Verification of Structured Processes: A Method Based on an Unsoundness Profile

Jorge Roa[1], Omar Chiotti[2], and Pablo Villarreal[1]

[1] CIDISI, Universidad Tecnológica Nacional - Facultad Regional Santa Fe, Santa Fe, Argentina
{jroa, pvillarr}@frsf.utn.edu.ar
[2] INGAR-CONICET, Santa Fe, Argentina
chiotti@santafe-conicet.gov.ar

Abstract.
The verification of business processes has been widely studied in the last two decades achieving significant results. Despite this, existing verification techniques based on state space exploration suffer, for large processes, the state space explosion problem. New techniques improved verification performance by structuring processes as trees. However, they do not support complex constructs for advanced synchronization and exception management. To cope with this issue we propose the definition of an unsoundness profile of a given process language, which specifies all possible combinations of control flow constructs that can lead to errors in the behavior of structured processes defined with such a language. In addition, we introduce the sequential and hierarchical soundness properties, which make use of this profile to determine soundness of a structured process with complex constructs in polynomial time. As an example, we defined an unsoundness profile for a subset of the BPMN language and verified the behavior of a BPMN process model.

Keywords: Business Process, Verification, Soundness, Correctness Properties

1 Introduction

Business process modeling (BPM) emerged as a means to control, analyze, and optimize business operations [11]. The verification of the behavior of business processes is an important requirement for BPM, since they are used to deliver value-added products and services to clients. In the last two decades several methods have been proposed to cope with this issue achieving significant results [1].

Business processes can be modeled with high level languages such as BPMN [6]. However, their analysis requires the use of formal languages such as Petri nets [8] and properties like *soundness* [2]. A business process is sound if it is free of *deadlocks* and *lack of synchronizations* in its control flow.

The performance of the analysis of these properties is an important requirement to consider in verification methods. Workflow nets [8] (WF-Nets) are a special type of Petri nets which have been used to formalize the behavior of processes and determine their soundness. However, they may suffer the state space explosion problem [12], having a negative impact on performance. In addition, they do not support complex constructs for advanced synchronization, cancellation, and exception management.

It has been proved that the structure of a process impacts on verification performance [3,4,9]. A structured process have a special topology where each split/decision is associated with a corresponding join/merge such that the set of nodes between each split/join and decision/merge defines a single-entry/single-exit (SESE) fragment [3]. However, although verification of structured processes can be performed in linear time [1], complex constructs are also not supported.

To cope with this issue we propose a verification method for structured processes which supports complex constructs for advanced synchronization and exception management. The method is based on an unsoundness profile of a given process language, and two behavioral properties called sequential and hierarchical soundness. An unsoundness profile specifies all possible minimal combinations of control flow constructs that can lead to errors in the behavior of structured processes defined with a given language. This is performed at a language level, and each of these combinations define what we call a minimal process. Sequential and hierarchical soundness properties make use of this profile to determine soundness of a structured process model in polynomial time. The use of minimal processes leads to the exact combination of elements which can be the source of an error. As an example, we defined an unsoundness profile for a subset of BPMN and verified the behavior of a BPMN process model with advanced synchronization, cancellation, exception management, and loops.

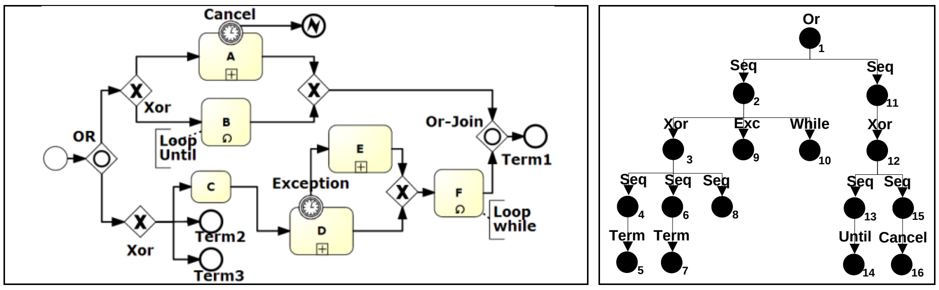
This work is structured as follows. Section 2 presents the structure and behavior of block-structured processes. Section 3 introduces the minimal behavioral decomposition and presents correctness criteria for structured processes. Section 4 presents the verification method. Section 5 establishes a discussion. Finally, Section 6 presents conclusions and future work.

2 Block-Structured Business Processes

In this section, we study both the structure and behavior of block-structured business processes and introduce the concept of minimal process.

2.1 Structure of Block-Structured Processes

A *block-structured process* is a business process with a special topology where each split/decision is associated with a corresponding join/merge such that the set of nodes between each split/join and decision/merge defines a single-entry/single-exit (SESE) fragment [3]. In a block-structured process each control flow element can have an *opening* and a *closing behavioral semantics*, which correspond to the split/join (for concurrency) or decision/merge (for mutual exclusion) of a control flow element. We consider that a block-structured process language has three primitive constructs $Activity$, $Sequence$, and $Termination$. Other constructs for concurrency, mutual exclusion, loops, exception, etc. can be represented in a block-based manner by combining primitive constructs. Since a business process model is always composed of instances of constructs, in this work we use the term *process element* or just *element* to refer to an instance of a construct in a process model. From now on, we refer to block-structured processes and structured processes indistinctly.



(a) Block-structured business process *P* (b) Process *P* structured as a tree

Fig. 1. A BPMN process model and its block-structured representation

Figure 1 shows an example of a structured process model. The BPMN process *P* (Figure 1a) starts with an *Or* gateway having two sequence flows. The first sequence flow is followed by an *Xor* where the execution of subprocess *A* and activity *B* is mutually exclusive. Activity *B* will be executed in a loop until a given condition is reached, whereas subprocess *A* is associated with a timer event. If the timer occurs an exception is raised and the process finishes with an error. The second sequence flow outgoing from the element *Or* is composed of an *Xor* where there are three mutually exclusive sequence flows. Two of them finishes the process (*Term2* and *Term3*). The other one executes activity *C*. After *C*, subprocess *D* can be executed. If *D* takes more time than expected, an exception is raised and subprocess *E* is invoked. Both subprocesses *D* and *E* are followed by activity *F*, which will be executed in a loop while a given condition holds. *Term1* represents the end of the process.

A structured process can be graphically represented as a tree. Figure 1 b) shows the tree-structured representation of process P. Each node has a unique identifier specified as a subscript, and represents both the opening and closing behavioral semantics, e.g. node 1 represents elements *Or* and *Or-Join* of the BPMN process of Figure 1 a).

Figure 2 shows structural aspects of structured processes. For each node of the tree there is a unique *tree path* which connects the node with the root of the tree. Figure 2 a) shows four tree paths of process *P*. Given two elements e_1, e_n, we say e_n is *structurally reachable* from e_1 if there is a tree path where e_1 precedes e_n. Since this work focuses on the control flow of processes, constructs such as $Activity$ and the initial and final events, are omitted in the block-structured graphical representation. Sequences in leaf nodes are also omitted whenever possible.

In a structured process, elements are combined in a parent child/relationship, where there is a set of elements C (the children) which are directly connected (nested) to an element e (the parent). The combination can be sequential or hierarchical. In a *sequential combination*, e is a sequence and C is an ordered set of process elements different from sequence, e.g. in Figure 2 b), there is a sequential combination of elements, where $e = Seq_1$ and $C = \{Xor_2, Exc_3, While_4\}$.

In a *hierarchical combination*, both the parent and elements from C must not be a sequence. Figure 2 e) shows an example where, $e = Xor_1$ and $C = \{Loop_3, Term_5\}$. Since constructs are always composed of sequences, in order to define a hierarchical combination it is always necessary to make use of elements *Sequence*. In this example, element Xor_1 together with sequences Seq_2, Seq_4 and Seq_6 are considered as a unique construct (an *Xor* with three sequence flows). Since such sequences are part of Xor_1, they are not considered in C.

A control flow element can be minimal or non-minimal. Given a construct C and a control flow element e which is instance of C, e is a *minimal process element* if it complies exactly with the minimal metamodel constraints necessary to generate an instance of C, e.g. the construct *And* is usually restricted to have at least two concurrent sequences. Hence, an *And* with two sequences is a minimal element (Figure 2 c)).

Minimal process elements are of particular interest in this work, since they can be used to define minimal processes. P_M is a *minimal sequential process* if it is composed only of a sequential combination of two control flow elements e_1 and e_2, written as $P_M = \{r, e_1, e_2\}$, where r is the root of P_M. Figure 2 d) shows a minimal sequential process, where $r = Seq_1, e_1 = Xor_2$ and $e_2 = While_3$. P_M is a *minimal hierarchical process* if it is composed only of a hierarchical combination of three control flow elements r, e_1, and e_2, written as $P_M = \{r, e_1, e_2\}$, or a combination of two elements, written as $P_M = \{r, e_1\}$, where r is the root of P_M. For example, Figures 2 f) and g) show two minimal hierarchical processes. The former is composed of three control flow elements $P_M = \{And_1, Xor_3, Until_5\}$, whereas the latter is composed of two elements $P_M = \{Until_1, And_3\}$.

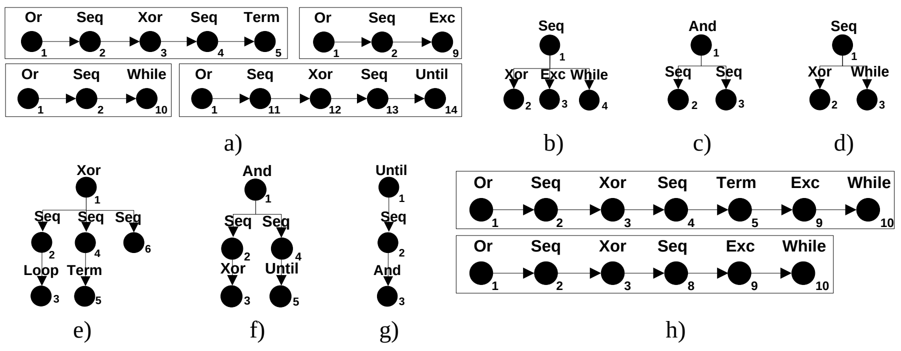


Fig. 2. Structural aspects of structured processes

A structured process is a *minimal process* if it is composed only of a minimal process element, or if it is either a minimal sequential process or a minimal hierarchical process. Otherwise, it is a *non-minimal process*. Processes in Figures 2 c), d), f), and g) are minimal, whereas those in Figures 2 b) and e) are non-minimal.

2.2 Behavior of Block-Structured Processes

The behavior of a business process is usually defined by the states of its activities [5]. However, in this work, we focus on the execution state of control flow elements rather than activities. During the execution of a business process, each control flow element can be on a specified *execution state*: enabled, disabled, in execution, completed, and canceled. The execution state of a process is defined by the set of all execution states of its control flow elements.

A control flow element is considered to be in execution until all activities and control flow elements within its scope finish their execution. The state "in execution" is of particular interest to analyze the behavior of processes. We distinguish between a foreground and a background execution. A control flow element e is in *foreground execution*, if e is in execution and there is no other control flow element structurally reachable from e in execution. A control flow element e is in *background execution*, if there is at least one element structurally reachable from e in execution.

A structured process is composed of a finite set of execution paths which determine each possible execution of the process. Execution paths are key to analyze the behavior of structured processes. The process P shown in Figure 1 b) has five execution paths. Two of them are shown in Figure 2 h). The rest of them are: (1) $Or_1 \longmapsto Seq_2 \longmapsto Xor_3 \longmapsto Seq_6 \longmapsto Term_7 \longmapsto Exc_9 \longmapsto While_{10}$; (2) $Or_1 \longmapsto Seq_{11} \longmapsto Xor_{12} \longmapsto Seq_{13} \longmapsto Until_{14}$; and (3) $Or_1 \longmapsto Seq_{11} \longmapsto Xor_{12} \longmapsto Seq_{15} \longmapsto Cancel_{16}$.

Given a structured process BP and two different process elements e_0, e_n of BP. (1) e_n is *behaviorally reachable* from e_0, if there exists an execution path starting in e_0 and finishing in e_n, e.g. in process P, the element $While_{10}$ is behaviorally reachable from Xor_3, whereas $Until_{14}$ is unreachable from Xor_3 (see Figures 1 b) and 2 h)); (2) e_n is called *executable* from e_0, if there exists an execution path where e_n is behaviorally reachable from e_0 and a pair of two different execution states es_0, es_n, where state es_n is reachable from state es_0, such that element e_0 is in foreground execution in state es_0 and element e_n is in foreground execution in state es_n.

An important aspect of structured processes is that the behavior of each process element depends only on its predecessors. Hence, elements which are not part of the same execution path are independent from each other, e.g. if in a given execution state, elements $While_{10}$ and Xor_{12} are both in execution, $While_{10}$ cannot affect the behavior of Xor_{12} (and vice versa). This will be a key concept in the next section.

3 Correctness Properties of Structured Processes

In this section, we present the behavioral decomposition of structured processes, and the correctness properties used to determine their soundness.

3.1 Correctness of the Behavior of Block-Structured Business Processes

Soundness [2] is a well known property which was originally proposed for the verification of the control flow of Workflow Systems. In this work, we abstract from both the formal grounds of this property and the formal languages that support it, and map these concepts directly to structured processes.

Definition 1. A block-structured process BP is *sound* iff it has neither deadlocks nor lack of synchronizations in its control flow. A *deadlock* occurs if the execution of BP reaches a state where there is a set of sequences which cannot be synchronized/merged, or if there is a sequence which cannot finish its execution. *Lack of synchronization* occurs if there are two process elements e_1 and e_2, where e_2 is executable from e_1, and there is an execution state es_n where both elements e_1 and e_2 are in foreground execution in state es_n.

3.2 Behavioral Relationship between Minimal and Non-Minimal Processes

To see the relationship between minimal and non-minimal processes we show some examples based on process P presented in Section 2. Suppose that elements $Term_5$ and $Until_{14}$ are in execution. In this state, although not directly connected, both elements $Term_5$ and $Until_{14}$ are ruled by the behavioral semantics of Or_1, and hence, they execute concurrently. In general, two elements of a structured process which are not part of the same execution path, are related to each other by means of the behavioral semantics of a predecessor common to both elements. It is important to recall from Section 2.2 that the behavior of each process element of a structured process depends only on its predecessors.

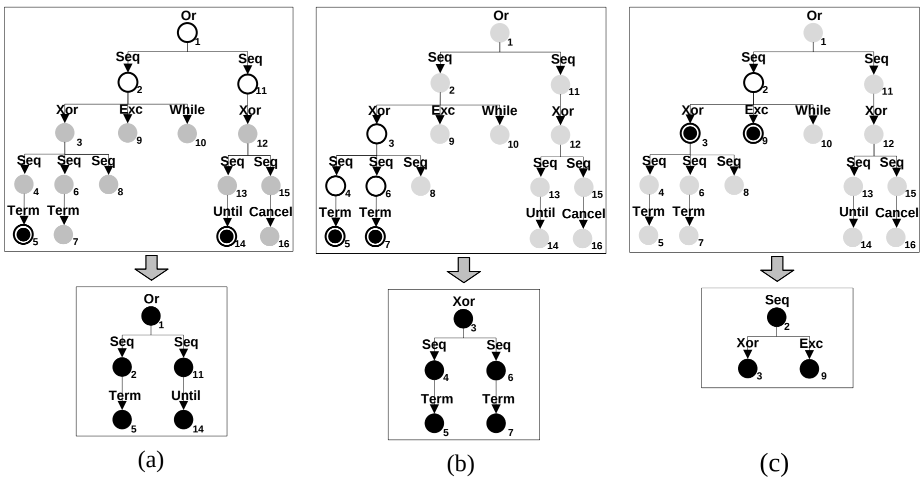


Fig. 3. Relationship between non-minimal and minimal structured processes

This relation between control flow elements and their predecessors enables the generation of a set of minimal processes from the process P. The upper section of Figure 3 shows different execution states of process P from which it is possible to generate the minimal processes shown in the lower section. White nodes represent the root of a minimal process, whereas black nodes represent nested elements of a minimal process, e.g. from the aforementioned execution state, it is possible to define the minimal process $P_{M_A} = \{Or_1, Term_5, Until_{14}\}$ (Figure 3 a)), where the root of the process is Or_1, which is composed of the elements $Term_5$ and Xor_{14}. Grey nodes are ignored. Similarly, we can define other minimal processes such as $P_{M_B} = \{Or_1, Term_5, Xor_{12}\}$, $P_{M_C} = \{Or_1, Seq_2, Cancel_{16}\}$, $P_{M_D} = \{Or_1, Exc_9, Until_{14}\}$, and so on. All these minimal processes have in common their root (Or_1) and that they were generated from hierarchical combinations of control flow elements. We can also consider hierarchical combinations with other root, e.g. $P_{M_E} = \{Xor_3, Term_5, Term_7\}$ shown in Figure 3 b). Besides hierarchical combinations, it is also possible to generate minimal processes from sequential combinations, such as $P_{M_F} = \{Seq_2, Xor_3, Exc_9\}$ shown in Figure 3 c).

The importance of decomposing process P into minimal processes is that they can be used to analyze the behavior of P. See that in this example, process P is not sound, since element $Term_5$ disables the synchronization of element Or_1, causing a deadlock when elements $Term_5$ and $Until_{14}$ are in foreground execution. However, this situation can also be determined by analyzing the execution states of the minimal process P_{M_A}, since such states are a subset of those of the process P. Hence, for this example, it would be sufficient to verify the behavior of P_{M_A} to determine the unsoundness of P, which its makes much simpler and accurate. However, process P is composed of other unsound minimal processes which do not imply unsoundness of P, e.g. although $P_{M_E} = \{Xor_3, Term_5, Term_7\}$ is not sound, such minimal process is not sufficient to guaranty unsoundness of P. In the remainder of this section, we show the conditions that minimal processes must satisfy to determine soundness of a structured process.

Algorithm 1. Code to generate the minimal behavioral decomposition.

```
generateMBD(root)
children ⟵ getChildren(root); i ⟵ 0 ; mbd ⟵ ∅
for all child such that child ∈ children do
  pivots ⟵ getChildren(children, 0, i+1)
  nodes ⟵ getChildren(children, i+1, |children|)
  if (getOpeningSemantics(root)=SEQUENCE) then
    mdb ⟵ mdb ∪ generateMSD(root, pivots, nodes)
  else if (not getOpeningSemantics(root)=TERMINATION) then
    mdb ⟵ mdb ∪ generateMHD(root, pivots, nodes)
  end if
  pivot ⟵ child; i ⟵ i + 1
  generateMBD(pivot)
end
```

3.3 Minimal Behavioral Decomposition of Block-Structured Processes

The decomposition of a structured process into minimal processes is called the *minimal behavioral decomposition* and is shown in Algorithm 1, which is split into a sequential and a hierarchical decomposition. Depending on the type of control flow element, functions `generateMSD()` (for a sequential decomposition) and `generateMHD()` (for a hierarchical decomposition) are invoked. Algorithm 2 and 3, shows the code to generate the *minimal sequential decomposition* and the *minimal hierarchical decomposition*, which implies the generation of all possible minimal sequential and hierarchical processes of a structured process respectively. Finally, Algorithm 4, shows the code to generate each minimal hierarchical process.

Algorithm 2. Code to generate the minimal sequential decomposition.

```
generateMSD(root, pivots, nodes)
msd ⟵ ∅
for all pivot such that pivot ∈ pivots do
  for all n such that n ∈ nodes do
    mProc ⟵ mProc ∪ {{root,pivot,n}}
  end
end
return msd
```

Algorithm 3. Code to generate the minimal hierarchical decomposition.

```
generateMHD(root, pivots, nodes)
mhd ⟵ ∅
for all pivot such that pivot ∈ pivots do
  mhd ⟵ mhd ∪ addMHP(root,pivot,nodes)
  children ⟵ getChildren(pivot)
  mhd ⟵ mhd ∪ generateMHD(root,children,nodes)
end
return msd
```

Algorithm 4. Code to generate minimal hierarchical processes.

```
addMHP(root,pivot,nodes)
mHProc ⟵ ∅
if (nodes=∅ and (getOpeningSemantics(root)=LOOP_UNTIL
              or getOpeningSemantics(root)=LOOP_WHILE))
  mHProc ⟵ mHProc ∪ {{root,pivot}}
else
  for all n such that n ∈ nodes do
    mHProc ⟵ mHProc ∪ {{root,pivot,n}}
    mHProc ⟵ mHProc ∪ addMHP(root,pivot, getChildren(n))
  end
end if
return mHProc
```

Table 1. Minimal sequential and hierarchical decompositions of the process *P*

Minimal Hierarchical Decomposition		
$P_{M_1}^{+} = \{Or_1, Seq_2, Seq_{11}\}$	$P_{M_2}^{+} = \{Or_1, Seq_2, Xor_{12}\}$	$P_{M_3}^{+} = \{Or_1, Seq_2, Until_{14}\}$
$P_{M_4}^{-} = \{Or_1, Seq_2, Cancel_{16}\}$	$P_{M_5}^{+} = \{Or_1, Xor_3, Seq_{11}\}$	$P_{M_6}^{+} = \{Or_1, Xor_3, Xor_{12}\}$
$P_{M_7}^{+} = \{Or_1, Xor_3, Until_{14}\}$	$P_{M_8}^{-} = \{Or_1, Xor_3, Cancel_{16}\}$	$P_{M_9}^{-} = \{Or_1, Term_5, Seq_{11}\}$
$P_{M_{10}}^{-} = \{Or_1, Term_5, Xor_{12}\}$	$P_{M_{11}}^{-} = \{Or_1, Term_5, Until_{14}\}$	$P_{M_{12}}^{-} = \{Or_1, Term_5, Cancel_{16}\}$
$P_{M_{13}}^{+} = \{Or_1, Exc_9, Seq_{11}\}$	$P_{M_{14}}^{+} = \{Or_1, Exc_9, Xor_{12}\}$	$P_{M_{15}}^{+} = \{Or_1, Exc_9, Until_{14}\}$
$P_{M_{16}}^{-} = \{Or_1, Exc_9, Cancel_{16}\}$	$P_{M_{17}}^{+} = \{Or_1, While_{10}, Seq_{11}\}$	$P_{M_{18}}^{+} = \{Or_1, While_{10}, Xor_{12}\}$
$P_{M_{19}}^{+} = \{Or_1, While_{10}, Until_{14}\}$	$P_{M_{20}}^{-} = \{Or_1, While_{10}, Cancel_{16}\}$	$P_{M_{21}}^{-} = \{Xor_3, Term_5, Term_7\}$
$P_{M_{22}}^{+} = \{Xor_3, Term_5, Seq_8\}$	$P_{M_{23}}^{+} = \{Xor_{12}, Until_{14}, Cancel_{16}\}$	
Minimal Sequential Decomposition		
$P_{M_{24}}^{+} = \{Seq_2, Xor_3, Exc_9\}$	$P_{M_{25}}^{+} = \{Seq_2, Exc_9, While_{10}\}$	

Table 2. Minimal behavioral decomposition of the process *P*

Execution path	Minimal processes
1- $Or_1, Seq_2, Xor_3, Seq_4, Term_5, Exc_9, While_{10}$	$P_{M_1} P_{M_2} P_{M_3} P_{M_4} P_{M_5} P_{M_6} P_{M_7} P_{M_8} P_{M_9} P_{M_{10}} P_{M_{11}} P_{M_{12}} P_{M_{13}} P_{M_{14}} P_{M_{15}} P_{M_{16}} P_{M_{17}} P_{M_{18}} P_{M_{19}} P_{M_{20}} P_{M_{21}} P_{M_{22}} P_{M_{24}} P_{M_{25}}$
2- $Or_1, Seq_2, Xor_3, Seq_6, Term_7, Exc_9, While_{10}$	$P_{M_1} P_{M_2} P_{M_3} P_{M_4} P_{M_5} P_{M_7} P_{M_8} P_{M_{13}} P_{M_{14}} P_{M_{15}} P_{M_{16}} P_{M_{17}} P_{M_{18}} P_{M_{19}} P_{M_{20}} P_{M_{21}} P_{M_{22}} P_{M_{24}} P_{M_{25}}$
3- $Or_1, Seq_2, Xor_3, Seq_8, Exc_9, While_{10}$	$P_{M_1} P_{M_2} P_{M_3} P_{M_4} P_{M_5} P_{M_6} P_{M_7} P_{M_8} P_{M_{13}} P_{M_{14}} P_{M_{15}} P_{M_{16}} P_{M_{17}} P_{M_{18}} P_{M_{19}} P_{M_{20}} P_{M_{21}} P_{M_{22}} P_{M_{24}} P_{M_{25}}$
4- $Or_1, Seq_{11}, Xor_{12}, Seq_{13}, Until_{14}$	$P_{M_1} P_{M_2} P_{M_3} P_{M_4} P_{M_5} P_{M_6} P_{M_7} P_{M_9} P_{M_{10}} P_{M_{11}} P_{M_{12}} P_{M_{13}} P_{M_{14}} P_{M_{15}} P_{M_{17}} P_{M_{18}} P_{M_{19}} P_{M_{23}}$
5- $Or_1, Seq_{11}, Xor_{12}, Seq_{15}, Cancel_{16}$	$P_{M_1} P_{M_2} P_{M_4} P_{M_5} P_{M_6} P_{M_8} P_{M_9} P_{M_{10}} P_{M_{12}} P_{M_{13}} P_{M_{14}} P_{M_{16}} P_{M_{17}} P_{M_{18}} P_{M_{20}} P_{M_{23}}$

Algorithm 1 enables the generation of all possible minimal processes which can be defined from a given structured process in polynomial time. Function `gener-ateMBD(root)` returns a set of sets, where each set is a minimal process. Table 1 shows the minimal sequential and hierarchical decompositions of the process *P* returned by Algorithm 1. Equivalent minimal processes are not shown, e.g. the minimal process $P_M = \{Or_1, Seq_4, Seq_{11}\}$ is behaviorally equivalent to P_{M_1}. An implementation of these algorithms can be found at https://code.google.com/p/minimal-behavioral-decomposition.

The *minimal behavioral decomposition* is composed of all minimal processes generated from both the minimal sequential and hierarchical decompositions. It is a set of sets, where minimal processes are grouped according to their associated execution paths, such that for each execution path there is a set of minimal processes, where each minimal process of this set has at least one element different from the root which is part of its associated execution path.

Table 2 shows the minimal behavioral decomposition of process P. Minimal processes (right column) are grouped according to their related execution paths (left column). Each minimal process on the right column is composed of at least one process element of the execution path indicated in the left column, e.g. minimal process $P_{M_{23}}$ is composed of elements $Until_{14}$ and $Cancel_{16}$, and hence, it is related to execution paths 4 and 5. Similarly, P_{M_1} is related to executions paths 1 to 5, since it is composed of elements Seq_2 and Seq_{11} which are part of all execution paths.

3.4 Correctness Properties for Block-Structured Business Processes

We split correctness properties according to the type of combination of control flow elements: sequential or hierarchical.

Definition 2. A structured process is *sequentially sound* if each minimal process of the minimal sequential decomposition is sound.

Definition 3. A structured process is *hierarchically sound* if there is at least one set of the minimal behavioral decomposition where each minimal process is sound.

Definitions of sequentially and hierarchically soundness do not make any assumptions about formal languages. Each minimal process can be formalized and verified with any existing technique. The only restriction is that, as seen in Section 2.2, the formal behavior of each process element must depend only on its predecessors upwards the root of the tree.

To determine if process *P* is sequentially and hierarchically sound each construct defined in *P* was formalized with Petri nets according to the workflow control flow patterns [7], except the *Cancel* and *Exception* which were considered as a special case of an *Xor* to keep the behavior of each node independent from each other. See that in a structured process, the trigger of a *Cancel* may execute at any time within the scope of the *Cancel*, and since nodes are independent from each other, it is equivalent to say that the trigger either executes or not, like an *Xor*.

Each minimal process of Table 1 was verified with the classical soundness property [2]. Superscripts + and - over their names indicate whether each minimal process is sound or unsound respectively. See that both minimal processes of the minimal sequential decomposition are sound, which (according to Definition 2) means that *P* is sequentially sound. However, there is not any group of minimal processes in Table 2 having all its minimal processes sound. Therefore, from Definition 3, it means that *P* is not hierarchically sound.

Necessary and Sufficient Conditions for Soundness of Structured Processes.
For a given structured process BP, we state that if BP is sequentially and hierarchically sound, then BP is sound. No proof is given due to space restrictions.

Theorem 1. *A structured process* BP *is sound iff it is sequentially and hierarchically sound.*

By Theorem 1, it is possible to infer that process P of Figure 1 is not sound, since it is not hierarchically sound. The source of errors can be deduced from the not sound minimal processes. In this case, since the behavioral semantics of the Or establishes that once a path has been enabled it must be synchronized [7], the process has three different deadlocks: If $Cancel_{16}$ is triggered, or the process reaches elements $Term_5$ or $Term_7$, then element Or_1 will never reach its synchronization. These problems can be easily detected by checking not sound minimal processes such as P_{M_4} and P_{M_9} shown on Table 1.

4 A Method to Verify Structured Processes Based on an Unsoundness Profile

In the minimal behavioral decomposition shown in Table 1, we mentioned that behaviorally equivalent minimal processes can be omitted in the verification process. In fact, since languages have a finite set of constructs, and there is a finite number of different ways of combining such constructs, the set of all possible minimal processes which can be generated from a language is finite. If we know the set of minimal processes which can be defined from a process language, it would be sufficient to verify their soundness only once, and hence, verification results could be reused by sequential and hierarchical soundness properties as needed. To this end, we propose the specification, for a given process language, of the collection of all possible unsound combinations of minimal control flow elements. This collection defines what we call an *unsoundness profile* of a process language.

We explain this in Figure 4, which is split into a language (or meta) level, and a model level. At the language level (upper section of Figure 4), given a finite set of constructs of a process language, it is possible to generate every possible combination of constructs according to the language's metamodel, such as BPMN, UP-ColBPIP [14], etc. Based on this combination, a finite set of minimal processes representing all possible combinations of constructs of a given process language is generated. Since these minimal processes are verified independently from each other, it is possible to use different formalisms to verify each of them, e.g. if Petri nets have problems to verify advanced synchronization, minimal processes having such type of constructs could be formalized and verified with another language. Once minimal processes are verified, results are used to define the unsoundness profile of a process language.

At the model level (lower section of Figure 4), a structured process is decomposed into minimal processes by applying the minimal behavioral decomposition described in Section 3.3. See that these minimal processes are a subset of those generated at the language level, and hence, sequential and hierarchical soundness properties can be checked by determining if minimal processes are part of the unsoundness profile. With this approach, the verification of each minimal process is performed only once, at the language level.

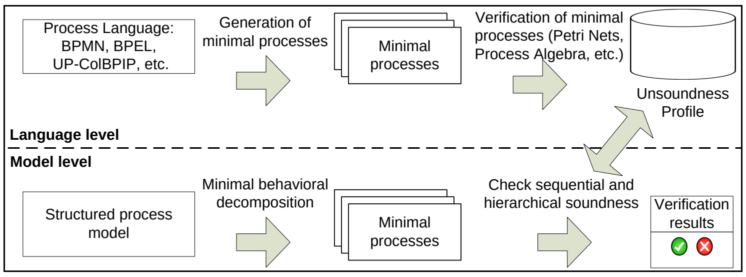

Process Language: BPMN, BPEL, UP-ColBPIP, etc.
Generation of minimal processes
Minimal processes
Verification of minimal processes (Petri Nets, Process Algebra, etc.)
Unsoundness Profile
Language level
Model level
Structured process model
Minimal behavioral decomposition
Minimal processes
Check sequential and hierarchical soundness
Verification results

Fig. 4. Using minimal processes to detect behavioral errors

Tables 3 to 6 consider a subset of BPMN constructs showing all possible unsound combinations of the minimal control flow elements *Sequence (S)*, *Xor-XorJoin (X)*, *And-AndJoin (A)*, *Or-OrJoin (Or)*, *Loop-While (W)*, *Loop-Until (U)*, *Cancel (C)*, *Exception (E)*, and *Termination (T)*. Symbols + and - are used to denote sound and unsound combinations respectively. Due to space restrictions, sound combinations where omitted whenever possible. Table 3 shows all possible unsound sequential combinations of constructs. The intersection of *Element 1* and *Element 2* defines a minimal sequential process written as $P_M = \{Seq, Elem1, Elem2\}$, e.g. $P_M = \{Seq, Term, Xor\}$ which is not sound. Table 4 shows all possible unsound combinations of constructs with loops *While* and *Until*. Similarly, the intersection of *Root* and *Element* defines a minimal hierarchical process written as $P_M = \{Root, Element\}$, e.g. $P_M = \{While, Termination\}$. Table 5 shows all possible unsound combinations where the root is a concurrent or a mutually exclusive control flow element. The intersection of *Root*, *Elem 1*, and *Elem 2* defines a minimal hierarchical process written as $P_M = \{Root, Elem1, Elem2\}$, e.g. $P_M = \{Or, Termination, Xor\}$. Finally, Table 6 shows all possible unsound combinations with cancellation and exception management, where *Elem 1* and *Elem 2* represent the scope and handler respectively, e.g. $P_M = \{Cancel, Termination, Xor\}$. Unsound combinations of elements shown in these tables define the unsoundness profile for this subset of constructs of BPMN.

By using this profile, there is no need to verify each minimal process of the minimal behavioral decomposition of process *P*, since soundness results shown in Table 1 can be directly obtained by means of the unsoundness profile.

Table 3. Soundness results for minimal sequential combinations

	Element 2								
Element 1	*X*	*A*	*Or*	*W*	*U*	*MI*	*C*	*E*	*T*
T	-	-	-	-	-	-	-	-	-

Table 4. Soundness results for minimal processes with loops

	Element
Root	T
W	-
U	-

Table 5. Soundness results for minimal processes with concurrent and mutually exclusive control flow elements

	Elem 1	T	T	T	T	T	T	T	T	X	A	Or	W	U	C	C
	Elem 2	T	X	A	Or	W	U	C	E	C	C	C	C	C	C	E
Root	*X*	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	A	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-
	Or	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 6. Soundness results for minimal processes with cancel and exception management

	Elem 1	T	T	T	T	T	T	X	A	Or	W	U	C	E
	Elem 2	T	X	A	Or	W	U	T	T	T	T	T	T	T
Root	*C*	-	-	-	-	-	-	-	-	-	-	-	-	-
	E	-	-	-	-	-	-	-	-	-	-	-	-	-

5 Discussion

The verification of the behavior of business processes has been widely studied for several years. The soundness property [8] (and its variants [2]) enables the detection of control flow errors in Workflow Nets. However, due to the state space explosion problem [12] complex constructs for advanced synchronization and exception management are not supported. In [13], authors presented a method to verify BPMN processes which use these properties. However, it does not support the construct Or. In this work, we verified a BPMN model with Or, Cancel, Exception, and Loops.

Decomposing processes into small components has improved verification performance [9, 3, 4]. This technique structures a process in a tree like manner such that each node can be verified independently from the other nodes of the tree. In this work, however, instead of verifying each independent node, we propose the verification of processes which results from the combination of a given set of nodes. This combination of nodes is what we call the minimal behavioral decomposition, where minimal processes are used to determine if the process is hierarchically and sequentially sound.

The minimal behavioral decomposition can be related to the behavioral profile and footprint, which are two techniques that capture behavioral relations between transitions of net systems [11] such as order, exclusiveness, and concurrency. However, such relations are defined between atomic transitions instead of control flow elements. This is useful to characterize the behavior of process models, but it could be redundant for verification purposes, since we are interested in the relationship between control flow constructs as a whole rather than their internal transitions.

Results returned by verification methods are key to fix problems found in processes. Some of the existing verification methods provide precise information where a deadlock occurs in a process [1,4]. However, since process elements are ruled by the behavioral semantics of their predecessors upwards the initial event of the process, an error could be caused by an element different than that pointed out by the deadlock. For example, element $Term_5$ of process P causes a deadlock. However, the designer could determine that $Term_5$ itself is only a trigger for the deadlock, but the real error is in the element Or_1, since in this case it should be replaced, for instance, by an Xor. This situation can be easily detected if we check any of the minimal processes P_{M_9} to $P_{M_{12}}$ of Table 1. Hence, an important benefit of the correctness criteria proposed in this work, is that it returns not only the position where a problem occurs, but also the exact combination of elements causing the problem.

The verification method proposed in this work is directly applicable to block-based languages such as UP-ColBPIP [14], which is a UML profile for modeling collaborative processes. However, for graph-based languages such as BPMN it is necessary to get an structured representation of a process before it can be verified. This can be reached by using existing techniques for structuring process models [3].

The specification, for a given process language, of an unsoundness profile, can be used as a systematic approach to specify behavioral antipatterns of business processes. A behavioral anti-pattern represents a combination of control flow elements which should be avoided when modeling business processes [10], and it can be seen as a combination of the least number of control flow elements which make a process unsound. By using antipatterns, the complexity of the verification of structured processes could be reduced to lineal time.

However, an important requirement is that the specification of antipatterns must be complete. So far, the specification of antipatterns have been performed manually by inspecting repositories or current literature [10] and no proofs of completeness were provided. The unsoundness profile can be used to systematize such specification. We are currently working on these aspects, not just to prove that minimal processes are useful for the specification of behavioral antipatterns, but also to prove that all possible behavioral antipatterns for structured processes can be defined from an unsoundness profile.

6 Conclusions and Future Work

In this work, we proposed a verification method for structured processes, which is based on an unsoundness profile and the sequential and hierarchical soundness properties. The method supports the verification of structured processes with constructs for advanced synchronization and exception management in polynomial time, and it supports any process language as far as input models are block-structured. As an example, we verified a BPMN process with Or, Cancel, Exception, and Loops.

The unsoundness profile is defined at a language level, and is composed of all possible combinations of constructs (called minimal processes) that can lead to behavioral errors in a structured process model. Sequential and hierarchical soundness properties are used at the model level, and make use of this profile to determine soundness of a structured process model. With this approach the formal verification is performed at the language level. At the model level, however, it is only necessary to check if a given minimal process is part of the unsoundness profile. This results in a performance improvement when verifying the behavior of process models, since the final verification is narrowed to a simple matching technique.

The use of minimal processes presents two main benefits: (1) it enables the detection of the exact combination of elements causing a problem, making the correction of errors be focused on the elements of a minimal process; (2) each minimal process can be verified independently from each other, which is useful when considering advanced control flow constructs, since each minimal process can be verified with the most appropriate formalism.

Future work is concerned with the use of unsoundness profiles for a systematic specification of behavioral antipatterns of business processes, which is so far a pending issue, despite the huge number of existing verification methods.

References

1. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous Soundness Checking of Industrial Business Process Models. In Dayal, U., Eder, J., Koehler, J., Reijers, H., eds. : Business Process Management, vol. 5701, pp.278-293 (2009)
2. van der Aalst, W., van Hee, K., ter Hofstede, A., Sidorova, N., Verbeek, H., Voorhoeve, M., Wynn, M.: Soundness of workflow nets: classiffication, decidability, and analysis: Formal Aspects of Computing, 1-31, (2010).
3. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more Focused Control-Flow Analysis for Business Process Models through SESE Decomposition: Proc. ICSOC 2007, pp. 43-55, (2007).
4. Polyvyanyy, A., Weidlich, M., Weske, M. Connectivity of workflow nets: The foundations of stepwise verification. Acta Informatica (ACTA), vol. 48(4), pp. 213–242, (2011).
5. Grossmann, G., Schrefl, M., Stumptner, M.: Modelling and enforcement of inter-process dependencies with business process modelling languages. Journal of research and practice in information technology 42(4), 289-322 (2012)
6. OMG, BPMN 2.0: http://www.omg.org/spec/BPMN/2.0/
7. Russell, N., ter Hofstede, A., van der Aalst, W., Mulyar, N.Workflow control-flow patterns: A revised view. Technical Report BPM-06-22, BPMcenter.org, 2006. BPM Center Report.
8. van der Aalst, W. The application of Petri nets to workow management. Journal of Circuits, Systems, and Computers, 8(1):21-66, 1998.
9. Hauser, R.F., Friess, M., Kuster, J.M., Vanhatalo, J.: An Incremental Approach to the Analysis and Transformation of Workflows Using Region Trees. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 38(3), 347-359 (2008).
10. Koehler, J., Vanhatalo, J.: Process anti-patterns: How to avoid the common traps of business process modeling. IBM WebSphere Developer Technical Journal 10(2+4) (2007).
11. Weidlich, M., van der Werf, J. On profiles and footprints - Relational semantics for petri nets. In Application and Theory of Petri Nets, vol. 7347, pp. 148-167 (2012).
12. Valmari, A.: The state explosion problem. In: Lectures on Petri Nets I: Basic Models. Lecture Notes in Computer Science, vol. 1491 pp. 429–528. Springer, Berlin (1998)
13. Dijkman, R., Dumas, M., Ouyang, C. Semantics and analysis of business process models in BPMN, Information & Software Technology 50 (12) pp. 1281-1294 (2008).
14. Villarreal, P.D., Lazarte, I., Roa, J., Chiotti, O.: A Modeling Approach for Collaborative Business Processes Based on the UP-ColBPIP Language. In: Business Process Management Workshops. Volume 43. Springer. 318-329 (2010).