states can the process be in and what transitions may occur. As we will see later in Section 3.4, having a small reachability graph and small ECyM, does not necessarily mean that the WF-net in itself is simple. It may be that the WF-net is extremely complicated with various restrictions, but that the reachability graph is small because its size is reduced by the many restrictions.

To find the time complexity of the ECyM we need to consider two things. The time complexity of generating the reachability graph of the WF-net and calculating the strongly connected components of the graph. Typically we only consider WF-nets that are safe and sound. The time complexity of generating the reachability graph for such a WF-net $PN = (P, T, F)$ is $O(2^{|P|})$. The time complexity for finding the strongly connected components of the reachability graph $G = (V, E)$ is $O(|V| + |E|)$ if the graph is represented as an adjacency list [14]. If we assume that the WF-net is safe we know that the maximal size of the reachability graph will be $|V| = 2^{|P|} - 1$ (all permutations of empty and

non-empty places, without the case of all empty places), and $|E| = |V| \cdot (|V|-1)$ (the reachability graph is strongly connected, with the exception that it is not possible to leave the state $v \in V$ where a token is on the sink place and all other places are empty). This means that the time complexity of the ECyM is $O(2^{|P|} + |V| + |E|) = O(2^{|P|} + |V| + |V| \cdot (|V| - 1)) = O(2^{|P|} + |V|^2) = O(2^{|P|} + (2^{|P|} - 1)^2) = O(4^{|P|})$.

Although the time complexity of the ECyM might discourage its use, it is, however, in practice useful since most real-world safe and sound WF-nets have a reachability graph much smaller than $2^{|P|} - 1$ nodes. We will demonstrate this using the case study in Section 5.

3.3. Structuredness Metric

In this section, we propose a new complexity metric. This metric is triggered by problems related to the existing metrics such as the Cardoso metric and the Cyclomatic metric. *Metrics such as the Cardoso metric only focus on the syntax of the model and ignore the complexity of the behavior.* The AND-split is considered to be more simple than the XOR-split while the number of possible states in a concurrent net may be exponential in the number of tasks. This could be addressed by giving the AND-split a higher penalty. However, this would not solve the problem since the complexity of the behavior results from the interaction between different modeling components. *Metrics such as the Cyclomatic metric only focus on the resulting behavior and ignore the complexity of the model itself.* There may be two different models that have the same state space where one model is compact and simple while the other one is large and difficult. The addition of an implicit place (i.e., a place that does not affect the behavior) may make the net more complex because it becomes bigger. However, in some cases, such a place can also make the net simpler because of symmetry reasons. Variants of the Cardoso metric and Cyclomatic metric have two problems in common: they focus on a single aspect (behavior versus syntax) and they do not consider the interaction between the different elements. A net consisting of just AND-splits and AND-joins or just XOR-splits and XOR-joins is typically much simpler than a net with all kinds of mixtures of choice and concurrency. This also applies to parts of the process model, e.g., one long sequence is much easier to understand than a sequence interrupted by splits and joins. Moreover, certain types of patterns are more tricky than others, e.g., constructs involving arbitrary loops, milestones, and synchronizing merges [10] are perceived as complex.

The idea behind this metric stems from the observation that WF-nets are often *structured in terms of design patterns.* Parts of the process model may implement basic patterns such as sequence, choice, iteration, etc. or more advanced control-flow patterns [10]. These have a varying degree of how well they are understood and communicated by people. To capture this insight we have made a metric that recognizes different kinds of structures and scores each structure by giving it some "penalty" value. The sum of these values is used to define our *Structuredness Metric* (SM).

To define this metric, we need to introduce some notations. In Definition 16 we define the notion of a component, which is basically a WF-net with the exception that the source and/or sink may be transitions.

The idea is that a component corresponds to a behavioral pattern. In our approach we try to identify atomic patterns. Then iteratively these atomic patterns are replaced by new transitions. This means that the WF-net is reduced in several steps by identifying and removing patterns. Each time a component is removed the weight, or cost, of this component is calculated and associated with the new transition that the component is replaced by. The algorithm for calculating SM is finished when the WF-net is reduced to a trivial WF-net with just one transition. The complexity score of SM is then weight associated with the single transition.

Definition 16 (Component). Let $PN = (P, T, F)$ be a WF-net. C is a component of PN if and only if

(i) $C \subseteq P \cup T$,

(ii) there exists different source and sink nodes $i_C, o_C \in C$ such that

- $\bullet(C \setminus \{i_C\}) \subseteq C \setminus \{o_C\}$,
- $(C \setminus \{o_C\})\bullet \subseteq C \setminus \{i_C\}$, and
- $(o_C, i_C) \notin F$.

The definition of a component will be used to iteratively identify patterns. To define the different patterns (component types), we need the following notations.

Definition 17 (Component notations). Let $PN = (P, T, F)$ be a WF-net and let C be a component of PN with source i_C and sink o_C. We introduce the following notations and terminology:

- C is a PP-component if $i_C \in P$ and $o_C \in P$,
- C is a TT-component if $i_C \in T$ and $o_C \in T$,
- C is a PT-component if $i_C \in P$ and $o_C \in T$,
- C is a TP-component if $i_C \in T$ and $o_C \in P$,
- $\overline{C} = C \setminus \{i_C, o_C\}$,
- $PN||_C =$
  - $PN|_C$ if $i_C \in P$ and $o_C \in P$,
  - $PN|_{\overline{C}} \cup (\{p_{(i,C)}\}, \{i_C\}, \{(p_{(i,C)}, i_C)\}) \cup (\{p_{(o,C)}\}, \{o_C\}, \{(o_C, p_{(o,C)})\})$ if $i_C \in T$ and $o_C \in T$,[3]

[3]Note that $p_{(i,C)}$ are $p_{(o,C)}$ are the (fresh) identifiers of the places added to make a transition bordered component place bordered.

- $PN|_C \cup (\{p_{(o,C)}\}, \{o_C\}, \{(o_C, p_{(o,C)})\})$ if $i_C \in P$ and $o_C \in T$,
- $PN|_C \cup (\{p_{(i,C)}\}, \{i_C\}, \{(p_{(i,C)}, i_C)\})$ if $i_C \in T$ and $o_C \in P$.

- $[PN]$ is the set of non-trivial components of PN, i.e., all components containing two or more transitions.

This definition introduces PP-, TT-, PT-, TP-components to acknowledge that components can be place and/or transition bordered. The function $PN||_C$ maps a component to a WF-net that is place bordered by adding places to $PN|_C$ if needed.

In [11] we prove that $PN||_C$ yields a safe and sound WF-net if PN is a safe and sound WF-net. This result is very important for the remainder because it shows that components in a safe and sound WF-net can be seen as "black boxes" that have a well-defined input-output behavior. This allows us to iteratively replace components by transitions until we obtain a trivial WF-net consisting of just one transition.

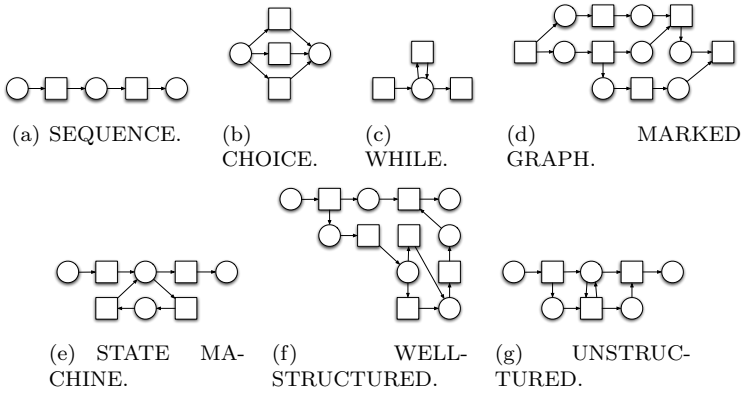


Figure 2: Different component types.

Figure 2 shows some of the components we are interested in. To find such components we provide the following definitions.

Definition 18 (Component types). Let $PN = (P, T, F)$ be a WF-net, $C \in [PN]$ a component in PN, and $PN||_C = (P_C, T_C, F_C)$.

- C is a SEQUENCE-component if and only if $PN||_C$ is a state machine and a marked graph.

- We say that C is a MAXIMAL SEQUENCE-component if and only if $\forall C' \in [PN]$, C' a SEQUENCE-component: $C \neq C' \Rightarrow C \not\subset C'$.

- C is a CHOICE-component if and only if $P_C = \{p_i, p_o\}$ and $T_C = p_i\bullet = \bullet p_o$.

- C is a WHILE-component if and only if $P_C = \{p\}$, $T_C = \{t_i, t, t_o\}$, and $F_C = \{(t_i, p), (p, t), (t, p), (p, t_o)\}$.
- C is a MARKED-GRAPH component if and only if $PN||_C$ is a marked graph net.
- We say that C is a MAXIMAL MARKED-GRAPH-component if and only if $\forall C' \in [PN]$, C' a MARKED-GRAPH-component: $C \neq C' \Rightarrow C \not\subset C'$.
- C is a STATE-MACHINE-component if and only if $PN||_C$ is a state machine net.
- We say that C is a MAXIMAL STATE-MACHINE-component if and only if $\forall C' \in [PN]$, C' a STATE-MACHINE-component: $C \neq C' \Rightarrow C \not\subset C'$.
- C is a WELL-STRUCTURED component if and only if $\forall (p, t) \in F_C : (|p\bullet| > 1 \Rightarrow |\bullet t| = 1) \wedge (|\bullet t| > 1 \Rightarrow |\bullet p| = 1)$ and $\forall n \in P_C \cup T_C : (n, n) \notin F_C^*$ (i.e. no cycles).
- We say that C is a MAXIMAL WELL-STRUCTURED-component if and only if $\forall C' \in [PN]$, C' a WELL-STRUCTURED-component: $C \neq C' \Rightarrow C \not\subset C'$.
- C is an UNSTRUCTURED-component if none of the above definitions apply for C.
- C is a MINIMAL-UNSTRUCTURED component if and only $\forall C' \in [PN]$: $C \neq C' \Rightarrow C' \not\subset C$.

Most of the component types correspond to well-known subclasses of Petri nets (state machines and marked graphs) or basic programming constructs (while loop). Also constructs from the π-calculus [32] such as prefixing and parallism can be expressed (sequences and marked graphs). The only component type that requires some explanation is the WELL-STRUCTURED component. The first requirement $|p\bullet| > 1 \Rightarrow |\bullet t| = 1$ is similar to the free-choice property. It is slightly more strict, i.e., if a place has multiple outgoing arcs the corresponding transitions should not synchronize. In other words: choice and synchronization are separated. The second requirement $|\bullet t| > 1 \Rightarrow |\bullet p| = 1$ can be seen as a dual property of the free-choice property. If a transition needs to synchronize, it should not be directly preceded by an XOR-join. This implies that for every synchronization transition, the set of transitions that supply tokens for such a synchronization is fixed. The third requirement states that the component should not have a cycle. Some process modeling languages locally enforce such requirements to keep the modeling and implementation simple. A nice example is the *flow* construct in BPEL that corresponds exactly to a well-structured component [11]. Note that alternative component types could be defined. This would not change the essence of our approach. The unstructured component type acts as a last resort, i.e., if no nice pattern can be discovered

this pattern is selected. Therefore, we prefer to take the minimal unstructured component rather than the largest one.

As indicated before, the goal is to identify atomic patterns in the form of component types and replace them by transitions. By doing this iteratively the whole WF-net is reduced until the trivial net is reached. The function *fold* (see Definition 19) takes care of the actual reduction. *fold* reduces a WF-net with a component C by removing C and replacing it by a "fresh" transition. This operation yield a safe and sound WF-net if the original WF-net was safe and sound (see Theorem 3 in [11]). This property is essential as it guarantees that the behavior in the rest of the net does not change by removing the atomic pattern identified.

Definition 19 (Fold). Let $PN = (P, T, F)$ be a WF-net and let C be a non trivial component of PN (i.e., $C \in [PN]$). Function *fold* replaces C in PN by a single transition t_C, i.e., $\mathit{fold}(PN, C) = (P', T', F')$ with:

- $P' = P \setminus \overline{C}$,
- $T' = (T \setminus C) \cup \{t_C\}$,
- $F' = (F \cap ((P' \times T') \cup (T' \times P'))) \cup \{(p, t_C) \mid p \in P' \cap (\{i_C\} \cup \bullet i_C)\} \cup \{(t_C, p) \mid p \in P' \cap (\{o_C\} \cup o_C \bullet)\}$.

Before we describe how we calculate the SM score, we extend the notion of a WF-net by adding an annotation (cf. Definition 20). This annotated WF-net allows us to express that each transition has a weight associated with it. When the algorithm for calculating SM reduces the WF-net iteratively, we assign a weight to the transitions corresponding to the collapsed components. The input WF-net to the SM algorithm is an annotated WF-net with annotation function to $\tau(t) = 1, \forall t \in T$. In other words, initially all transitions will have the weight 1.

Definition 20 (Annotated WF-net). $PN = (P, T, F, \tau)$ is an annotated WF-net if and only if:

- (P, T, F) is a WF-net.
- $\tau : T \to \mathbb{R}^+$ is a function that assigns a weight to each transition in PN.

Next we introduce two functions: A function ρ_{PN} that defines a priority of a component type, and a function ω_{PN} that defines the weight, or cost, of a component depending on its type. These two functions are used to determine the order in which components are reduced to transitions and the weight associated to the different component types.

Definition 21 (Component type priority function). Let $PN = (P, T, F, \tau)$ be an annotated WF-net. The component priority function $\rho_{PN} : [PN] \to \mathbb{N}$ is

defined as follows. For any $C \in [PN]$:

$$\rho_{PN}(C) = \begin{cases} 1, & \textbf{if } \text{C is a MAXIMAL SEQUENCE-component;} \\ 2, & \textbf{if } \text{C is a CHOICE-component;} \\ 3, & \textbf{if } \text{C is a WHILE-component;} \\ 4, & \textbf{if } \text{C is a MAXIMAL MARKED-GRAPH-component;} \\ 5, & \textbf{if } \text{C is a MAXIMAL STATE-MACHINE-component;} \\ 6, & \textbf{if } \text{C is a MAXIMAL WELL-STRUCTURED-component;} \\ 7, & \text{otherwise} \end{cases}$$

As function ρ_{PN} shows we first try to select a MAXIMAL SEQUENCE-component and not a MAXIMAL MARKED-GRAPH, MAXIMAL STATE-MACHINE, or MAXIMAL WELL-STRUCTURED-component. The reason is that any net representing a sequence is also a marked graph, state machine and well-structured net. Therefore, things are sorted from more specific patterns to more generic patterns. The penalty associated to the different component types is given by function ω_{PN}.

Definition 22 (Component weight function). Let $PN = (P', T', F', \tau)$ be an annotated WF-net. We define the component weight $\omega_{PN} : [PN] \to \mathbb{R}^+$ as follows. For any $C \in [PN]$, with source i_C and sink o_C, and $PN|_C = (P, T, F)$:

$$\omega_{PN}(C) = \begin{cases} \sum_{t \in T} \tau(t), & \textbf{if } \rho_{PN}(C) = 1; \\ 1.5 \cdot \sum_{t \in T} \tau(t), & \textbf{if } \rho_{PN}(C) = 2; \\ \sum_{t \in \{i_C, o_C\}} \tau(t) + 2 \cdot \sum_{t \in T \setminus \{i_C, o_C\}} \tau(t), & \textbf{if } \rho_{PN}(C) = 3; \\ 2 \cdot \sum_{t \in T} \tau(t) \cdot \mathit{diff}(T), & \textbf{if } \rho_{PN}(C) = 4; \\ 2 \cdot \sum_{p \in P} \tau(t) \cdot \mathit{diff}(P), & \textbf{if } \rho_{PN}(C) = 5; \\ 2 \cdot \sum_{t \in T} \tau(t) \cdot \mathit{diff}(P) \cdot \mathit{diff}(T), & \textbf{if } \rho_{PN}(C) = 6; \\ 5 \cdot (||F| - |P \cup T|| + 1) \cdot \sum_{t \in T} \tau(t) \cdot \mathit{diff}(P) \cdot \mathit{diff}(T), & \text{otherwise} \end{cases}$$

where $\mathit{diff} : 2^{P' \cup T'} \to \mathbb{N} \setminus \{0\}$ is defined as $\mathit{diff}(X) = ||\{x \in X \mid |x \bullet| > 1\}| - |\{x \in X \mid |\bullet x| > 1\}|| + 1$.

diff is a rough measure of how evenly matched split and merge points are in the component; the more unevenly matched these are the higher value of *diff*.

The above component weight function is based on practical experiences. However, the actual function remains subjective and may be configured differently based on the application domain or tools used. The particular choice for ω_{PN} is motivated in the remainder. SEQUENCE-components are considered simple and their weight is the sum of weights of the transitions involved. We feel that the transitions involved in the CHOICE-component are harder to understand than if they were in a SEQUENCE-component, so we multiply the sum of weights by 1.5. The WHILE-component can be seen as a sequence starting with the source transition and ending with the sink transition and in-between these two transitions there is a transition that can be executed an arbitrary number of times. We think that the transition involved in the iteration is more difficult to understand than the source and sink transitions

so we multiply its weight by two and add it to the weight of the other two transitions. The three remaining types of components (MARKED-GRAPH, STATE-MACHINE, and WELL-STRUCTURED) are weighted by multiplying the sum of the involved transitions weights by two and then multiplying the difference in how many splits and joins there exists in the component. The last factor is to reflect that components that have unevenly matched splits and joins are often harder to understand. UNSTRUCTURED-components are given the biggest penalty, and this is because we do not know what behavioral pattern they model. The penalty can be obtained by multiplying three factors: first a factor that describes how many arcs versus nodes multiplied by five – the more arcs there are than nodes, the more difficult we expect the component is to understand; second, the sum of weights of the transitions in the component; and third, the number of joins versus splits in the component. Notice that the UNSTRUCTURED-component penalty is larger than any of the others, so if we, e.g., score a WELL-STRUCTURED-component as an UNSTRUCTURED-component that would yield a larger penalty:

$$\begin{array}{lll} \omega_{PN}(C) & \overset{\rho_{PN}(C)=6}{=} & 2 \cdot \sum_{t \in T} \tau(t) \cdot \mathit{diff}\,(P) \cdot \mathit{diff}\,(T) \\ & < & 5 \cdot \sum_{t \in T} \tau(t) \cdot \mathit{diff}\,(P) \cdot \mathit{diff}\,(T) \\ & \leq & 5 \cdot (||F| - |P \cup T|| + 1) \cdot \sum_{t \in T} \tau(t) \cdot \mathit{diff}\,(P) \cdot \mathit{diff}\,(T) \\ & \overset{\rho_{PN}(C)=7}{=} & \omega_{PN}(C) \end{array}$$

Similar arguments can be used to show that the UNSTRUCTURED-component penalty is always larger than any of the other penalties.

Note that it is impossible to "prove" that the component weight ω_{PN} is the right one. Complexity is perceived in a subjective manner and therefore there is not a "best" way to assign weights. Therefore, we do not present ω_{PN} as "the" component weight function; it is just an example based on experience. Alternative weight functions can be defined as is explained later.

The algorithm that calculates the structuredness metric is given in Definition 23. The intuition behind the algorithm is as follows. First pick a component with the highest priority (i.e., the lowest ρ_{PN} value). After selecting the component use the function *fold* to remove this component from the WF-net. The weight of the added transition, is the weight (i.e. complexity) of the removed component.

This matching followed by reduction is repeated until the net contains no non-trivial components anymore; the weight of the transition in the final trivial WF-net is the SM score. Note that another implication of the algorithm is that components embedded in other components are penalized the more deeply embedded they occur in the WF-net, since we multiply the sum of weights by a number greater than one for most component types (cf. Definition 22). This is, we believe, a desirable property of our metric, since we feel that components that are wrapped in other components make the WF-net, as a whole, more difficult to understand.

Definition 23 (Structuredness metric algorithm). Let $PN = (P, T, F)$ be a WF-net, $\rho_X : [X] \to \mathbb{N}$ be a component priority function and $\omega_X : [X] \to \mathbb{R}^+$ a component weight function for an arbitrary WF-net X. The following algorithm calculates the structuredness metric (SM).

(i) $X := (PN, \tau)$, where $\tau(t) = 1, \forall t \in T$

(ii) while $[X] \neq \emptyset$ (i.e., $X = (PN, \tau)$ contains a non-trivial component)[4]

(iii) pick C so that $\rho_X(C) = min\{\rho_X(C') \mid C' \in [X]\}$

(iv) $PN' := fold(PN, C)$ where t_C is the added transition

(v) $\tau'(t_C) = \omega_X(C)$ and $\tau'(t) = \tau(t)$ for all other t

(vi) $X := (PN', \tau')$

(vii) Output $SM(PN) = \tau(t)$ ($T = \{t\}$ after the net is reduced).

Note that the algorithm will always reduce the WF-net completely. This follows from the observation that for each iteration of the while loop in step (ii) we reduce the net by one component when we use the function *fold*. Since a component has at least two transitions and it is replaced by a single transition, each invocation of *fold* will yield a smaller WF-net with fewer transitions. It is always possible to find a component if the WF-net contains multiple transitions, because as a last resort we fold a MINIMAL UNSTRUCTURED-component. Therefore, the algorithm given in Definition 23 will terminate and yield some positive value.

The time complexity of calculating the structuredness metric $SM(PN)$ can be divided into four parts: (1) the time required for finding a component; (2) the time for recognizing and scoring component; (3) the time needed for executing function *fold*; (4) and the number of times the while-loop in the algorithm of Definition 23 needs to be executed.

In the following let $N = |P \cup T|$. (1) The algorithm we choose to find the components are the following simple one: find all pairs of nodes in the net and see for each pair if they are the source and sink in a component, by checking if all paths from the source end up in the sink node. There may be more clever ways of finding the components, but we choose this version since it has proven efficient in practice. The time complexity of the algorithm is $O(N^3)$, and the reason is that for each possible source node $n \in P \cup T$ there are $N - 1$ possible sink nodes, so this means that there are $N \cdot (N-1) \in O(N^2)$ possible components in a net. Testing if a pair of source and sink node borders a component, takes $O(N)$ time, so all in all finding all components takes $O(N^3)$ time.

(2) Recognizing and scoring a component depends on the component type. Recognizing the component types that we have defined, as well as the evaluating its weight ω in Definition 22 both has an upper bound time complexity of $O(N)$.

[4]Note that this is the case as long as X is not reduced to a WF-net with just a single transition.

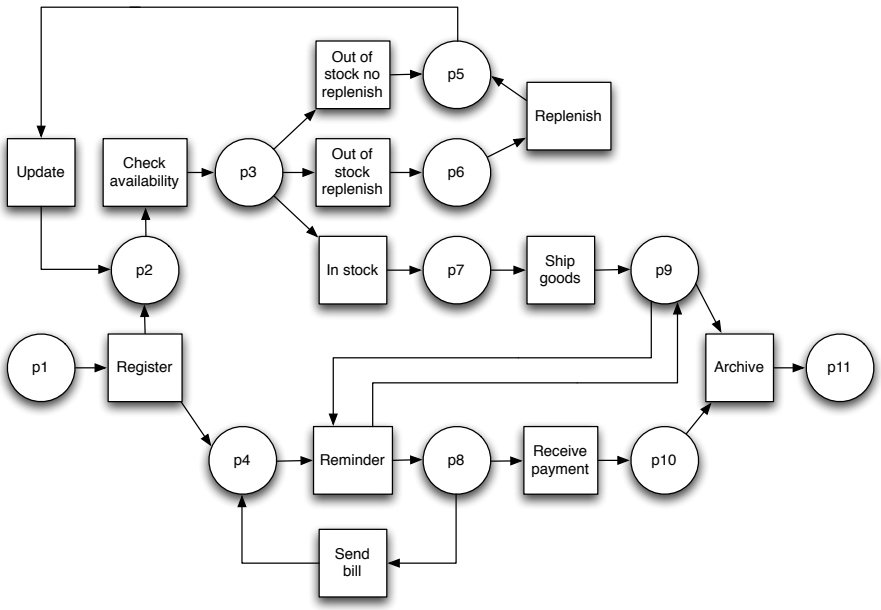


Figure 3: Order handling process.

(3) *fold* also runs in time $O(N)$. So one iteration of the while-loop has a time complexity of $O(N^3)$. (4) The while-loop removes at least one transition in each iteration, the number of iterations is $O(|T|)$, or $O(N)$.

Hence the time complexity for the whole algorithm $O(N^4)$, or $O(|P \cup T|^4)$, since the body of the while-loop runs in $O(N^3)$ and the while-loop iterates $O(N)$ times.

3.4. Comparison of the Three Metrics

We conclude this section by first presenting an example WF-net, where we show how each of the three metrics is calculated, and compare the different scores. The WF-net considered is shown in Figure 3. This is a fragment of an order handling process taken from [5]. Even though the example is small it still uses complicated structures such as the milestone pattern [9, 10] involving place p9 and transition Reminder.

The ECaM metric can easily be calculated.

$$\begin{aligned} ECaM(PN) &= \textstyle\sum_{p \in P} ECFC_P(p) \\ &= |\{t \bullet | t \in p_1\bullet\}| + |\{t \bullet | t \in p_2\bullet\}| + \ldots + |\{t \bullet | t \in p_{11}\bullet\}| \\ &= |\{\{p_2, p_4\}\}| + |\{\{p_3\}\}| + \ldots + |\emptyset| \\ &= 1_{p_1} + 1_{p_2} + 3_{p_3} + 1_{p_4} + 1_{p_5} + 1_{p_6} + 1_{p_7} + 2_{p_8} + 2_{p_9} + 1_{p_{10}} + 0_{p_{11}} \\ &= 14 \end{aligned}$$

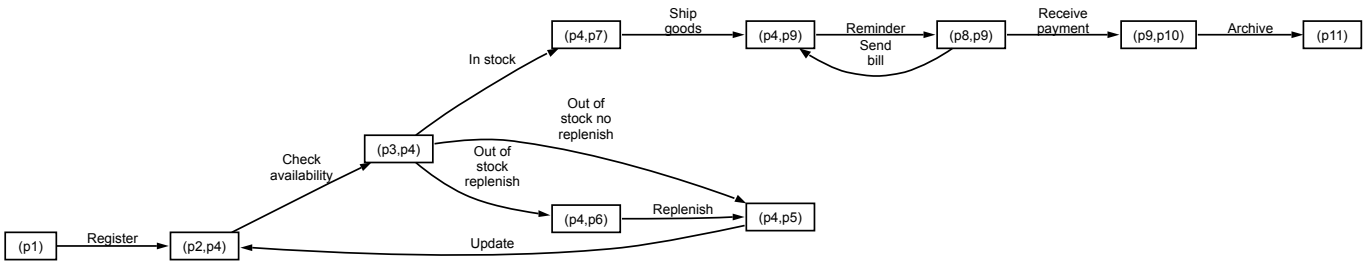


Figure 4: Reachability graph of the order handling process.

To calculate the ECyM metric we first have to generate the reachability graph of the order handling process. The resulting graph is shown in Figure 4. ECyM can be calculated from as follows:

$$ECyM(PN) = |E| - |V| + p = 12 - 10 + 6 = 8$$

This is because the reachability graph has 12 edges, 10 vertices, and 6 strongly connected components.

The 6 components are: $\{(p_1)\}$, $\{(p_2, p_4), (p_3, p_4), (p_4, p_5), (p_4, p_6)\}$, $\{(p_4, p_7)\}$, $\{(p_4, p_9), (p_8, p_9)\}$, $\{(p_9, p_{10})\}$, and $\{(p_{11})\}$.

To calculate the structuredness metric we need to execute the algorithm presented in this paper. Figures 5 through 8 illustrate how the order handling process is broken down and reduced. This is done in three iterations. This leads to the following penalty values:

Iteration 0 $\tau(t) = 1, \forall t \in T$

Iteration 1 $\tau(t_{C1}) = 2, \tau(t) = 1, \forall t \in T \backslash \{t_{C1}\}$ (C is a MAXIMAL SEQUENCE-component and $\omega_X(C) = 1$)

Iteration 2 $\tau(t_{C1}) = 2, \tau(t_{C2}) = 2, \tau(t) = 1, \forall t \in T \setminus \{t_{C1}, t_{C2}\}$ (C is a MAXIMAL SEQUENCE-component and $\omega_X(C) = 1$)

Iteration 3 $\tau(t_{C3}) = 168$ (C is a MINIMAL UNSTRUCTURED-component and $\omega_X(C) = 168$)

Hence $SM(PN) = \tau(t_{C3}) = 168$ for the order handling process.

As expected the three metrics return different results for the process model shown in Figure 3. The ECyM metric scores the net with 8 (the lowest), then comes the ECaM metric with 14, and finally the SM metric assigns a score of 168 (the highest) to the process. The low ECyM metric score can be explained by the fact that the model has a relatively small state space (only 10 states). The ECaM counts how many successor states can be reached from each place.

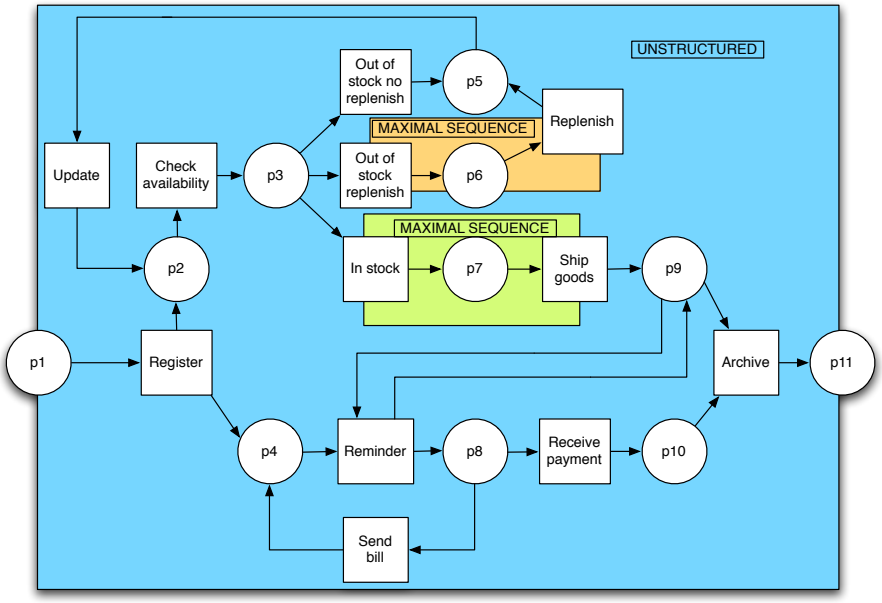


Figure 5: Calculating the SM for the order handling process. Before first iteration.

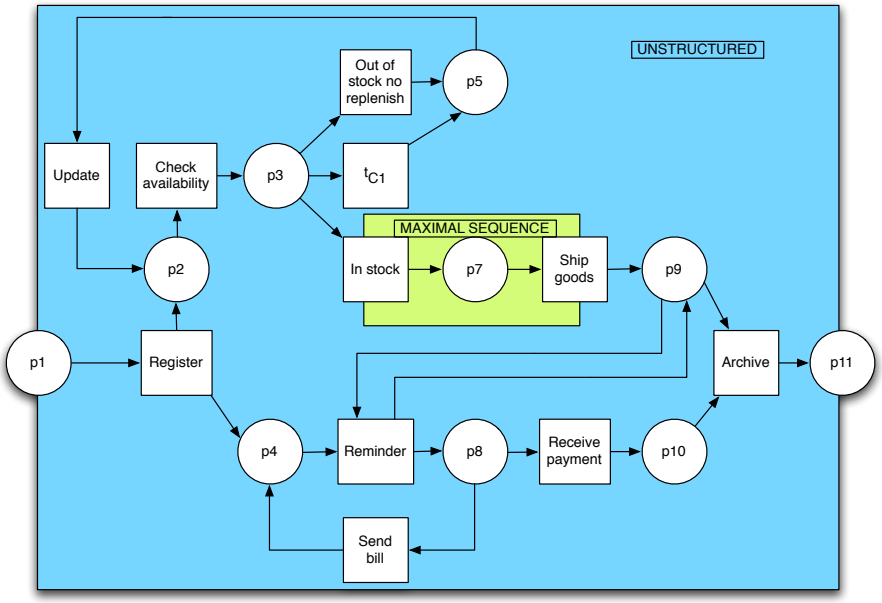


Figure 6: Calculating the SM for the order handling process. Before second iteration.

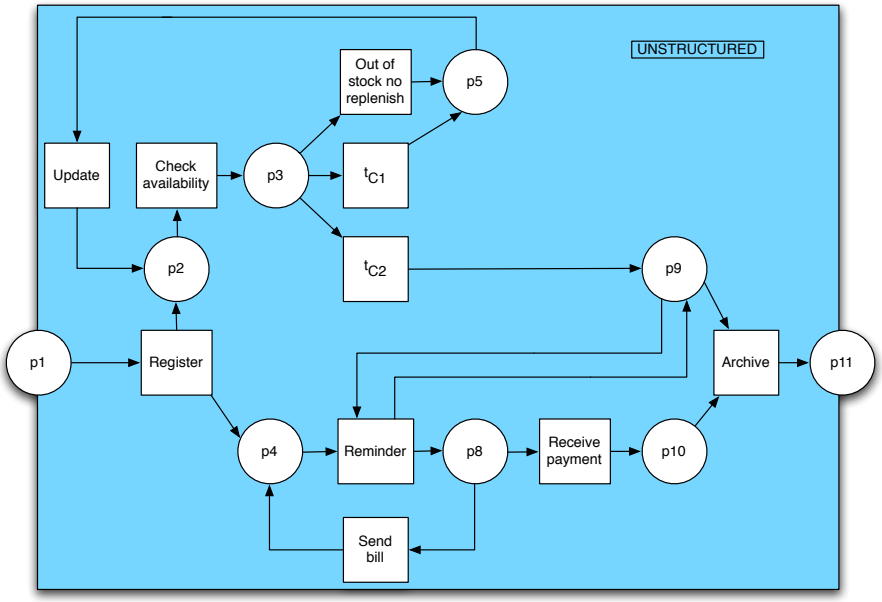


Figure 7: Calculating the SM for the order handling process. Before third iteration.

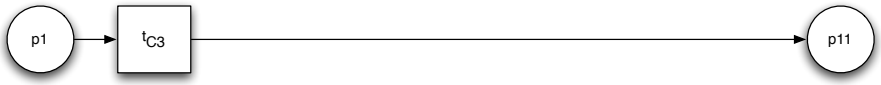


Figure 8: Calculating the SM for the order handling process. After third and final iteration.

Since there are only a few choices this results in a small number. Note that a purely sequential model would have a value of 10 rather than 14. This shows that despite its complex structure the WF-net is not rated as complex by this metric. The score based on the SM metric is the only one that acknowledges the complexity of this relatively small example process.

Note that it is not possible to compare the different metrics on the basis of a single example. This is why we have analyzed and compared 262 real-life process models. Before discussing this case study, we first describe the tool support we have developed.

4. Implementation

The three metrics presented in this paper have been implemented in the context of ProM [7, 34]. ProM is a plugable framework aiming at process analysis. Although most of the functionality is related to process mining, the tool also allows for various kinds of model-based analysis (e.g., verification). The ProM framework hosts a wide variety of plug-ins:

Import plug-ins It is possible to load a wide variety of files into ProM. Some of the important formats are: MXML, PNML (standard format for Petri net), ARIS AML (proprietary format for EPCs), EPML (standard format for EPCs), BPEL, YAWL, and Protos models. All of these are represented as first class objects in the framework. This means that, e.g., importing a PNML file results in a Petri net object which can be used as input for plug-ins that works on Petri nets.

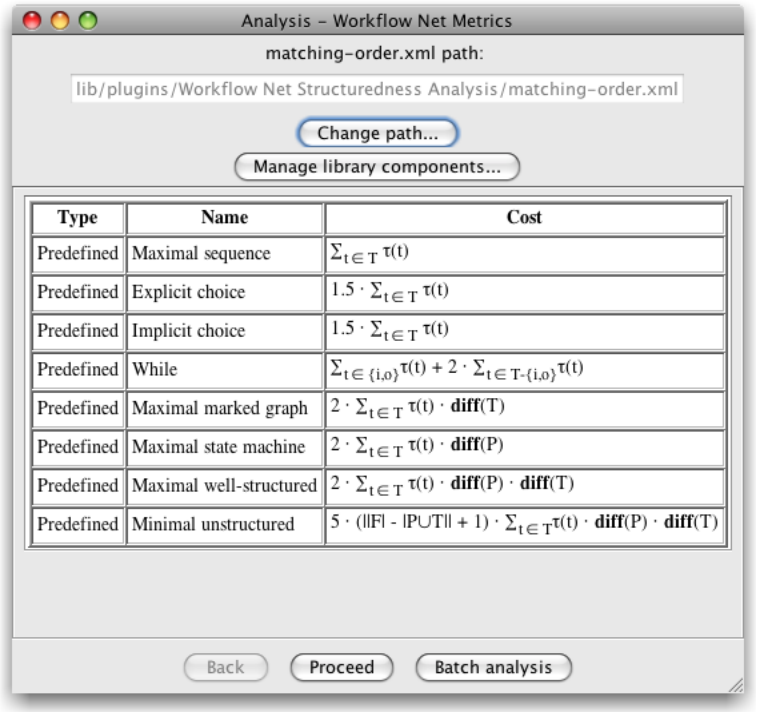
Mining plug-ins ProM was developed as a framework for process mining. Therefore, there are many plug-ins than can be used to discover models. For example, based on audit trails or event logs, various plug-ins can derive Petri nets, EPCs, social networks, etc.

Analysis plug-ins These are plug-ins used for analyzing the various model objects in ProM. For example, there are analysis plug-ins to test soundness of Petri nets or LTL checkers to test properties of an MXML object.

Conversion plug-ins These plug-ins are useful for converting between different object types. For example, it is possible to convert EPCs to Petri nets, Petri nets to BPEL, Petri nets to YAWL, and so on.

Export plug-ins These plug-ins are used to store objects in ProM to files so that they can be used by other tools, e.g., workflow management systems or simulation tools.

The approach presented in this paper is supported by a new analysis plug-in in ProM. This plug-in is easy to use on any Petri net object. First load a Petri net, choose the Workflow Net Metrics plug-in from the analysis menu, and the dialog shown in Figure 9 will appear. Note that the model may also be the



Type	Name	Cost
Predefined	Maximal sequence	$\Sigma_{t \in T} \tau(t)$
Predefined	Explicit choice	$1.5 \cdot \Sigma_{t \in T} \tau(t)$
Predefined	Implicit choice	$1.5 \cdot \Sigma_{t \in T} \tau(t)$
Predefined	While	$\Sigma_{t \in \{i,o\}} \tau(t) + 2 \cdot \Sigma_{t \in T\text{-}\{i,o\}} \tau(t)$
Predefined	Maximal marked graph	$2 \cdot \Sigma_{t \in T} \tau(t) \cdot \mathbf{diff}(T)$
Predefined	Maximal state machine	$2 \cdot \Sigma_{t \in T} \tau(t) \cdot \mathbf{diff}(P)$
Predefined	Maximal well-structured	$2 \cdot \Sigma_{t \in T} \tau(t) \cdot \mathbf{diff}(P) \cdot \mathbf{diff}(T)$
Predefined	Minimal unstructured	$5 \cdot (\lvert F \rvert - \lvert P \cup T \rvert + 1) \cdot \Sigma_{t \in T} \tau(t) \cdot \mathbf{diff}(P) \cdot \mathbf{diff}(T)$

Figure 9: Input dialog for the Workflow Net Metrics analysis plug-in.

result of process mining efforts or some model conversion. In fact it is possible to load an EPC or Protos model, convert it to a WF-net and then apply the plug-in. The dialog shown in Figure 9 allows the user to configure the order by which components are matched in the SM algorithm. In other words, the user may redefine the component priority function ρ.

Figure 9 also shows that different weight functions can be supported. As indicated before, the approach presented in this paper does not depend on a particular component priority ρ and weight ω functions. Therefore, this part is configurable and can be extended.

After the user is satisfied with the setup she may either press the Proceed button, or press the Batch analysis button. The first choice will make the analysis plug-in use the three metrics on a particular net and present the result with statistics of the calculations (see Figure 10). The second choice will lead the user to a file dialog where she must choose a folder that contains either PNML or TPN (a proprietary file format for Petri nets) files in its subfolders. When selecting the Batch analysis option, the plug-in uses the three metrics on all the Petri nets in the subfolder. Moreover, at the end the plug-in also provides different kinds of comparisons and aggregated results. The ability to automatically analyze large collections of models in batch mode is very useful for empirically investigating

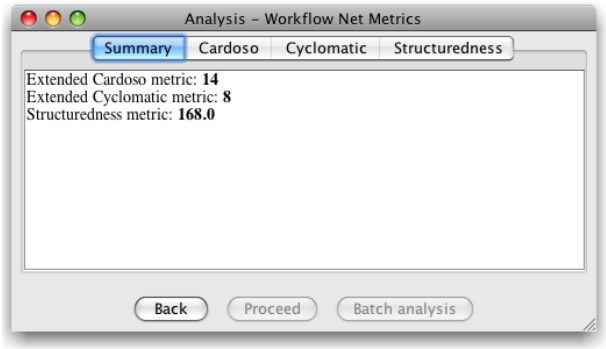


Figure 10: Results from the Workflow Net Metrics analysis plug-in when applied to the order handling process.

complexity in process models. This will be demonstrated in the next section.

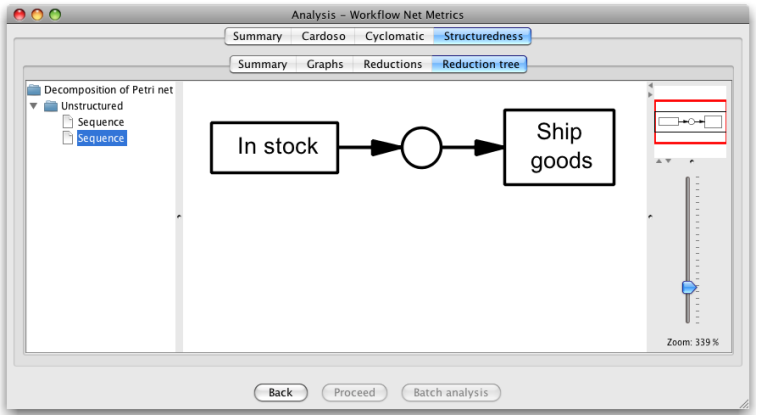


Figure 11: The tree-structure of the order handling process used when calculating the structuredness metric.

Figure 10 shows a summary of the metrics applied to the order handling process. Figure 11 shows the actual derivation of the structuredness metric, i.e., ProM is able to show the iterative process of collapsing components into transitions until one transition remains.

5. Comparison of Metrics: A Case Study

To evaluate the applicability of our approach and to compare and evaluate the different metrics we used 262 process models created using Protos. Protos (Pallas Athena) uses a Petri-net-based modeling notation and is a widely used

business process modeling tool. It is used by more than 1500 organizations in more than 20 countries. The number of users that use Protos for designing processes is estimated to be 25000. Some of the organizations have modeled more than 1500 processes. The 262 process models used for the evaluation resulted from student projects where students had to model and redesign realistic business cases using Protos. These models were collected in 2005, 2006, and 2007.

Protos can be used to model different perspectives, i.e., besides the control-flow also organizational structures, timing information, and data can be modeled. In this study we focused on the control-flow perspective. Protos supports different splits and joins. Any split or join is of type XOR, AND, or OR. Therefore, we first had to translate the Protos models into WF-nets. Fortunately, it is quite straightforward to automatically convert Protos models into WF-nets using ProM. The resulting 262 WF-nets were then analyzed using the new ProM plug-in described in this paper.

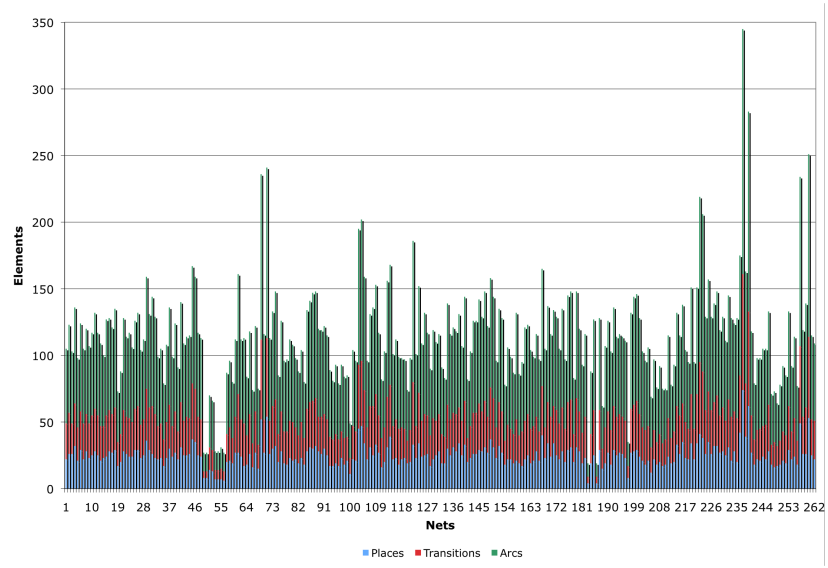
In Figure 12(a) we see the number of elements (places, transitions, and arcs) that each individual net contains. Figure 12(b) shows a different view that indicates how many nets have a certain number of places, transition, and arcs. The two figures show that the input nets span a wide range, i.e., from nets with only 19 elements to nets with 345 elements. On average each net contains 25 places, 28.2 transitions, and 62.26 edges. The correlation coefficients[5], $\rho_{Place,Transition} = 0.93$, $\rho_{Place,Arc} = 0.94$, and $\rho_{Transition,Arc} = 0.99$, indicate that there is a strong dependency between the number of places, transitions, and arcs in the input nets[6]. In a sense, the proportion of places, transitions and arcs is similar for all of the nets. The distribution of elements in the input nets is 22% places, 24% transitions, and 54% arcs.

In Figure 13 we show how the three metrics scored on the input nets. The y-axis is logarithmic since the scores generated by ECyM and SM cover a wide range of values. The mean score of the ECaM is 30.56, ECyM is 169.94, and SM is 1381.57. The correlation coefficients between the different scores are $\rho_{ECaM,ECyM} = 0.16$, $\rho_{ECaM,SM} = 0.32$, and $\rho_{ECyM,SM} = -0.02$. From this we can see that $ECaM, ECyM$ and $ECyM, SM$ seem to be pairwise independent since their correlation coefficients are rather low. Even though $\rho_{ECaM,SM} = 0.32$ is not as low as the two other correlation coefficients, we can say that $ECaM, SM$ are not strongly correlated and relatively independent.

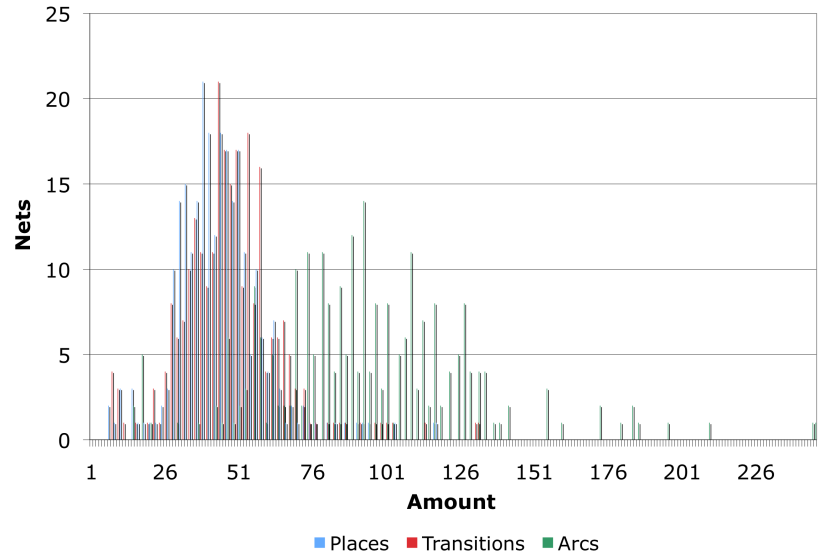
The correlation coefficients suggest that it may be worthwhile to introduce two new composed metrics: $(ECaM, ECyM)$ and $(SM, ECyM)$. For each pair it is clear that the functions do not linearly depend on each other. In other words, they seem to be orthogonal, and it could therefore make sense to define

[5]In this paper we use the term "correlation coefficient" to refer to Pearson's product-moment coefficient.

[6]Usually we interpret correlation coefficients $\rho_{X,Y}$ in the following way. A correlation coefficient between 0 and 0.5 indicates the range from no dependency to a stronger but still not definite dependency. A correlation coefficient between 0.5 and 1 corresponds to the range from a weak dependency to a strong dependency.



(a) Number of places, transitions and edges per net.



(b) Occurence of nets different amounts of places, transitions and edges.

Figure 12: Statistics for the 262 input nets.

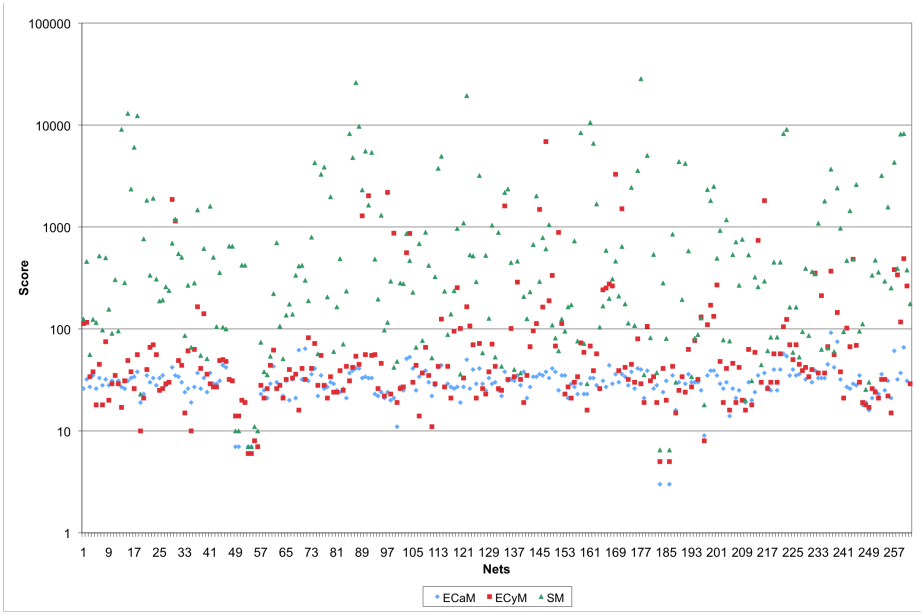


Figure 13: Scores of the three metrics.

these two metrics. Of course, the input nets of our case study, do not reflect all possible process definitions, so we may come across input nets in other case studies where the metrics are not orthogonal. But we do, however, feel that our data set provides a good representation of the process definitions that can be found in real life, so in this perspective it would really make sense to not use a single metric of the three metrics, but rather the pair $(ECaM, ECyM)$ or $(SM, ECyM)$.

We will not go into more details on how the different metrics scored for the individual nets. We did, however, make some observations on why each the metrics scored high for particular nets. ECaM scored high in situations where the input net had a high degree of fan-out from places. This was to be expected based on the way it was defined. ECyM scored high on nets that had a high degree of parallelism. Parallelism in a process typically leads to large reachability graphs, and since ECyM indeed measures the complexity of the reachability graph, it is no surprise that ECyM score high on nets that has parallelism. Finally, SM scored in general high on input nets that had many layers of components in components, but also on nets where MARKED-GRAPH, STATE-MACHINE, or WELL-STRUCTURED components did not have a matching number of splits and joins.

In addition to the scores of the three metrics, we also looked at what components where "matched" while iteratively calculating SM and how they were scored, not only by SM itself, but also how ECaM and ECyM scored these.

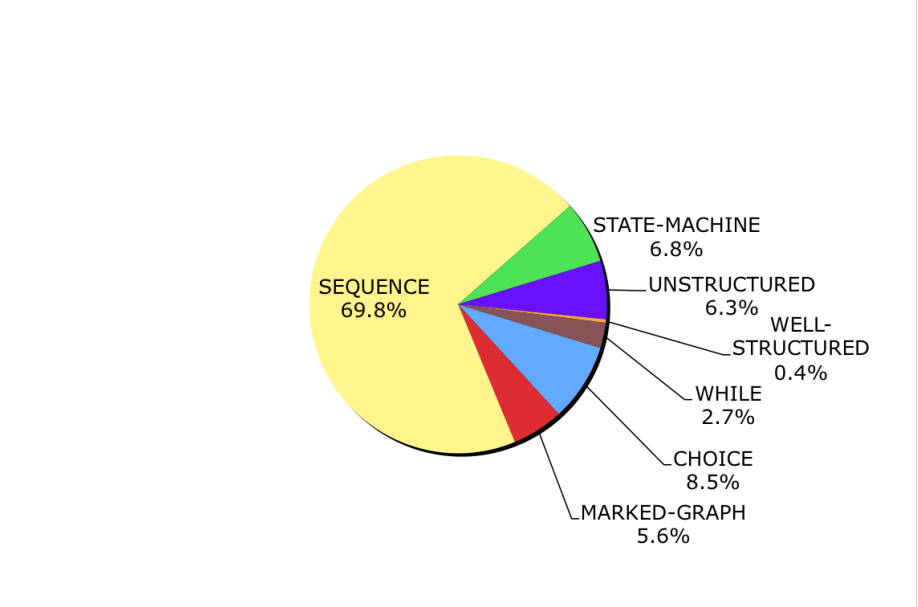


Figure 14: Components matched by the SM-algorithm.

In Figure 5 we present the different types of components that we discovered and how frequently they were used in the reduction (i.e., the components replaced using the function *fold*). All in all we matched and reduced 2769 components, including 234 CHOICE-components, 155 MAXIMAL MARKED-GRAPH-components, 1932 MAXIMAL SEQUENCE-components, 187 STATE MACHINE-components 175 MINIMAL UNSTRUCTURED-components, 10 MAXIMAL WELL-STRUCTURED-components, and 76 WHILE-components.

We have scored each component that was reduced in the SM algorithm with each of the three metrics to see how they differ. Since a component that is being reduced may have transitions that are the result of a reduction, it may be that SM will score that component a lot higher than it would had it not contained any of such transitions. Also, the other metrics do not take such issues into consideration, so to give a fair comparison we look at each component as if it does not contain transitions that are a result of a reduced component. This means that when we calculate SM for each component, we consider all transitions t to be annotated as $\tau(t) = 1$.

Table 1 shows the correlation coefficients for the different pairs of metrics. Note that the value "–" in the table is used when no correlation coefficient between the metrics could be calculated for a particular pair of metrics. The correlation coefficients could not be always be calculated because in some cases one of the metrics gave the same score for each component. This implies e.g. that it is not possible to calculate $\rho_{ECaM,ECyM}$ and $\rho_{ECaM,SM}$ for the choice

Table 1: Correlation coefficients of component scores.

Component type	$\rho_{ECaM,ECyM}$	$\rho_{ECaM,SM}$	$\rho_{ECyM,SM}$
CHOICE	–	–	1
MARKED-GRAPH	0.57	0.72	0.40
SEQUENCE	0.85	0.85	1
STATE-MACHINE	0.92	0.82	0.74
UNSTRUCTURED	0.34	0.58	0.11
WELL-STRUCTURED	0.88	0.15	0.32
WHILE	–	–	–

component.

Interestingly, the three pairs of metrics have in general a higher degree of correlation than they had when we considered scores for the whole nets. This indicates that they agree more on how to score components rather than whole WF-nets. Especially $ECyM, SM$ had almost no correlation, but looking at their correlation coefficients they tend to agree on their scoring of individual components. This is in part because SM penalizes embedded component explicitly by how it is defined, whereas $ECyM$ does not consider this aspect at all. A similar argument applies to the pair $ECaM$ and SM.

Our survey in this section allows us to conclude that the three metrics are very different. This can seen from the correlation coefficient absolute value for each pair of metrics, were below 0.32, and even as low as 0.02. It is not clear from the definitions of the three metrics that they would be so different, since it is possible to make small examples where their scores are very similar, as is backed up by our survey of how the correlation coefficient for each pair of metrics on each component scored in Table 1. If we only look at small examples that have this uniform behavior, we would have ended up by concluding that the three metrics are, although defined differently, highly correlated. But when, we look at complete models, with a high degree of behavioral patterns, the differences in the three metrics become evident.

To learn what impact it would have to change the penalties for each of the component we investigate at what *depth* the component is matched in the WF-net. The depth of a component is defined from the hierarchical component decomposition of the WF-net. Informally, we build the tree by running the algorithm in Definition 23 backwards, so that the top level component in the decomposition become the last reduced component. The children of each component C in the decomposition are the components that were folded and became transitions in C. We define the depth of the top-level component as 0, its children as 1, its children's children as 2, and so on. For an example, please refer to the tree view shown in Figure 11 and generated by our ProM plug-in. There the unstructured component has depth 0 and the two sequences has depth 1.

In Table 2 we show the match depth of each component type; the percentage

Table 2: Match depth of component types.

Depth	CHOICE	MARKED-GRAPH	SEQUENCE	STATE-MACHINE
0	0	21 (13.6%)	65 (3.4%)	103 (55.1%)
1	39 (16.7%)	91 (58.7%)	725 (37.5%)	52 (27.8%)
2	113 (48.3%)	24 (15.5%)	640 (33.1%)	26 (13.9%)
3	40 (17.1%)	17 (11.0%)	319 (16.5%)	4 (2.1%)
4	29 (12.4%)	2 (1.3%)	126 (6.5%)	2 (1.1%)
5	12 (5.1%)	0	44 (2.3%)	0
6	1 (0.4%)	0	12 (0.6%)	0
7	0	0	1 (0.05%)	0

Depth	UNSTRUCTURED	WELL-STRUCTURED	WHILE
0	63 (36.0%)	9 (90.0%)	1 (1.3%)
1	86 (49.1%)	1 (10.0%)	27 (35.5%)
2	21 (12.0%)	0	24 (31.6%)
3	4 (2.3%)	0	20 (26.3%)
4	1 (0.6%)	0	4 (5.3%)

after each number indicates how many times that component was matched at that depth compared to occurrences of it at other depths. The average match depth was: CHOICE 2.42; MARKED-GRAPH 1.28; SEQUENCE 1.95; STATE-MACHINE 0.66; UNSTRUCTURED 0.82; WELL-STRUCTURED 0.10; and, WHILE 1.99.

From the table we see that complex components such as MARKED-GRAPH, STATE-MACHINE, UNSTRUCTURED and WELL-STRUCTURED do not occur at such a low-level as, e.g., the SEQUENCE or CHOICE components. This makes sense because the later two have a higher priority according to ρ_{PN}.

What Table 2 doesn't show is the depth of each component compared to the maximal depth in the WF-net where it was found. However, the relative depth is interesting and, therefore, we introduce the *depth fraction function* in Definition 24 and a new metric in Definition 25.

Definition 24 (Depth fraction function). Let $PN = (P, T, F, \tau)$ be an annotated WF-net, *depth* be a function that calculates the depth of a component in a decomposition hierarchy, and *max_depth* be a function which calculates the maximal depth in a decomposition hierarchy. The depth fraction function $\delta_{PN} : dom(\rho_{PN}) \rightarrow [0; 1]$ is defined as follows. For $C \in dom(\rho_{PN})$:

$$\delta_{PN}(C) = \begin{cases} \frac{depth(C,PN)}{max\_depth(PN)} & \textbf{if } max\_depth(PN) > 0 \\ 0 & \textbf{if } max\_depth(PN) = 0 \end{cases}$$

Definition 25 (Aggregated depth fraction metric). Let S be a set of annotated WF-nets, $PN \in S$. The aggregated depth fraction metric $\Delta_S : rng(\rho_{PN}) \rightarrow$

$[0;1]$ maps a component type (according to the 7 classes in Definition 21) onto its average relative depth and is defined as follows. For $\alpha_C \in rng(\rho_{PN})$:

$$\Delta_S(\alpha_C) = \frac{\sum_{PN \in S} \left(\sum_{C \in [PN]: \rho_{PN}(C) = \alpha_C} \delta_{PN}(C) \right)}{\sum_{PN \in S} |\{C \in [PN] | \rho_{PN}(C) = \alpha_C\}|}$$

Intuitively Δ_S score close to 0 if the components with the component type α_C found in S all are found close to the top-level of their respective WF-net component decomposition hierarchy. Conversely, it score close to 1 if they are found close to the bottom-level.

We applied the Δ_S metric to the 262 nets and the results were: CHOICE 0.63; MARKED-GRAPH 0.41; SEQUENCE 0.64; STATE-MACHINE 0.23; UNSTRUCTURED 0.27; WELL-STRUCTURED 0.05; and, WHILE 0.59. This shows that CHOICE-, SEQUENCE-, and WHILE-components are more likely to be found at the bottom of each of the WF-net component decomposition hierarchy, whereas the rest of the component types are more likely to be found at the top-level.

If we take this knowledge about the depth where we find the various component types we can more accurately fine-tune the component weight function. By also observing that the metric penalty score is accumulated at each level of a component decomposition tree, changing the penalties of component types that are more likely to be found at the lower levels will have a greater effect on the total score of *SM* than changing those more likely to be found at the top-level.

In the survey we saw that nets where *ECaM* scored high, were as expected, nets with high fan-outs from places, but the nets themselves were not behaviorally complex. This leads us to believe that *ECaM* is not suited for scoring nets that contain many behavioral patterns, and where these are embedded in each other.

ECyM did in many ways, as for *ECaM*, not really reflect the complexity of the actual net. Its intent is to measure the behavioral complexity of the reachability graph, and not the actual net. However, we found that although *ECyM* actually scored very similar to *ECaM* and *SM* on small examples and on the components found in the *SM* algorithm, the way it scored a net in general did not truly reflect the complexity and effort needed to understand a net. Based on our empirical evaluation, we conclude that *ECyM* does not give us any insight in how complicated real-world process models are. It does, however, give us an understanding of how complicated the underlying behavior of the net is, and this makes *ECyM* a valuable supplement to other metrics such as *ECaM* and *SM* that do not take the reachability graph into account.

We, the authors, found that *SM* was the best metric. Nets that had a low *SM* value were easy to understand, and the nets that scored high were very difficult. We observed that nets that SM gave a higher score, had many levels of components embedded in each other and/or uneven number of splits and joins.

6. Related Work

As discussed in the introduction, it is assumed that simpler models are preferable over more complex models. Moreover, if models become too complex, they should be split into simpler models. While this seems obvious, little evidence can be found in literature. A notable exception is the empirical work presented in [27, 28]. Using a collection of 2003 EPC models it is shown that at least 10 percent of these models is flawed. In fact, for certain subsets (e.g., the SAP reference model consisting of 604 EPCs) it is shown that more than 20 percent have errors such as deadlocks, livelocks, etc. [27]. So people make errors and the complexity of these models is a good predictor. Using a simple regression model it is shown that 7 variables (including coefficient of connectivity, connector mismatch, cyclicity, separability, diameter) can be used to predict whether a model is flawed or not. In 95 percent of the cases such a prediction is correct. This shows that variables related to the complexity of the model are indeed predictors for errors. This supports our initial assumption.

The study reported in [27, 28, 29] also shows that none of the existing complexity metrics is able to adequately capture complexity in two ways: they define metrics by looking at local syntactical complexity, with the exception of the structuredness metric[7], and the interpolation of the seven metrics as a single metric are difficult to understand. We believe that by identifying behavioral patterns in a process and scoring these according to their type and elements, we get a metric that is better at scoring complexity. Also, our emphasis on penalizing components that are embedded in others give a more true reflection on the complexity of a process. Another benefit of only using a single metric as we would use *SM* is that for the end-user it will be much clearer why a net scored as it did. With the tool support that we described in Section 4 we were able to achieve this.

The Cardoso metric which inspired us to the definition of ECaM is described in [13]. The Cardoso metric is in many way simple to understand and argue why it is sensibly defined, if we look at small examples. But as we showed in our case study in Section 5 for large models with many different kinds of behavioral patterns, it does not come close to truly measuring the complexity of the nets.

Vanderfeesten et al. [37] define the Cross-Connectivity metric (CC) as a measure for the complexity of a process model. Their starting point is a process model with an explicit representation of splits and joins (similar to EPCs) of the types AND, OR, XOR. From there they define a value for a path which is calculated from the nodes on that path. Now, CC is calculated by summing values of all possible paths in the process graph and dividing it with the number

[7] The structuredness metric defined in [27] is a very simplistic version of *SM* and has not yet been formally defined. It relates to the essential Cyclomatic metric [26], and it measures how well the process can be decomposed into well-defined components. It does not, as we do, consider different types of components, and score them different accordingly. The structuredness metric is then $\Phi_N = 1 - \frac{|N'|}{|N|}$; N is the nodes in the original graph and N' the nodes in the graph after all possible reductions are applied.

of possible connections; $CC = \frac{\sum_{n_1,n_2 \in N} V(n_1,n_2)}{|N| \cdot (|N|-1)}$. CC is similar to the Cardoso metric in the sense that it focuses on the syntax of the process, although it is not as simple in that CC takes entire paths into considerations. Although CC tries to consider a larger portion of the process model when evaluating a model, it still doesn't consider complete behavioral structures as we do with SM.

Much empirical work has been done by Mendling et al. [31, 30], to learn what makes a model understandable. They operationalize understandability by introducing three categories of factors that they feel are important in understanding a model: personal (beyond psychological and intellectual); structural (model characteristics); and textual (description in the model). Besides characterizing understandability they do a web survey to test a number of hypothesis on the three categories of understandability. Among their findings they saw that higher knowledge of theory of concurrency and daily work with models lead to better understanding of models. Also, that the larger the score the participants of the web survey got wrt. a particular model was positively correlated with the structuredness and soundness of the model, regardless of their prior knowledge of the theory of concurrency. Their experiments show that there is a connection between the degree of structuredness in a process model and the understandability of it, and thereby also to lower complexity of the process model.

Muehlen and Recker [38] investigate how many of the BPMN language constructs is actually used in real-world models. Interestingly they find that the most used constructs is from the core set of BPMN. The core set of BPMN has constructs for sequencing tasks, splitting them in flows or choices. The authors argue that people often don't extend their usage of constructs besides the core elements because they are confused of how they should be used. In comparison to our work we also see that the majority of components we found in all of the process models were of the predefined types and that only 6.1% were actually unstructured.

7. Conclusion

Process models play an important role in the analysis of business processes and their implementation. The correctness and quality of these models are vital for any organization. A workflow management system configured on the basis of an incorrect model may lead to costly problems. Analysis results obtained using e.g. simulation are only reliable if the models are an adequate reflection of the perceived or intended reality. Sadly, both process engineers and end-users, have problems understanding models. Therefore, it is important to carefully assess and measure the complexity of process models. Contemporary metrics have problems adequately capturing this complexity. Therefore, we proposed a new metric based on the structuredness of process models. Highly structured process models based on simple design patterns are classified as "easy" while spaghetti-like process models using advanced patterns are classified as "difficult".

We have implemented different complexity metrics using ProM. The new metric based on structuredness is completely configurable and can be fine-tuned to specific application domains or languages. To test this metric we have evaluated 262 process models and compared the different metrics. This study suggests that the new structuredness outperforms existing ones.

We believe that *SM* is a true alternative to already known metrics and in many ways supersedes them in the way that *SM* consider the process as a whole, and that *SM* is defined in such a way that it is easy to explain to people and make them understand why a process scores the way it did – an aspect the other metrics mentioned in the related work in Section 6 all struggle with.

Although we have tested the metrics on hundreds of models, notions of complexity are subjective and the actual weights of the different constructs used by *SM* in this paper are just an initial proposal. Therefore, we suggest to a field study as proposed in other papers [30] where the metrics are compared with the (subjective) opinions of experts and end-users.

References

[1] W. M. P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.

[2] W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.

[3] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[4] W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 161–183. Springer-Verlag, Berlin, 2000.

[5] W.M.P. van der Aalst. Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 1–65. Springer-Verlag, Berlin, 2004.

[6] W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.

[7] W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. ProM 4.0: Comprehensive Support for Real Process Analysis. In J. Kleijn and A. Yakovlev, editors, *Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007)*, volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer-Verlag, Berlin, 2007.

[8] W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of Workflow Nets: Classification, Decidability, and Analysis. BPM Center Report BPM-08-02, BPMcenter.org, 2008.

[9] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns Home Page. http://www.workflowpatterns.com.

[10] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[11] W.M.P. van der Aalst and K.B. Lassen. Translating Unstructured Workflow Processes to Readable BPEL: Theory and Implementation. *Information and Software Technology*, 2006.

[12] W.M.P. van der Aalst, K.M. van Hee, and R.A. van der Toorn. Component-Based Software Architectures: A Framework Based on Inheritance of Behavior. *Science of Computer Programming*, 42(2-3):129–171, 2002.

[13] J. Cardoso. Control-flow Complexity Measurement of Processes and Weyuker's Properties. In *Transactions on Enformatika, Systems Sciences and Engineering*, volume 8, pages 213–218. Springer Verlag, Berlin, Budapest, Hungary, 6 edition, October 2005.

[14] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[15] J. Dehnert. *A Methodology for Workflow Modeling: From Business Process Modeling Towards Sound Workflow Specification*. PhD thesis, TU Berlin, Berlin, Germany, 2003.

[16] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.

[17] R. Eshuis and J. Dehnert. Reactive Petri nets for Workflow Modeling. In W.M.P. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 295–314. Springer-Verlag, Berlin, 2003.

[18] R.J. van Glabbeek and D.G. Stork. Query Nets: Interacting Workflow Modules that Ensure Global Termination. In W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, editors, *International Conference on Business Process Management (BPM 2003)*, volume 2678 of *Lecture Notes in Computer Science*, pages 184–199. Springer-Verlag, Berlin, 2003.

[19] K. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In W.M.P. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 335–354. Springer-Verlag, Berlin, 2003.

[20] G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley, Reading MA, 1998.

[21] B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2003. Available via http://www.workflowpatterns.com.

[22] B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of Control Flow in Workflows. *Acta Informatica*, 39(3):143–209, 2003.

[23] E. Kindler and W.M.P. van der Aalst. Liveness, Fairness, and Recurrence. *Information Processing Letters*, 70(6):269–274, June 1999.

[24] E. Kindler, A. Martens, and W. Reisig. Inter-Operability of Workflow Applications: Local Criteria for Global Soundness. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 235–253. Springer-Verlag, Berlin, 2000.

[25] A. Martens. On Compatibility of Web Services. *Petri Net Newsletter*, 65:12–20, 2003.

[26] T. McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering*, 2:308–320, 1976.

[27] J. Mendling. *Detection and Prediction of Errors in EPC Business Process Models*. PhD thesis, Vienna University of Economics and Business Administration, Vienna, Austria, 2007.

[28] J. Mendling, M. Moser, G. Neumann, H.M.W. Verbeek, B.F. van Dongen, and W.M.P. van der Aalst. Faulty EPCs in the SAP Reference Model. In S. Dustdar, J.L. Faideiro, and A. Sheth, editors, *International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 451–457. Springer-Verlag, Berlin, 2006.

[29] J. Mendling, G. Neumann, and W.M.P. van der Aalst. Understanding the Occurrence of Errors in Process Models Based on Metrics. In F. Curbera, F. Leymann, and M. Weske, editors, *Proceedings of the OTM Conference on Cooperative information Systems (CoopIS 2007)*, volume 4803 of *Lecture Notes in Computer Science*, pages 113–130. Springer-Verlag, Berlin, 2007.

[30] J. Mendling, H.A. Reijers, and J.Cardoso. What Makes Process Models Understandable? In *Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings*, volume 4714 of *Lecture Notes in Computer Science*, pages 48–63. Springer-Verlag, Berlin, 2007.

[31] J. Mendling and M. Strembeck. Influence Factors of Understanding Business Process Models. In Witold Abramowicz and Dieter Fensel, editors, *BIS*, volume 7 of *Lecture Notes in Business Information Processing*, pages 142–153. Springer, 2008.

[32] R. Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, Cambridge, UK, 1999.

[33] M. Nüttgens. Event-driven Process Chain (EPC) / Ereignisgesteuerte Prozesskette (EPK). http://www.iwi.uni-sb.de/nuettgens/EPK/epk.htm.

[34] ProM. ProM, 2006. `http://prom.sourceforge.net`.

[35] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.

[36] R. van der Toorn. *Component-Based Software Design with Petri nets: An Approach Based on Inheritance of Behavior*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2004.

[37] I.T.P. Vanderfeesten, H. Reijers, J. Mendling, W.M.P. van der Aalst, and J. Cardoso. On a Quest for Good Process Models: The Cross-Connectivity Metric. In Z. Bellahsene and M. Léonard, editors, *Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE 2008)*, volume 5074 of *Lecture Notes in Computer Science*, pages 480–494. Springer-Verlag, Berlin, 2008.

[38] M. zur Muehlen and J. Recker. How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In Z. Bellahsene and M. Léonard, editors, *Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE 2008)*, volume 5074 of *Lecture Notes in Computer Science*, pages 465–479. Springer-Verlag, Berlin, 2008.