Instantaneous Soundness Checking of Industrial Business Process Models

Dirk Fahland[1], Cédric Favre[2], Barbara Jobstmann[4], Jana Koehler[2], Niels Lohmann[3], Hagen Völzer[2], and Karsten Wolf[3]

[1] Humboldt-Universität zu Berlin, Institut für Informatik, Unter den Linden 6, 10099 Berlin, Germany, fahland@informatik.hu-berlin.de

[2] IBM Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland, (ced|koe|hvo)@zurich.ibm.com

[3] Universität Rostock, Institut für Informatik, 18051 Rostock, Germany, (niels.lohmann|karsten.wolf)@uni-rostock.de

[4] EPF Lausanne, 1015 Lausanne, Switzerland, barbara.jobstmann@epfl.ch

Abstract. We report on a case study on control-flow analysis of business process models. We checked 735 industrial business process models from financial services, telecommunications and other domains. We investigated these models for soundness (absence of deadlock and lack of synchronization) using three different approaches: the business process verification tool Woflan, the Petri net model checker LoLA, and a recently developed technique based on SESE decomposition. We evaluate the various techniques used by these approaches in terms of their ability of accelerating the check. Our results show that industrial business process models can be checked in a few milliseconds, which enables tight integration of modeling with control-flow analysis. We also briefly compare the diagnostic information delivered by the different approaches.

1 Introduction

Various studies [1] show that many business process models contain control-flow errors such as deadlocks. Such errors obstruct the correct simulation, code generation and execution of these models. Therefore, detecting and removing control-flow errors becomes crucial in view of the increasing popularity of these use cases. Preventing errors by using a restricted, for example a purely block-oriented modeling language is rarely an option because a model typically needs to reflect the real causal process structures present in an enterprise.

In this paper, we are interested in checking business process models for the classical notion of soundness [2, 3], which entails the absence of deadlocks and lack of synchronization, which are explained in more detail below. Our interest in soundness is motivated by an increased need in creating business process models not only for documentation purposes, but for an input into a translation and code generation process where, e.g., WS-BPEL code is generated. Soundness is necessary to translate a process modeled in a graph-based language, such as UML Activity Diagrams or BPMN, to WS-BPEL in a way that preserves the execution semantics *and* the structure of the process. This use case requires a process to be checked in a relatively short amount of time, say

500 ms or less, because checks are to be performed on each major modification, that is, at least on each save operation on the process model. Moreover, entire libraries of up to several hundred processes have to be checked when models are exchanged between modeling tools. Short response times make it possible to integrate control-flow analysis tightly with modeling such that errors are found at the earliest possible time, which would allow the user to relate an error to the latest change in the model. Furthermore, use cases such as code generation from models also require that an analysis produces sufficient diagnostic information to allow the user to locate and repair the detected errors.

A variety of techniques for checking soundness exists in the literature. They differ in their completeness, worst-case complexity, and quality of diagnostic information returned. Most techniques can be easily combined to optimize performance. The most flexible technique is state space exploration. It is most likely applicable to other similar use cases, such as checking a relaxed notion of soundness or checking more expressive languages supporting OR-joins and other advanced synchronization constructs. But state space exploration suffers from the state space explosion problem, i.e., the fact that the number of reachable states can be exponential in the size of the process model. On the other hand, many business process models have a simple structure, for instance, they are sequential to a large extent, hence they do not necessarily have a large state space.

At the onset of our project, it was not clear from the literature how large the state spaces of control-flow models of realistic business processes are and hence which additional techniques are needed to check their soundness as fast as required by our use case. It was completely open whether such a check can be performed in the required time and in such a way that sufficient diagnostic information is obtained. In addition, given the variety of available approaches, it was unclear which would be the most suitable techniques.

In this case study, we investigated three approaches implemented in three different tools as outlined in Fig. 1:

1. The Petri net model checker LoLA [4], from which we used CTL model checking with partial order reduction.
2. The business process verification tool Woflan [3], which uses a mixture of Petri net analysis techniques, most notably structural Petri net reduction and S-coverability analysis, as well as a form of state space exploration based on coverability trees.
3. The process validation technique used in the IBM WebSphere Business Modeler, which combines SESE decomposition [5] with heuristics and state space exploration.

The data set for our case study was a large collection of process libraries available in the IBM WebSphere Business Modeler tool. The first two approaches required a translation of these models into Petri nets, whereas for the third approach, the models were translated into workflow graphs.

We obtained the following results: Based on the 735 process models that we analyzed, soundness of industrial business process models can be decided in a few milliseconds per process. Although many processes are simple enough that state space exploration alone would be sufficient to decide soundness, this method is not sufficient

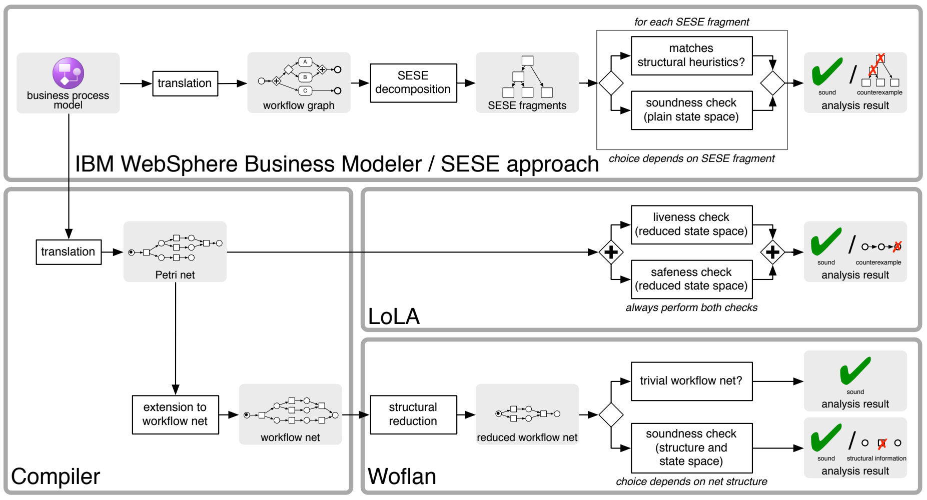

Fig. 1: Three different approaches and tools to check soundness.

in general. However, all three approaches perform similarly fast, meeting the above-mentioned performance requirements. This implies that one can focus on different requirements such as the quality of the returned diagnostic information when deciding for a soundness-checking technique. Our study also shows that there is a high percentage of unsound models, confirming the need for better tool support for execution-aware modeling.

Previous studies [6, 7, 3] on checking soundness or the similar notion of *EPC soundness* of realistic business process models concentrate on error findings and error prediction. These studies do not report runtimes for the analysis. Mendling [8] reports an average analysis time of 1.8 secs and maximal time of 142 secs for checking the EPC soundness of 604 processes. His technique of using structural reduction rules that operate directly on the process model does not find all violations of soundness. A post-processing with state space exploration is not included in these runtimes. The same set of processes was also checked for relaxed soundness [9] with a reported runtime of 46 secs per process on average [8, 1]; however, no maximal times are reported. Recent work [10] extends control-flow analysis to more advanced synchronization constructs such as OR-joins and cancelation regions, but so far no empirical results have been reported. A preliminary and incomplete version of the SESE decomposition technique that used heuristics only, but did not include state space exploration, was partially evaluated on a different set of data [5].

The remainder of this paper is organized as follows: In Sect. 2, we discuss the data used in this study, their translation to workflow graphs and Petri nets, and the notion of soundness. Sections 3, 4, and 5 present the three approaches together with the results they achieved on the data. In Sect. 6, we review the results in a comparison of the three approaches and draw conclusions.

2 Selecting the Empirical Data and Preparing the Case Study

2.1 Sampling the process data

We scanned a large set of real-world data available to the IBM team for our practical validation of the soundness-checking approaches and tools. These data mostly resulted from modeling activities in customer projects within a SOA context, i.e., processes were captured with the final goal of implementing them in a Service-Oriented Architecture. The models covered various industry domains such as financial services, automotive, telecommunications, construction, supply chain, health care, and customer relationship management. We also looked at large collections of reference processes that were created for the insurance and banking domain by users who explored different modeling styles, i.e., different ways of capturing data and control-flow at varying level of granularity. All models were available in the IBM WebSphere Business Modeler tool represented in a language that currently combines elements from UML Activity Diagrams and the Business Process Modeling Notation (BPMN), but some of them had originally been created in other tools first and then imported into the IBM product.

It turned out that only some of the model collections considered are useful for our purposes. Many process models are in fact quite small, as good modeling practice suggests an appropriate structuring of processes into subprocesses, and are therefore not a challenge for our soundness-checking approaches. Others, in particular those created in other tools, might not have been created with the appropriate notion of soundness or might have been created by non-experts and consequently turned out to be syntactically incomplete and therefore flawed in such a way that it made no sense to consider them further. In the course of our experimental studies, we therefore reduced our initial test set of approx. 3000 models to 5 libraries of 735 different models in total from the insurance, banking, customer relationship, as well as construction and automotive supply chain domains. We completely anonymized the data in these models, e.g., task names would be replaced by enumerations $t1, t2, \ldots$, and named these libraries A, B1, B2, B3, and C. These anonymized libraries, which have been stripped off all semantics and represent only purely structural information, were the input for the tools LoLA, Woflan, and the SESE approach. Libraries B1, B2, and B3 partially overlap as they represent a series of models from the same domain created over a period of two years, in which a library changed to the next by adding more process models and refining all models with further detail. The number of 735 different processes therefore counts only the latest library in this series, which is B3 with 421 processes, together with the 282 processes from library A and 32 processes from library C.

Table 1 characterizes the data from our process libraries by measuring the number of nodes that represent tasks, subprocesses, gateways, start and end events, and the number

Table 1: Static data.

	A	B1	B2	B3	C
Avg. / max. number of nodes	14 / 46	17 / 69	16 / 67	18 / 83	27 / 118
Avg. / max. number of edges	33 / 127	29 / 147	31 / 202	33 / 195	33 / 145
Avg. / max. node inflow	2.52 / 13	1.76 / 15	1.90 / 69	1.86 / 27	1.84 / 4
Avg. / max. node outflow	1.03 / 8	0.94 / 13	0.99 / 15	1.05 / 30	1.83 / 4

of edges that represent control- and data-flow connections between nodes. The inflow and outflow numbers capture the branching degree that occurs in the models. Note that for libraries B1 and B2 the average outflow is smaller than 1, because many end events occurring in these models have outflow 0.

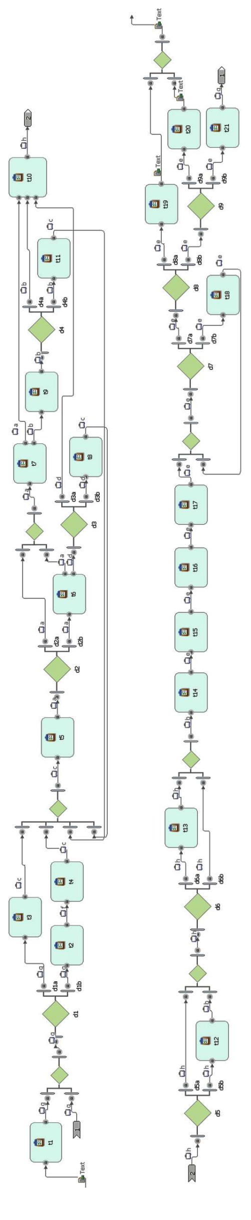
Fig. 2: Structure of a typical, average-sized process model.

To illustrate such a process model, we show a typical average-sized example from library C in Fig. 2. We split the flow into two parts: the end of the left flow continues at the beginning of the right flow. This process model contains 21 tasks representing elementary, not further distinguished process steps, 16 gateways to encode XOR-splits and -merges, and 51 edges representing data- and control-flow connections. A task can have multiple incoming and outgoing edges that encode implicit AND-splits and -joins of the control and data flows. The example model also contains several cycles: There is a large cycle that spans almost the entire process and there are three smaller cycles within this large cycle – two of them are nested within each other, whereas the third occurs at the end of the process.

2.2 Translation into workflow graphs and Petri nets

Data-flow constructs in the language of the current version of the IBM WebSphere Business Modeler are similar to UML activity diagrams. Here, we only consider explicit data-flow connections and no repositories, because each such connection implies a control-flow connection. Control-flow constructs are visualized in BPMN.

The translation of the process models into the format required by the soundness checkers focuses on the following modeling elements: start and end events, tasks, subprocesses, control flow, input and output sets, and gateways. Data flow is ignored during the translation, i.e., each explicit data-flow connection is replaced by a control-flow connection. Data flow connections from and to repositories were not considered at all. The current language supported by IBM WebSphere Business Modeler contains XOR- and AND-gateways as well as an OR-split, but no OR-join. The translation is well-known and therefore not repeated here; details are provided elsewhere [11].

A task can have multiple incoming and outgoing edges (inputs and outputs) that can be grouped into sets. Input and output sets of tasks are translated into gateway logic as illustrated in Fig. 3. In Fig. 3, task A has inputs a, b grouped into one set and inputs c, d, e grouped into another set with the meaning that A can execute if it either receives a and b as input or c, d, and e. The output (sets) of task A are f, g, h and i, j, k. The presence of an input or output is expressed by placing a *token* on an edge between two nodes. Tokens move through the process as a task or gateway executes, taking the process from one state to another state in the usual way.

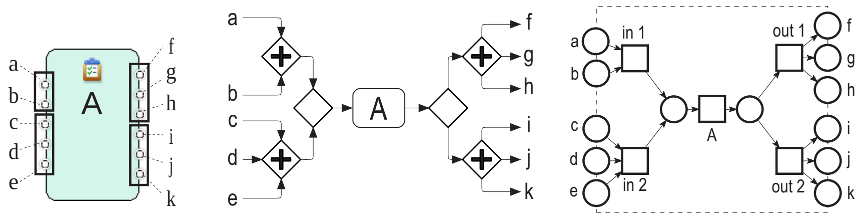

Fig. 3: Translation of a task with disjoint input and output sets (left) into the corresponding workflow graph (center) and Petri net patterns (right).

In the center of Fig. 3, we see the translation into a *workflow graph* [2, 5], which is a control-flow graph containing only gateways and tasks. To the right, we see the resulting Petri net. In general, input and output sets can overlap, which would lead to *non-free-choice* Petri nets as a result of the translation [12]. However, none of the syntactically valid process models contained in our test set used overlapping inputs or output sets, i.e., the translation will only return free-choice nets in our case study. This makes it possible to benefit from fast analysis techniques for free-choice Petri nets, see for example Sect. 4. Furthermore, users of the tool can specify which input set activates which output set, but this information was not provided in any of the models. For the translation, we therefore assumed that each input set can potentially activate each output set. Two different translations into workflow graphs and Petri nets were implemented, although the Petri nets could also be directly obtained from the workflow graphs by a well-known construction [2]. The Petri net models are available at http://www.service-technology.org/soundness in PNML format.

2.3 Soundness

Figure 4 shows a workflow graph without any tasks as it occurs in the middle part of the process in Fig. 2 and to which we added a start and an end event. This process model contains a *lack of synchronization* error as well as a *local deadlock*, which are not so easy to spot in the first place.

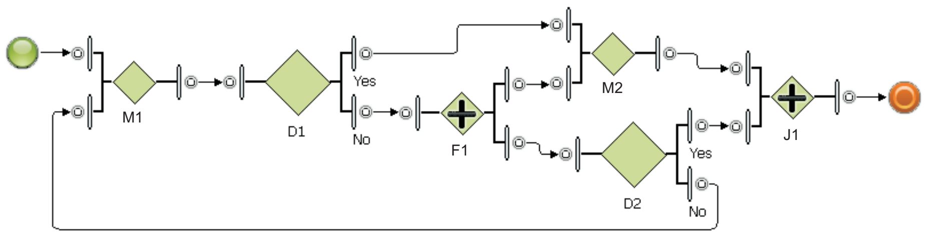

Fig. 4: Workflow graph with deadlock and lack of synchronization errors.

A *local deadlock* is a reachable state s of the process that has a token on an incoming edge e of an AND-join such that each state that is in turn reachable from s also contains a token on e, i.e., the token is 'stuck' on e. A deadlock arises for example, if two alternative paths are merged by an AND-join or if an AND-join occurs as an entry to a cycle. In the example in Fig. 4, a deadlock occurs when a token travels the Yes edge leaving the XOR-split $D1$. Eventually, this token will reach the AND-join $J1$ via the upper incoming branch. However, no other token will ever arrive at the lower incoming branch of $J1$.

A reachable state s contains a *lack of synchronization* if there is an edge that has more than one token in s. If such an edge contained a task, it would be executed twice. A lack of synchronization arises for example, if two parallel paths are merged by an XOR-merge or if the exit of a cycle is an AND-split. In the example in Fig. 4, a lack of synchronization occurs when a token travels the No edge leaving the XOR-split $D1$. This token will activate the AND-split $F1$, which leads to a token reaching the XOR-merge $M2$ and another token traveling the cycle $D2, M1, D1, F1$. This can result in multiple tokens on the edge from $F1$ to $M2$.

A process model that has neither a lack of synchronization nor a local deadlock is said to be *sound*. This definition of soundness is equivalent to the classical definition of soundness in free-choice Petri nets [3]. There are other equivalent characterizations that are exploited by some of the tools used in our case study, see for example Sect. 5. Their formal treatment can be found elsewhere [2, 3, 13, 5].

Table 2 summarizes the results of our analysis for the libraries. On average, only 46% of all process models are sound ranging from 37% for library B1 to 53% for library A. The table also shows the degree of concurrency that can be found in a process model, i.e., the maximum number of tokens that occur in a single reachable non-error state of the process. Row 5 shows the number of processes with more than one million reachable states, which include error states, and processes that have infinitely many reachable states such as the process shown in Fig. 4. To exclude those, we measured the size of the state space of each *sound* process, which is always finite, which still returned a few processes with more than one million states. The average values, however, suggest that such processes are rare.

Table 2: Dynamic data.

	A	**B1**	**B2**	**B3**	**C**
Processes in library	282	288	363	421	32
sound	152	107	161	207	15
unsound	130	181	202	214	17
Avg. / max. concurrency	2 / 13	8 / 14	16 / 66	14 / 33	2 / 4
Processes with >1000000 states	26	19	29	38	7
Processes with >1000000 states (only sound)	0	1	4	4	0
Avg. number of states (only sound, <1000000 states)	26	71	322	4911	680
Max. number of states (only sound, <1000000 states)	213	2363	28641	588507	8370

3 State Space Verification with LoLA

LoLA [4] is a tool that decides numerous properties by an inspection of the state space of a given Petri net. For making state-space inspection feasible, it offers several state-space reduction techniques. The experiments were carried out with the current version of LoLA 1.11 [14].

Soundness as a model-checking problem. The process models have to be translated into Petri nets prior to the verification as sketched in Sect. 2.2. To verify soundness, LoLA works in two runs on the resulting Petri nets. In the first run, it checks for local deadlocks and in the second run for lack of synchronization.

A process has no deadlock iff a *final* state can be reached from every reachable state; a state is final iff each token has reached an end node. The latter can easily be expressed as a state predicate in LoLA. The former can be expressed as a CTL formula over this predicate and checked by LoLA directly. LoLA checks the property *on-the-fly*, i.e., while the state space is being generated. As soon as LoLA detects a violation, it stops and returns the violating state. Once an error state has been found, a reachability check is used to produce a trace to the error state.

LoLA has a switch that causes state-space generation to be stopped if an *unsafe* state is generated. A state is *unsafe* if a single place contains more than one token, which indicates a lack of synchronization in the original process model. This simultaneous check for lack of synchronization in the first run prevents that LoLA tries to generate an infinite state space and also optimizes performance for finite state spaces. If an unsafe state is found, a trace leading to it is returned immediately. However, the test for unsafe states cannot detect all lack of synchronization errors. Therefore, if no error has been detected during the first run, LoLA is invoked a second time on each net, this time explicitly checking for lack of synchronization.

Lack of synchronization, i.e., unsafeness of states, can be expressed in LoLA as the state predicate $\bigvee_{p \in P} m(p) > 1$, where P is the set of places of the Petri net. As this set can become very large, e.g., on our test data, a maximum of 275 places occurred, we simplified this predicate to optimize performance. We can assert by construction for several places in the Petri net that they cannot obtain more than one token unless a preceding place is also able to do so. In essence, only places that represent an XOR-merge or an exit of a cycle need to be considered. The resulting state predicate is checked for reachability by LoLA. If the predicate is satisfied, a lack of synchronization is identified and LoLA produces a trace to the error state.

We used *partial order reduction* [15] for the results of this paper. This technique suppresses insignificant orderings of concurrently enabled events. LoLA ensures that the property to be checked is preserved by the reduction.

For the example depicted in Fig. 2, LoLA detects a lack of synchronization in the first run, concludes that the net is unsound, and returns an error trace consisting of 36 states.

Experimental setup. After translating the process models into Petri nets with our compiler [11, 16], we performed the two checks explained above. We ran the experiments on a notebook with a 2.16 GHz processor and 2 GB RAM. We set a bound of one million states for each net and classified a net as *intractable* if this bound was reached.

Table 3: Analysis statistics for LoLA.

	A	B1	B2	B3	C
Intractable processes (no partial order reduction)	0	2	5	4	0
Avg. number of explored states (partial order reduction)	50.42	40.60	37.52	60.76	127.28
Max. number of explored states (partial order reduction)	187	1591	1591	6467	1469
Avg. length of error trace (partial order reduction)	30.24	10.81	12.12	11.21	53.17
Max. length of error trace (partial order reduction)	67	110	75	103	120
Analysis time for library (partial order reduction) [ms]	2680	2356	3184	3878	305
Analysis time for library (struct. reduced, partial order reduction) [ms]	2523	2192	3025	3575	275

Experimental results. Compared with the original models, the Petri nets that we obtained have about 5.5 times as many nodes and edges, see Table 1, which is due to the more fine-grained representation of the process logic in Petri nets as illustrated by Fig. 3. The largest net results from a process model in library C and has 558 nodes and 607 edges.

Without partial order reduction, not all nets could be analyzed, see row 1 of Table 3. When partial order reduction is used, there is no intractable process. In fact, the largest state space explored consists of only 6467 states. Only around 100 states need to be explored on average. During the experiments, LoLA never consumed more than 2 MB of memory, which allows for an unobtrusive verification process, which was not clear in advance. Table 3 summarizes the results.

In a variant of the experiment, we also applied structural Petri net reduction rules [17] to each Petri net before checking it with LoLA. These rules reduce the size of the net, while preserving soundness. The last row of Table 3 shows that structural net reduction hardly has any effect on the runtime. Note that these runtimes do not contain the time needed for structural reduction.

The longest error trace contains 120 Petri net states. When mapped to the original process model, this trace corresponds to a sequence of 40 tasks.

4 Soundness Verification with Woflan

Woflan [3] is a tool for verifying the soundness of business processes modeled as Petri nets. It poses syntactic restrictions on the Petri nets it can analyze, most notably, that each net must have a unique terminal place. Such a net is called a *workflow net*.

Preparing the input for Woflan. Only a few process models from our libraries have a unique terminal node, hence only a few of the resulting Petri nets would have a single terminal place and thus be workflow nets. However, a multi-terminal net N can be *extended* to a workflow net N' using the algorithm of Kiepuszewski et al. [13, Proof of Theorem 5.1]. This algorithm adds new edges to N that cause every terminal place of N to be marked in every run. It then synchronizes all terminal places of N by a final transition, which produces a token on a new unique terminal place. Kiepuszewski et al. [13] show that soundness is preserved by the extension assuming that the original net N is a free-choice Petri net. As we discussed in Section 2, our data set meets this assumption. It is also easy to see that the extension preserves unsoundness. Extending *N* only requires a depth-first search in *N* for each of its terminal places.

The tool Woflan. Woflan implements a complex algorithm [3] to check soundness. It uses various techniques from Petri net structure theory as well as state space exploration. If the workflow net is a free-choice net, which is the case in our experiments, Woflan's algorithm reduces to the following procedure (recall also Fig. 1):

(1) First, soundness-preserving structural reduction rules from Petri net theory [17] reduce the size of the input. If the resulting net is *trivial*, i.e., it has only one transition, Woflan immediately concludes that it is sound. (2) Otherwise, Woflan checks the *S-coverability* of the net [3] to exploit the following properties: (2a) A free-choice Petri net that is not S-coverable is unsound, and Woflan quits; the unsoundness can be caused by a deadlock or a lack of synchronization. (2b) A Petri net that is S-coverable has no lack of synchronization, but may contain a local deadlock [3]. (3) If step (2b) applies, Woflan searches for local deadlocks–in Petri net terms a *dead* or a *non-live* transition–by state space exploration, i.e., by constructing the net's coverability graph. The techniques underlying steps (2) and (3) have exponential worst-case complexity in the size of the net.

Woflan provides two kinds of diagnostic information in this setting: If step (2a) applies, it returns a list of places that are not S-coverable, i.e., that contribute to a deadlock or a lack of synchronization. If Woflan detects a deadlock in step (3), it returns a list of dead and non-live transitions that create this deadlock.

Experimental setup. We verified the workflow nets resulting from the translation with a command-line version of Woflan in a batch on a notebook with a 1.66 GHz processor and 2 GB RAM. We ran the experiments twice, the first time without applying structural reduction, the second time with. Aiming at *instantaneous* verification, we interrupted Woflan if the verification time exceeded 5000 ms. In these cases, we classified the process as *intractable* for the analysis.

Experimental results. Table 4 summarizes the results of our Woflan experiments. Our first analysis on the unreduced workflow nets was intractable for 46% of library A and for 19%-28% of libraries B1 to B3. The size of these nets corresponds to the numbers presented for LoLA in Sect. 3. Surprisingly, the analysis became intractable mostly when Woflan checked S-coverability–the technique's exponential worst-case complexity explains this observation. If S-coverability completed successfully, proving absence of deadlocks by state space exploration was tractable in all but 11 cases. Library C was analyzed completely and fairly quickly, see Table 4, row 4. The structure of its models seems to be more suitable for Woflan. We observed that without capping analysis after 5000 ms, Woflan's analysis frequently required between 15 min to more than 1 h per process.

In the second experiment, we let Woflan apply structural Petri net reduction rules prior to analysis, which on average reduced nets in size by a factor 5. The largest net, which resulted from a process in library B3, has 74 nodes and 232 edges. About a third

Table 4: Analysis statistics for Woflan.

	A	B1	B2	B3	C
1) Without structural reduction					
Intractable processes	129	54	77	119	0
due to S-coverability	129	53	74	112	0
due to state space exploration	0	1	3	7	0
Analysis time [ms]	860812	288218	429343	755875	2375
2) With structural reduction - no intractable processes					
Sound by structural reduction	81	79	134	162	8
Unsound by S-coverability	130	176	197	210	11
Processes that required state space exploration	71	32	32	49	8
Max. number of explored states	8	7	8	8	12
Analysis time per library [ms]	1120	1305	1795	2315	165
per process [ms], avg. / max.	3.97 / 20	4.55 / 40	4.94 / 91	5.50 / 1142	6.11 / 90

of all models were reduced to the trivial workflow net, see Table 4, row 5. Thus, structural reduction alone identified 53% (libraries A and C) to 80% (libraries B) of all sound processes. Woflan classified about two thirds of the remaining nets as unsound by proving that a net is not S-coverable and free-choice. These nets constitute almost 100% of all *unsound* models. For example as Table 2 shows, library B3 has 213 unsound processes, out of which 210 are not S-coverable. Only for the remaining nets–between 9% (library B2) and 25% (libraries A and C) of the processes–was a state space of at most 12 states explored to complete the analysis. Woflan checks soundness of a process in about 4 to 6 ms on average, with a maximum runtime of less than 90 ms. The one exception in library B3 ran into the exponential worst-case complexity of the S-coverability check, see Table 4, row 10.

Interpreting Woflan's diagnostic information on the original process model is not trivial. For instance, in the workflow net that corresponds to the model of Fig. 4, Woflan reports *all* places to be not S-coverable, hiding the concrete source or location of the error.

We conclude that S-coverability checking alone does not sufficiently speed up the analysis for instantaneous verification of free-choice Petri nets. However, this technique becomes very powerful in combination with Petri net reduction rules. For up to 91% of our examples, soundness or unsoundness was proven alone by these two techniques. Only in the remaining cases, was a fairly simple state space exploration required.

5 The SESE Decomposition Approach

The SESE approach structurally decomposes a business process model into smaller fragments, for which soundness is analyzed by heuristics and state space exploration. If each fragment is sound, then the entire process is sound. The analysis is done on a workflow graph, which is obtained from the original process model as sketched in Sect. 2.2. The SESE approach combines the following three techniques.

State space exploration with SESE. The base technique for the SESE approach is state space exploration. Soundness of a workflow graph can be decided by checking that no explored state has more than one token on a single edge (lack of synchronization) and that each non-terminal state has a successor state (*global deadlock*). If a workflow graph has no lack of synchronization, then every local deadlock manifests itself eventually in a global deadlock in each execution. The workflow graph's state space is explored by depth-first search. The analysis terminates upon the first state that violates one of these two properties and returns a trace leading to this state. If there is no error, the entire state space must be explored.

SESE decomposition. To mitigate the state space explosion problem, we use a parsing technique called the *Refined Process Structure Tree (RPST)* [18]. The RPST decomposes a workflow graph into a hierarchy of *fragments* with a single entry and single exit (SESE) of control. A SESE fragment of a workflow graph is a subgraph that has a single entry node and a single exit node. Fig. 5 shows an example of a workflow graph that is decomposed into such fragments. Multiple end nodes can be handled by adding a unique dummy end node as shown in Fig. 5. Soundness is compositional with respect to SESE fragments, i.e., each fragment can be checked in isolation [5]. To verify the soundness of a fragment, each child fragment can be treated as a task (node) of the workflow graph.

The soundness of a SESE fragment can be checked using plain state space exploration. Because fragments are usually considerably smaller than the entire workflow graph, the input to the state space exploration is smaller, in turn resulting in smaller state spaces to be explored. The decomposition is done in linear time and the number of fragments is at most linear in the size of the workflow graph. The time to analyze an entire workflow graph is then dominated by the size of its largest fragment.

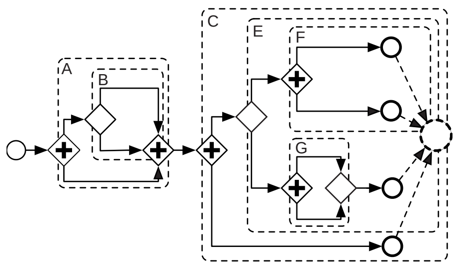

Fig. 5: Decomposition of a workflow graph using the Refined Process Structure Tree.

The diagnostic information returned is a fragment showing the error as a trace relative to the fragment. This shows an error inside a smaller scope and shortens the error trace. Moreover, the checker can detect multiple errors at once, up to one per fragment. This includes 'unreachable' errors, such as a lack of synchronization in a fragment, e.g., in fragment *G* in Fig. 5, that cannot be reached by plain state space exploration because this fragment is obstructed by another deadlock earlier in the process, e.g., fragment *A* in Fig. 5.

Heuristics. In practice, many fragments have a simple structure that can be recognized as sound or unsound in linear time using structural heuristics [5]. For example, if a fragment contains only XOR-gateways, it is purely sequential and therefore sound. If a fragment contains at least one XOR-split, but no XOR-join it must be unsound. In this case, the XOR-split can be highlighted inside the highlighted fragment as diagnostic information. We implemented 14 heuristics, all of which can be evaluated based on a single count of the gateway types within a fragment. Only a fragment that does not match any of the heuristics becomes the subject of state space exploration; such a fragment is said to be *complex*. Therefore, heuristics are expected to speed up the analysis by bypassing the state space exploration.

Experimental setup. The SESE approach is implemented as part of the IBM WebSphere Business Modeler, in which we also conducted the experiments collecting results from the debugging console. The analysis time reported also includes the production of the regular error report in the tool.

We conducted three experiments to measure the impact of the SESE decomposition and the heuristics: First, we used plain state space exploration only. Second, we decomposed each process into its SESE fragments, and *all* fragments were then analyzed by state space exploration. In the third experiment, we used decomposition in combination with heuristics and state space exploration, i.e., state space exploration was applied only to complex fragments.

The analysis time is computed as an average over five runs. The overhead for loading the process models from the hard drive into memory was measured separately and factored out from the analysis time. The SESE experiment was conducted on a notebook with a 2 GHz processor and 3 GB RAM.

A process is *intractable* if more than 100000 states have to be explored. This threshold value is based on the experience that the time needed would otherwise exceed a value that is acceptable in the use case of instantaneous verification as described in Sect. 1.

Experimental results. Table 5 shows the results for the three experiments described above. For plain state space exploration, we observe that at most 6 out of 363 processes (all contained in library B2) are intractable, i.e., less than 2 percent. Analyzing library A, which contains no intractable process, only requires 490 ms.

When using the decomposition into fragments, we observe that there no longer is an intractable process. However, the analysis time of library B2 is dominated by one particular process which took 25 sec to be analyzed. All other processes took less than 1 sec each. SESE decomposition reduces the size of the input to state space exploration by an average factor between 1.5 and 4. The number of states that are explored for a particular process is the sum of the number of states explored for each fragment of the process. Table 5 shows that the number of states that have to be explored for a process on average reduces by up to a factor of 13.8 with respect to experiment 1. After decomposition, there is still a fragment that has 16403 states.

Library A shows that computing the decomposition does not always pay off: This library is analyzed faster without decomposition. The analyses of the other libraries, however, clearly benefit from the decomposition: Decomposition reduces the analysis time by a factor between 5 and 67 with respect to plain state space exploration.

In addition, we recorded the length of the error trace in both experiments. Error traces are notably smaller when they relate only to a fragment, rather than to the entire workflow graph. The error trace lengths were reduced by a factor of 4.7 on average. Note that the error trace using the decomposition into fragments starts at the start node

Table 5: Experimental results for the SESE decomposition approach.

		A	B1	B2	B3	C
1) State space exploration - reference						
Explored states per process (avg.)		42.8	826.4	1879.3	1508.1	149.7
Explored states per process (max.)		241	17176	28684	28688	2517
Intractable processes		0	2	6	5	0
Analysis time [ms]	library	490	30019	197670	135178	30019
	process (max.)	16	13186	76700	24624	62
2) Using the decomposition - no intractable processes						
Size reduction (workflow graph / largest fragment) (avg.)		4.1	3.8	3.9	4.7	2.7
Explored states	process (avg.)	52.4	31.6	86.7	38.4	61.7
	reduction w.r.t. exp. 1 per process (avg.)	1.0	13.8	13.4	10.2	1.5
	process (max.)	201	268	16534	311	356
	fragment (max.)	53	117	16403	68	120
Analysis time [ms]	library	1587	1359	35495	2446	447
	process (max.)	16	16	25286	32	32
3) Using the heuristics - no intractable processes						
Portion of fragments analyzed by heuristics		97%	97%	98%	98%	99%
Explored states	process (avg.)	6.0	2.3	3.2	2.5	10.1
	reduction w.r.t. exp. 2 per process (avg.)	28.3	22.9	78.4	29.6	34.1
	process (max.)	53	36	165	24	120
	fragment (max.)	53	36	165	20	120
Analysis time [ms]	library	1247	1390	1681	2303	318
	process (max.)	16	31	16	31	62

of the fragment and not at the start node of the workflow graph. The decomposition allows us to detect multiple errors per process, at most one per fragment. For library B2, we measured an average of 1.55 and a maximum of 7 unsound fragments per unsound process.

The third experiment shows that the heuristics speed up the analysis further. For all libraries, more than 97 percent of the fragments match some heuristic and only the remaining ones have to go into state space exploration. We noted that a process usually contains not more than one complex fragment, out of an average of 16 fragments per process. Only the largest process, which has 122 fragments, contains two complex fragments; no process contained more. The small number of complex fragments results in a reduction factor of up to 78.4 for the average number of states that were explored to analyze a process. The use of the heuristics reduces the analysis time of library B2 by a factor 21 with respect to experiment 2. For the other libraries, the differences in the analysis times is not significant. The maximum analysis times per process range from 10 to 62 ms.

6 Conclusion

We showed that different techniques can be used to check the soundness of industrial business process models reliably in fractions of a second.

For the *state space approach* using LoLA, we found that partial order reduction and on-the-fly verification are the essential factors for success. Although many processes could have been verified on a brute force state space, some state spaces exploded without the use of partial order reduction. While it was difficult to handle full state spaces, the exploration of erroneous state spaces up to the first error was efficient. Surprisingly, the prior application of structural Petri net reduction has only a minor impact on performance. This may be because many of the existing reduction rules address situations that also partial order reduction on the state space is dealing with.

In the *structural approach* using Woflan, we saw that the original models can easily be translated into the more restrictive notion of workflow nets with just one terminal node. Another observation was that the performance of Woflan can mainly be attributed to the structural Petri net techniques. In the few cases where Woflan had to explore a state space, this state space was rather small because of prior application of structural reduction. Here, structural reduction turned out to be beneficial as Woflan does not provide partial order reduction.

In the *decomposition approach* using SESE fragments, we learned that the approach did not suffer from severe state space explosion as the state space is only computed locally for a typically small fragment of the process model. Moreover, structural heuristics are sufficient to handle most of the fragments, which allows one to bypass state space exploration altogether.

While being similar in their performance, the three approaches chosen vary with respect to the diagnostic information they provide. The state space approach used by LoLA is able to return an error trace of manageable size that can be simulated or animated. The SESE approach can detect multiple errors in one analysis run and localizes each error in a particular, typically small, fragment of the original model. This also reduces the length of the error trace by a factor of 4.7 on average. Moreover, the approach can provide additional information depending on the heuristics applied. Woflan returns some Petri-net specific information that needs to be interpreted carefully before it can be shown to a business user.

Another notable difference between the three approaches is that Woflan is specifically built for checking soundness and the SESE approach is specifically designed to check soundness instantaneously, whereas LoLA is a generic model checker for Petri nets that could more easily be adapted to check other temporal properties of business processes.

We would like to point out that there are other promising algorithms to check soundness, especially polynomial-time algorithms exploiting the free-choice property [12]. We could not include those in our case study because we are not aware of available implementations.

Finally, note also that the various techniques could easily be combined in different ways. For example, one could apply SESE decomposition to break the model into smaller fragments, then use heuristics and structural Petri net reduction to quickly sort out sound fragments that have a simple structure, and then finally check the remaining fragments with state space exploration based on partial order reduction to obtain detailed localized error information.

Acknowledgements We would like to thank Eric Verbeek for his substantial support in providing a Woflan version for our experiments. Dirk Fahland is funded by the DFG-Graduiertenkolleg

"METRIK" (1324). Niels Lohmann and Karsten Wolf are supported by the DFG project "Operating Guidelines for Services" (WO 1466/8-1). Jana Koehler and Hagen Völzer were partially supported by the SUPER project (`http://www.ip-super.org`) under the EU 6th Framework Programme Information Society Technologies Objective (contract no. FP6-026850).

References

1. Mendling, J.: Empirical Studies in Process Model Verification. Trans. Petri Nets and Other Models of Concurrency (ToPNoC) **2** (2009) 208–224
2. van der Aalst, W.M.P., Hirnschall, A., Verbeek, H.M.W.E.: An alternative way to analyze workflow graphs. In: CAiSE. Volume 2348 of LNCS, Springer (2002) 535–552
3. Verbeek, H.M.W.E., Basten, T., van der Aalst, W.M.P.: Diagnosing Workflow Processes using Woflan. Comput. J. **44**(4) (2001) 246–279
4. Wolf, K.: Generating Petri net state spaces. In: PETRI NETS 2007. Volume 4546 of LNCS, Springer (2007) 29–42
5. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through SESE decomposition. In: ICSOC 2007. Volume 4749 of LNCS, Springer (2007) 43–55
6. van Dongen, B.F., Jansen-Vullers, M., Verbeek, H.M.W.E., van der Aalst, W.M.P.: Verification of the SAP reference models using EPC reduction, state-space analysis, and invariants. Comput. Ind. **58**(6) (2007) 578–601
7. Mendling, J., Neumann, G., van der Aalst, W.M.P.: Understanding the occurrence of errors in process models based on metrics. In: OTM 2007: CoopIS, DOA, ODBASE, GADA, and IS. Volume 4803 of LNCS, Springer (Nov. 2007) 113–130
8. Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. PhD thesis, Vienna University of Economics and Business Administration (May 2007)
9. Mendling, J., Verbeek, H.M.W.E., van Dongen, B.F., van der Aalst, W.M.P., Neumann, G.: Detection and prediction of errors in EPCs of the SAP reference model. Data Knowl. Eng. **64**(1) (2008) 312–329
10. Wynn, M., Verbeek, H.M.W.E., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Business process verification: Finally a reality! Business Process Management Journal **15**(1) (2009) 74–92
11. Fahland, D.: Translating UML2 activity diagrams to Petri nets. Informatik-Berichte 226, Humboldt-Universität zu Berlin, Berlin, Germany (2008)
12. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge University Press, New York, NY, USA (1995)
13. Kiepuszewski, B., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Fundamentals of control flow in workflows. Acta Inf. **39**(3) (2003) 143–209
14. LoLA v1.11 available at `http://service-technology.org/lola`
15. Valmari, A.: Stubborn sets for reduced state space generation. In: Applications and Theory of Petri Nets. Volume 483 of LNCS, Springer (1989) 491–515
16. UML2oWFN compiler available at `http://service-technology.org/uml2owfn`
17. Murata, T.: Petri nets: Properties, analysis and applications. Proc. of the IEEE **77**(4) (1989) 541–580
18. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In: Business Process Management. Volume 5240 of LNCS, Springer (2008) 100–115