RZ 3694 (# 99704) 07/09/07
Computer Science 16 pages

Research Report

Faster and More Focused Control-Flow Analysis for Business Process Models through SESE Decomposition*

Jussi Vanhatalo[1,2], Hagen Völzer[1], and Frank Leymann[2]

[1]IBM Research GmbH
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

E-mail: {juv,hvo}@zurich.ibm.com

[2]Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstr. 38
D-70569 Stuttgart, Germany

E-mail: frank.leymann@iaas.uni-stuttgart.de

*A shorter version of this report has been published in volume 4749 of Springer Lecture Notes in Computer Science, entitled *Service-Oriented Computing ICSOC 2007*, the Proceedings of the Fifth International Conference on Service-Oriented Computing, Vienna, Austria, September 17-20, 2007, pp. 43-55. Copyright © Springer-Verlag, 2007.

Faster and More Focused Control-Flow Analysis for Business Process Models through SESE Decomposition

Jussi Vanhatalo[1,2], Hagen Völzer[1], and Frank Leymann[2]

[1] IBM Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland
{juv,hvo}@zurich.ibm.com
[2] Institute of Architecture of Application Systems, University of Stuttgart
Universitätsstrasse 38, D-70569 Stuttgart, Germany
frank.leymann@iaas.uni-stuttgart.de

Abstract. We present a technique to enhance control-flow analysis of business process models. The technique considerably speeds up the analysis and improves the diagnostic information that is given to the user to fix control-flow errors. The technique consists of two parts: Firstly, the process model is decomposed into single-entry-single-exit (SESE) fragments, which are usually substantially smaller than the original process. This decomposition is done in linear time. Secondly, each fragment is analyzed in isolation using a fast heuristic that can analyze many of the fragments occurring in practice. Any remaining fragments that are not covered by the heuristic can then be analyzed using any known complete analysis technique.
We used our technique in a case study with more than 340 real business processes modeled with the IBM WebSphere Business Modeler. The results suggest that control-flow analysis of many real process models is feasible without significant delay (less than a second). Therefore, control-flow analysis could be used frequently during editing time, which allows errors to be caught at earliest possible time.

1 Introduction

The quality of a business process model becomes crucial when it is executed directly on a workflow engine or when it is used for generating code that is to be executed. A correct model is also important when one tries to obtain realistic business measures from a process model through simulation. Detecting and fixing errors as early as possible can therefore substantially reduce costs.

The control flow of a business process can be modeled as a *workflow graph* [11, 14]. A workflow graph that has no structural errors such as *deadlocks* or *lack of synchronization* [11] is said to be *sound* [14]. Soundness can and should be checked automatically during the modeling phase. To achieve a high acceptance among the users, the soundness check should

- be as fast as possible and not delay the process of constructing the model — note that a fast soundness check that can be done after each small change of the model allows the user to identify the change that introduced an error — and
- produce useful diagnostic information that helps to locate and fix errors.

When reviewing the techniques currently available for deciding soundness, there seems to be a trade-off between the two requirements. The fastest technique known (cf. [14, 6]) translates the workflow graph into a *free choice Petri net* (cf. [2]) and then decides soundness of that Petri net using the *rank theorem* (cf. [2]). This technique uses time that is cubic in the size of the workflow graph, but does not provide useful diagnostic information. The best diagnostic information is currently provided by a search of the state space of the workflow graph. This can return an execution sequence that leads to the error but it can use time that is exponential in the size of the workflow graph. Esparza [3] (cf. also [2]) provides a technique that can be used to decide soundness in polynomial time (more than cubic) which could potentially provide some diagnostic information, but the latter has not yet been worked out. The analysis tool Woflan [17] can decide soundness and provide diagnostic information, but because the tool ultimately resorts to state space search, it can also take exponential time.

Some authors provide algorithms for deciding soundness for the special case of acyclic workflow graphs. Perumal and Mahanti [10] gave an algorithm that takes quadratic time, which improves on previous approaches for that special case, which were either slower [7] or incomplete [11].

Given any complete technique for deciding soundness from above, we propose two enhancements in this paper. Firstly, we propose to decompose the workflow graph into a tree of *single-entry-single-exit* (*SESE*) *fragments*. This technique is known from compiler theory and can be done in linear time [5]. To check soundness of the workflow graph, one can now check soundness of each fragment in isolation. The overall time used now depends mainly on the size of the largest fragment. We show by experimental evidence on a large number of industrial workflow models that the largest fragment of a workflow graph is usually considerably smaller than the workflow graph itself.

Zerguini [18] and Hauser et al. [4] have proposed similar techniques of deciding soundness through decomposition into fragments. However, they decompose into multiple-entry-multiple-exit (MEME) fragments. These fragments are more general, and include SESE fragments as a special case. This however implies that a fragment can be less intuitive in general. Moreover, their decomposition into fragments is no longer unique and their decomposition algorithms are slower; while Zerguini's algorithm [18] uses quadratic time, the time complexity of the approach of Hauser et al. [4] is unknown, but we conjecture it to be at least quadratic. Both techniques could be used after our fast SESE decomposition.

A nice feature of the decomposition (SESE or MEME) approach is that each error is contained in a fragment. Thus, the error can be shown in a small local context, which in turn should help fixing the error. Errors that are located in disjoint fragments are likely to be independent. Hence, the decomposition also allows multiple independent errors to be detected in one pass.

The second enhancement we propose are two heuristics that can prove soundness or unsoundness of some fragments in linear time. The heuristics are meant to be used before any of the complete techniques from the literature are used, because the latter are likely to be more expensive. The heuristics are based on the observation that many of the fragments found in real process models have a simple structure that can be recognized quickly. The first heuristic uses ideas from Hauser et al. [4].

Note that simple *reduction rules* (e.g. [2, 11, 15]) can also be used to speed up the verification. Usually applied with low cost, they reduce the process model while preserving soundness.

We have implemented our technique and tried it on two libraries of altogether more than 340 industrial process models. 81% of the process models can be completely analyzed with the SESE decomposition and the heuristics alone. For the remaining cases, the analysis task becomes considerably smaller through SESE decomposition.

Mendling et al. [9, 8] have analyzed more than 2000 EPC process models using the Woflan tool [17] for a relaxed version of soundness. We are not aware of any other published case study with large industrial data.

This paper is structured as follows. In Sect. 2, we recall the definition of workflow graphs and their soundness. Section 3 describes our approach in detail. Section 4 presents the results of the case study. This technical report extends a conference paper [16] with an appendix outlining proofs that the paper omits due to lack of space.

2 Sound Workflow Graphs

In this section, we recall the definition of sound workflow graphs [11, 14]. We also give an equivalent characterization of soundness, which will be used later in this paper.

2.1 Workflow Graphs

A *workflow graph* is a directed graph $G = (N, E)$, where a node $n \in N$ is exactly one of the following: a *start node*, a *stop node*, an *activity*, a *fork*, a *join*, a *decision*, or a *merge* such that

1. there is exactly one start node and exactly one stop node; the start node has no incoming edges and exactly one outgoing edge, whereas the stop node has exactly one incoming edge but no outgoing edges;
2. each fork and each decision has exactly one incoming edge and two or more outgoing edges, whereas each join and each merge has exactly one outgoing edge and two or more incoming edges; each activity has exactly one incoming and exactly one outgoing edge;
3. each node $n \in N$ is on a path from the start node to the stop node.

It follows from the definition that no node is directly connected to itself. Figure 1 shows an example of a workflow graph. An activity is depicted as a square, a fork and a join as a thin rectangle, a decision as a diamond, and a merge as a triangle. Start and stop nodes are depicted as (decorated) circles. The unique outgoing edge of the start node is called the *entry edge*, and the unique incoming edge of the stop node is called the *exit edge* of the workflow graph.

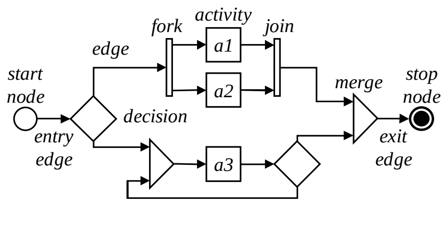

Fig. 1. A workflow graph

The semantics of a workflow graph is, similarly to Petri nets, defined as a token game. A state of a workflow graph is represented by tokens on the edges of the graph. Let $G = (N, E)$ be a workflow graph. A *state* of G is a mapping $s : E \rightarrow \mathbb{N}$, which assigns a natural number to each edge. When $s(e) = k$, we say that edge e carries k *tokens* in state s. The semantics of the various nodes is defined as usual. An activity, a fork, and a join remove one token from each of its ingoing edges and add one token to each of its outgoing edges. A decision node removes a token from its incoming edge, nondeterministically chooses one of its outgoing edges, and adds one token to that outgoing edge. A merge node nondeterministically chooses one of its incoming edges on which there is at least one token, removes one token from that edge, and adds a token to its outgoing edge.

To be more precise, let s and s' be two states and n a node that is neither a start nor a stop node. We write $s \xrightarrow{n} s'$ when s changes to s' by executing n. We have $s \xrightarrow{n} s'$ if

1. n is an activity, fork or join and
$$s'(e) = \begin{cases} s(e) - 1 & e \text{ is an incoming edge of } n, \\ s(e) + 1 & e \text{ is an outgoing edge of } n, \\ s(e) & \text{otherwise.} \end{cases}$$
2. n is a decision and there exists an outgoing edge e' of n such that
$$s'(e) = \begin{cases} s(e) - 1 & e \text{ is an incoming edge of } n, \\ s(e) + 1 & e = e', \\ s(e) & \text{otherwise.} \end{cases}$$
3. n is a merge and there exists an incoming edge e' of n such that
$$s'(e) = \begin{cases} s(e) - 1 & e = e', \\ s(e) + 1 & e \text{ is an outgoing edge of } n, \\ s(e) & \text{otherwise.} \end{cases}$$

Node n is said to be *activated* in a state s if there exists a state s' such that $s \xrightarrow{n} s'$. A state s' is *reachable from* a state s, denoted $s \xrightarrow{*} s'$, if there exists a (possibly empty) finite sequence $s_0 \xrightarrow{n_1} s_1 \ldots s_{k-1} \xrightarrow{n_k} s_k$ such that $s_0 = s$ and $s_k = s'$.

2.2 Soundness

To define *soundness* [14] of a workflow graph G, we use the following notions. The *initial state* of G is the state that has exactly one token on the entry edge and no tokens elsewhere. The *terminal state* of G is the state that has exactly one token on the exit edge and no tokens elsewhere. A *stopping state* of G is a state of G in which the exit edge carries at least one token.

G is *live* if for every state s that is reachable from the initial state, a stopping state is reachable from s. G is *safe* if the terminal state is the only stopping state that is reachable from the initial state. G is *sound* if it is live and safe. The soundness criterion is a global view on correctness. Liveness says that each run can be completed, and safeness says that each completion of a run is a proper termination, i.e., there are no tokens inside the graph upon completion. The workflow graph in Fig. 1 is sound. Figure 2 shows simple examples of unsound graphs. The graph in part (a) is not live, the graph in part (b) is not safe.

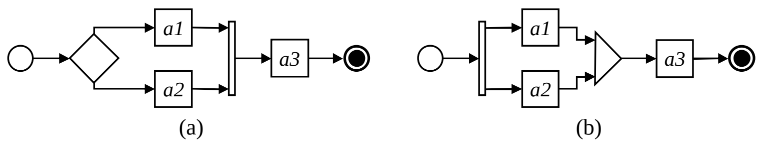

Fig. 2. Structural conflicts: (a) a local deadlock (b) a lack of synchronization

The two examples of unsound workflow graphs in Fig. 2 are examples of a *structural conflict*, viz. a *local deadlock* (part a) and a *lack of synchronization* (part b) [11]. A *local deadlock* is a state s such that there exists a join n where (i) at least one incoming edge of n carries a token in s and (ii) there is an incoming edge e of n such that e does not carry a token in any state s' that is reachable from s. That is, that join will never get 'enough' tokens. A state s of G has *lack of synchronization* if there is a merge n such that more than one incoming edge of n carries a token, i.e., that merge gets 'too many' tokens. Note that a lack of synchronization can lead to a state where there is more than one token on a single edge. Van der Aalst et al. [14] have shown that for acyclic workflow graphs, soundness is equivalent with the condition that neither a local deadlock nor a state with lack of synchronization is reachable from the initial state. We generalize this here for arbitrary workflow graphs, therefore providing a local view of correctness for arbitrary workflow graphs.

Definition 1. *Let G be a workflow graph. G is* locally live *if there is no local deadlock that is reachable from the initial state. G is* locally safe *if no state is reachable from the initial state that has more than one token on a single edge.*

Theorem 1. *A workflow graph is sound if and only if it is locally safe and locally live.*

3 Enhanced Control-Flow Analysis

In this section, we explain the decomposition of a workflow graph into SESE fragments and show how some fragments can be quickly recognized as sound or unsound.

3.1 Decomposition into Fragments

Figure 3 shows a workflow graph and its decomposition into *SESE fragments* (cf. e.g. [5]). A SESE fragment is depicted as a dotted box. Let $G = (N, E)$ be a workflow graph. A *SESE fragment* (*fragment* for short) $F = (N', E')$ is a nonempty *subgraph* of G, i.e., $N' \subseteq N$ and $E' = E \cap (N' \times N')$ such that there exist edges $e, e' \in E$ with $E \cap ((N \setminus N') \times N') = \{e\}$ and $E \cap (N' \times (N \setminus N')) = \{e'\}$; e and e' are called the *entry* and the *exit* edge of F, respectively.

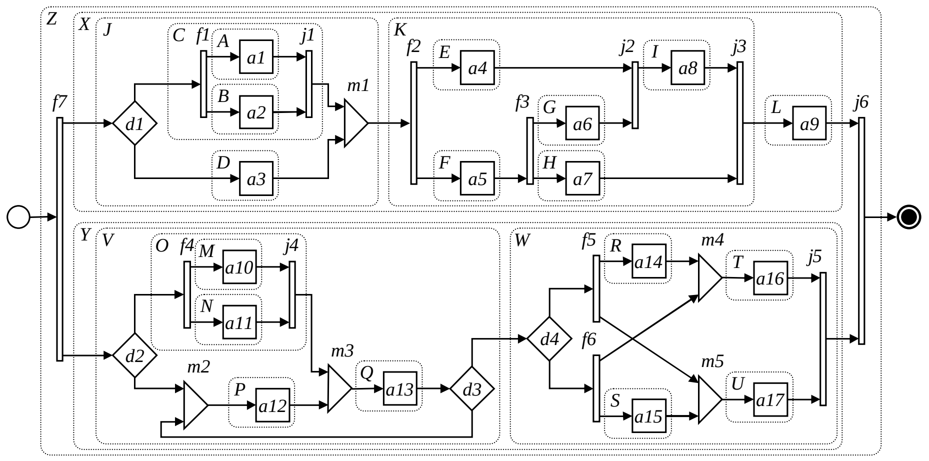

Fig. 3. Decomposition of a workflow graph into canonical fragments

The workflow graph shown in Fig. 3 has more fragments than those that are shown explicitly. For example, the union of fragments J and K, denoted $J \cup K$, as well as $K \cup L$ are fragments. Those however are not of interest here and they are subsumed in fragment X. Interesting fragments will be called *canonical*, which are defined in the following. We say that two fragments F and F' are *in sequence* if the exit edge of F is the entry edge of F' or vice versa. The union $F \cup F'$ of two fragments F and F' that are in sequence is a fragment again. A fragment F is *non-canonical* if there are fragments X, Y, Z such that X and Y are in sequence, $F = X \cup Y$, and F and Z are in sequence; otherwise F is said to be *canonical*.

The fragments shown in Fig. 3 are exactly the canonical fragments of that workflow graph. Canonical fragments do not overlap. Two canonical fragments are either nested or disjoint [5]. Therefore, it is possible to organize the canonical fragments in a unique tree, similarly to the Program Structure Tree shown in [5]. We call this tree the *process structure tree* of a workflow graph. It can be computed in time linear in the size of the workflow graph [5][3]. As we are only interested in canonical fragments, we mean 'canonical fragment' whenever we say 'fragment' in the following.

Figure 4 shows the process structure tree of the workflow graph from Fig. 3. A fragment is represented as a boxed tree node. In addition, we represent the nodes of the workflow graph as leaves in the tree. The *parent* of a fragment F (a workflow graph node n) is the smallest fragment F' that contains F (n). Then, we also say that F is a *child fragment* of F' (n is a *child node* of F').

To check the soundness of a workflow graph, it is sufficient to analyze the soundness of its fragments in isolation. Note that a fragment can be viewed as a workflow graph by adding entry and exit edges as well as a start and a stop node. Hence we can apply the notion of soundness also to fragments. The following theorem follows from classical Petri net theory (e.g. [12], cf. also [13, 14, 18]).

[3] Note: Ananian [1] gives a slightly modified linear time algorithm that includes corrections.

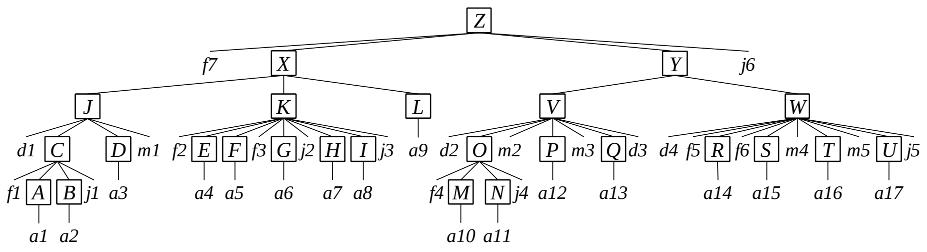
Z
X
Y
f7
j6
J
K
L
V
W
d1 C D m1 f2 E F f3 G j2 H I j3 a9 d2 O m2 P m3 Q d3 d4 f5 R f6 S m4 T m5 U j5
f1 A B j1 a3 a4 a5 a6 a7 a8 f4 M N j4 a12 a13 a14 a15 a16 a17
a1 a2 a10 a11

Fig. 4. The process structure tree of the workflow graph in Fig. 3

Theorem 2. *A workflow graph is sound if and only if all its child fragments are sound and the workflow graph that is obtained by replacing each child fragment with an activity is sound.*

Checking soundness of fragments can therefore be done along the structure of the process structure tree, starting from the leaves upwards. If a fragment *F* was checked for soundness, checking soundness of the parent fragment (in the tree) can abstract from the internal structure of *F*, i.e., *F* can be treated as an activity in the parent fragment. Figure 5 shows fragments *J* and *V* from Figs. 3 and 4, where fragment *J* abstracts from the structure of the child fragments *C* and *D* and fragment *V* abstracts from the structure of fragment *O*.

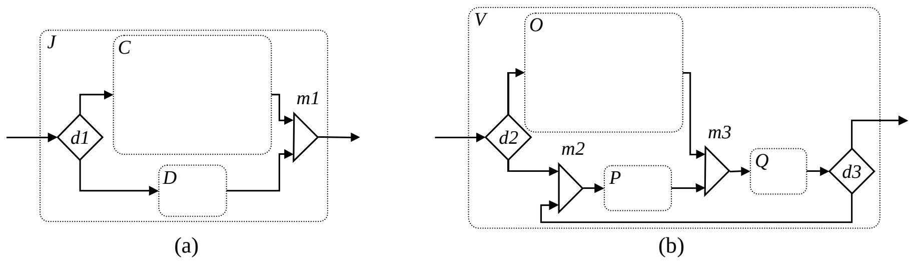

Fig. 5. Fragments *J* and *V* ignoring the structure of their child fragments

3.2 Heuristic for Sound Fragments

Many fragments that occur in practice have a simple structure that can easily be recognized, which identifies those fragments as being sound. To this end, we define the following categories, based on definitions given by Hauser et al. [4].

Definition 2. *Let F be a fragment of a workflow graph. F is*

1. well-structured *if it satisfies one of the following conditions:*

- *F has no decisions, merges, forks or joins as children in the process structure tree (*sequence*),*
- *F has exactly one decision and exactly one merge, but no forks and no joins as children. The entry edge of F is the incoming edge of the decision, and the exit edge of F is the outgoing edge of the merge (*sequential branching*),*
- *F has exactly one decision and exactly one merge, but no forks and no joins as children. The entry edge of F is an incoming edge of the merge, and the exit edge of F is an outgoing edge of the decision (*cycle*),*
- *F has exactly one fork, exactly one join, no decisions and no merges as children. The entry edge is the incoming edge of the fork. The exit edge is the outgoing edge of the join. (*concurrent branching*).*

2. *an* unstructured concurrent *fragment if F is not well-structured, contains no cycles, and has no decisions and no merges as children.*
3. *an* unstructured sequential *fragment if F is not well-structured and has no forks and no joins as children.*
4. *a* complex *fragment if it is none of the above.*

It is easy to see that it can be decided in linear time to which of the four categories listed above a fragment belongs.

Theorem 3. *If a fragment F is well-structured, an unstructured concurrent, or an unstructured sequential fragment, then F is sound if and only if all its child fragments are sound.*

This theorem was already observed by Hauser et al. [4]. Note that all fragment categories ignore the structure of child fragments, taking only the top-level structure into account. In Fig. 3, fragments *X* and *Y* are well-structured (sequence) and so are also fragments *C*, *O*, *Z* (concurrent branching) and *J* (sequential branching). Fragments *K* and *V* are examples of unstructured concurrent and unstructured sequential fragments, respectively. Note that unstructured sequential fragments may contain cycles, whereas unstructured concurrent fragments must not.

A complex fragment may be sound or unsound. Fragment *W* in Fig. 3 is a sound complex fragment. It follows from Theorems 2 and 3 that the entire workflow graph in Fig. 3 is sound.

3.3 Heuristic for Unsound Fragments

Some complex fragments can be efficiently determined as not being sound:

Theorem 4. *A complex fragment F is not sound if it satisfies one of the following conditions:*

1. *F has one or more decisions (merges), but no merges (decisions) as children in the process structure tree,*
2. *F has one or more forks (joins), but no joins (forks) as children,*
3. *F contains a cycle, but has no decisions or no merges as children.*

It is again easy to see that this heuristic can be applied in linear time. We actually found numerous errors in real process models using this heuristic (see Sect. 4.2). The relative strength of this heuristic is due to the fact that, similar to the heuristic in Sect. 3.2, the structure of child fragments is ignored.

4 Case Study

In this section, we describe the results of an application of our proposed technique in a case study with industrial data.

4.1 The Data

We have analyzed the soundness of more than 340 workflow graphs that were extracted from two libraries of industrial business processes modeled in the IBM WebSphere Business Modeler. Although the modeling language used there is more expressive than workflow graphs, it was possible to translate the process models into workflow graphs because strict guidelines were used for the construction of these process models. The description of the translation is beyond the scope of this paper.

Library 1 consists of more than 140 processes. The extracted workflow graphs have, on average, 67 edges, with the maximum being 215. Library 2 is an experimental extension of Library 1. It contains similar processes, but many features were added to the processes and also some processes were added. It contains more than 200 processes, the extracted workflow graphs have 99 edges on average, with the maximum being 342.

4.2 The Results

We analyzed the libraries using an IBM ThinkPad T43p laptop that has a 2.13 GHz Intel Pentium M processor and 2 GB of main memory. The entire Library 1 is analyzed in 9 seconds, and Library 2 in 15 seconds. Thus, the average analysis time per workflow graph is less than 0.1 seconds.

SESE Decomposition As described in Sect. 1, the worst-case time a complete technique needs for checking the soundness of a workflow graph can be polynomial or exponential in the size g of the workflow graph, which is defined to be its number of edges. Similarly, the size of a fragment is defined as its number of edges plus 2 (for the entry edge and the exit edge). If we use a complete technique after the SESE decomposition according to the procedure in Sect. 3.1, the time used is linear in the number of fragments. Note also that the number of fragments in a workflow graph is at most twice the number of nodes. The overall time used therefore mainly depends on the size $f_{\max}$ of

Table 1. Graph size (i.e., number of edges) compared to the size of the largest fragment in the graph and size reductions for the workflow graphs in Library 1

	Graph size g	Largest fragment size $f_{\max}$	Reduction $g - f_{\max}$	Reduction $g/f_{\max}$
Maximum	215	51	191	9.0
Average	67	24	44	2.8
Minimum	11	11	0	1.0

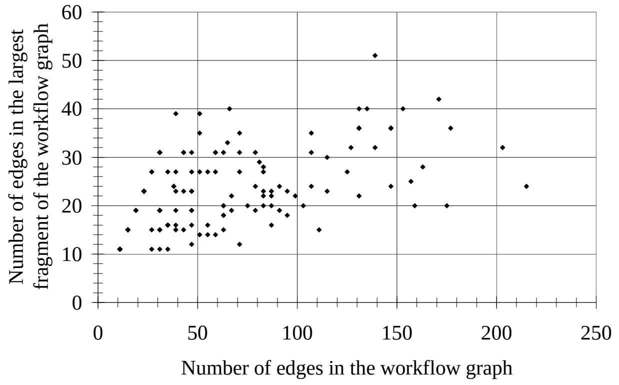

Fig. 6. Size of largest fragment in relation to graph size for all workflow graphs in Library 1

the largest fragment to which we have to apply the complete technique. If the complete technique uses polynomial time g^c for some constant c, then the reduction that SESE decomposition could achieve is $g^c/f^c_{\max} = (g/f_{\max})^c$. If the complete technique uses exponential time c^g, then the possible reduction is $c^g/c^{f_{\max}} = c^{g-f_{\max}}$. Table 1 shows the values for $g/f_{\max}$ and $g - f_{\max}$ for Library 1 as an indication of the reduction achieved due to SESE decomposition.

Figure 6 shows the largest fragment size in relation to the graph size for each workflow graph in Library 1. It shows that the graph size has only a minor impact on the largest fragment size. Therefore, the reduction increases as the graph size increases. Thus, the technique is most useful when the complete techniques would be most time consuming. Even a small reduction can be significant, as the complete techniques for checking soundness can take a time that is cubic or exponential in the graph size.

Table 2 shows the reduction statistics for the workflow graphs in Library 2. The graphs are larger, and also the reduction is higher.

Table 2. Graph size compared to the size of the largest fragment in the graph, and size reductions for Library 2

	Graph size g	Largest fragment size $f_{\max}$	Reduction $g - f_{\max}$	Reduction $g/f_{\max}$
Maximum	342	82	328	24.4
Average	99	21	78	5.6
Minimum	12	6	5	1.5

Using both heuristics from Sect. 3, we can decide soundness for 68.5% of the workflow graphs in Library 2. For the remaining graphs, our prototype tool highlights the complex fragments that may be unsound. A complete analysis method is needed to

Table 3. Library 2: Graph size, largest fragment size, and reduction for the remaining 31.5% of workflow graphs for which soundness is unknown after applying our heuristics

	Graph size g	Largest fragment size $f_{\max}$	Reduction $g - f_{\max}$	Reduction $g/f_{\max}$
Maximum	334	82	284	10.4
Average	126	32	94	4.3
Minimum	40	12	25	1.6

decide their soundness, or they can be reviewed manually. The reduction statistics for these remaining workflow graphs are shown in Table 3.

Fragment Categories Even though our heuristics from Sect. 3 are incomplete, we were able to decide soundness for all the workflow graphs from Library 1. They are all sound.

The first column in Table 4 illustrates the distribution of fragments according to the categories defined in Sects. 3.2-3.3 for Library 1. We excluded here any fragments that are well-structured sequences from these statistics, because most fragments are sequences and those are trivially sound and thus not interesting.

We can also put entire workflow graphs into the various categories. For example, a workflow graph is *complex* if it has at least one complex fragment. Complex graphs are further divided into those known to be not sound by applying the heuristic in Sect. 3.3 and those for which soundness is unknown. A workflow graph is *unstructured* if it has at least one unstructured fragment and no complex fragments. Otherwise, a graph has only *well-structured* fragments and it is therefore called *well-structured*. Column 3 of Table 4 shows the distribution of workflow graphs in Library 1 in the various categories. The last two columns present the same statistics for Library 2.

Table 4. Categories of fragments and workflow graphs in the libraries

Fragment category / Workflow graph category	Library 1 Percentage of fragments	Percentage of graphs	Library 2 Percentage of fragments	Percentage of graphs
Well-structured (sound)	54.8%	37.5%	65.4%	33.3%
Unstructured (sound)	45.2%	62.5%	14.9%	23.1%
- Unstructured concurrent	1.4%	-	6.0%	-
- Unstructured sequential (acyclic)	29.2%	-	4.4%	-
- Unstructured sequential (cyclic)	14.6%	-	4.6%	-
Complex	0.0%	0.0%	19.7%	43.5%
- Complex (not sound)	0.0%	0.0%	5.4%	12.0%
- Complex (soundness unknown)	0.0%	0.0%	14.3%	31.5%

Most fragments are well-structured, which makes it attractive to analyze fragments separately. However, only a third of the workflow graphs are well-structured and there is a considerable number of sound unstructured workflow graphs. Therefore, although well-structuredness is also an appealing correctness requirement, it seems to be overly restrictive. As unstructured fragments occur often, it makes sense to detect those with fast heuristics before using a complete analysis technique. Our heuristics can decide soundness not only for many fragments, but also for a significant proportion of the workflow graphs.

In Library 2, 43.5% of the workflow graphs contain at least one complex fragment. Only one workflow graph has more than one complex fragment. 19.7% of the fragments in Library 2 are complex fragments. Our heuristic recognized 27.3% of these fragments as being unsound. We have not yet checked the soundness of the remaining complex graphs by integrating our tool with a complete analysis method. The high error rate in Library 2 is due to its experimental nature.

5 Conclusion

We proposed a technique to focus and speed up control-flow analysis of business process models that is based on decomposition into SESE fragments. The SESE decomposition could also be used for other purposes such as browsing and constructing large processes, discovery of reusable subprocesses, code generation, and others.

We also proposed a partition of the fragments into various categories, which can be computed fast. We think that tagging a fragment with its category may help to better understand the process model and may help to establish modeling patterns. It also helps to speed up the control-flow analysis as many of the correct fragments that occur in practice have a simple structure.

We plan to integrate our prototype with existing complete verification techniques and measure the impact of SESE decomposition on the analysis time. In addition, we plan to investigate the errors that occur in Library 2, together with approaches to fix them.

Acknowledgments We thank Michael Friess for suggesting to apply SESE decomposition to workflow graphs. We thank Wil van der Aalst, Rainer Hauser, Rania Khalaf, Jana Koehler, Oliver Kopp, Jochen Küster and Ksenia Ryndina for helpful discussions and comments.

The work published in this article was 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. C. Scott Ananian. The static single information form. Master's thesis, Massachusetts Institute of Technology, September 1999.
2. Jörg Desel and Javier Esparza. *Free Choice Petri Nets*. Cambridge University Press, 1995.

3. Javier Esparza. Reduction and synthesis of live and bounded free choice Petri nets. *Inf. Comput.*, 114(1):50–87, 1994.
4. Rainer Hauser, Michael Friess, Jochen M. Küster, and Jussi Vanhatalo. An incremental approach to the analysis and transformation of workflows using region trees. *IEEE Transactions on Systems, Man, and Cybernetics - Part C*, to appear, also available as IBM Research Report RZ 3693, June 2007.
5. Richard Johnson, David Pearson, and Keshav Pingali. The program structure tree: Computing control regions in linear time. In *Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation (PLDI)*, pages 171–185, 1994.
6. Peter Kemper. Linear time algorithm to find a minimal deadlock in a strongly connected free-choice net. In *Proceedings of the 14th International Conference on Application and Theory of Petri Nets*, volume 691 of *Lecture Notes in Computer Science*, pages 319–338. Springer, 1993.
7. Hao Lin, Zhibiao Zhao, Hongchen Li, and Zhiguo Chen. A novel graph reduction algorithm to identify structural conflicts. In *Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS-35 2002)*, page 289, 2002.
8. Jan Mendling. *Detection and Prediction of Errors in EPC Business Process Models*. PhD thesis, Vienna University of Economics and Business Administration (WU Wien), Austria, May 2007.
9. Jan Mendling, Michael Moser, Gustaf Neumann, H. M. W. Verbeek, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Faulty EPCs in the SAP reference model. In *Proceedings of the 4th International Conference Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 451–457. Springer, 2006.
10. Sinnakkrishnan Perumal and Ambuj Mahanti. A graph-search based algorithm for verifying workflow graphs. In *Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA 2005)*, pages 992–996. IEEE Computer Society, 2005.
11. Wasim Sadiq and Maria E. Orlowska. Analyzing process models using graph reduction techniques. *Inf. Syst.*, 25(2):117–134, 2000.
12. Robert Valette. Analysis of Petri nets by stepwise refinements. *Journal of Computer and System Sciences*, 18(1):35–46, 1979.
13. Wil M. P. van der Aalst. Workflow verification: Finding control-flow errors using Petri-net-based techniques. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 161–183, London, UK, 2000. Springer-Verlag.
14. Wil M. P. van der Aalst, Alexander Hirnschall, and H. M. W. (Eric) Verbeek. An alternative way to analyze workflow graphs. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE 2002)*, volume 2348 of *Lecture Notes in Computer Science*, pages 535–552. Springer, 2002.
15. Boudewijn F. van Dongen, Wil M. P. van der Aalst, and H. M. W. Verbeek. Verification of EPCs: Using reduction rules and Petri nets. In *Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE 2005)*, volume 3520 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2005.
16. Jussi Vanhatalo, Hagen Völzer, and Frank Leymann. Faster and more focused control-flow analysis for business process models though SESE decomposition. In Bernd Krämer, KJ Lin, and Priya Narasimhan, editors, *5th International Conference on Service-Oriented Computing (ICSOC)*, volume 4749 of *Lecture Notes in Computer Science*, pages 43–55. Springer-Verlag Berlin Heidelberg, September 2007.
17. H. M. W. (Eric) Verbeek, Twan Basten, and Wil M. P. van der Aalst. Diagnosing workflow processes using Woflan. *Comput. J.*, 44(4):246–279, 2001.
18. Loucif Zerguini. A novel hierarchical method for decomposition and design of workflow models. *Journal of Integrated Design and Process Science*, 8(2):65–74, 2004.

6 Appendix - Proofs

Note. This technical report extends a conference paper [16] with this appendix that outlines the proofs that were omitted from the conference paper due to lack of space.

First, we extend the definition of an *activated* node to cover the stop node. Stop node n is said to be *activated* in a state s if the incoming edge e of the stop node carries at least one token in state s, $s(e) \geq 1$. This extension is needed for the following lemma that we use frequently in the sequel.

Lemma 1. *Let G be a workflow graph that is locally live, s a state that is reachable from the initial state, e and edge that carries a token in s and n a node of G. If there is a path from e to n in G, then there exists a state s' such that n is activated in s'.*

Proof. The claim is proven by induction over the length of the path. The base case is only not trivial when n is a join. Then, the claim directly follows from local liveness. The induction step uses the same argument.

Theorem 1. *A workflow graph is sound if and only if it is locally safe and locally live.*

Proof. Let G be a workflow graph.

$\Rightarrow$ Let G be live and safe.
  1. We have to show that G is locally live. Let s be a local deadlock that is reachable from the initial state and e be an edge that carries a token; Since G is live, a stopping state s' can be reached from s. Since G is safe, s must be the terminal state which contradicts s being a local deadlock.
  2. We have to show that G is locally safe. This follows from a corresponding results on Petri nets: Every sound free choice workflow net is 1-safe (cf. [14, 2]).

$\Leftarrow$ Let G be locally live and locally safe.
  1. We have to show that G is live. Let s be a state that is reachable from the initial state. It is clear from the definitions of workflow graphs and their semantics that their exists an edge e such that e carries a token in s. The edge e lies on a path from the entry edge to the exit edge. Since G is locally live, that token can be moved to the exit edge yielding a stopping state (Lemma 1).
  2. We have to show that G is safe. Suppose s is a stopping state that is reachable from the initial state such that besides one token on the exit edge, there is an edge e carrying another token; e could also be the exit edge. Since G is locally live, we can move that second token to the exit edge (Lemma 1) hence obtaining a state with two tokens on the exit edge. That contradicts G being locally safe.

Theorem 2. *A workflow graph is sound if and only if all its child fragments are sound and the workflow graph that is obtained by replacing each child fragment with an activity is also sound.*

Proof. When a workflow graph is translated to a free choice Petri net in a standard way [14], the workflow graph is sound if and only if the Petri net is sound. This theorem then becomes a theorem for Petri nets, which is well known. It can be easily derived from the results of the classic paper by Valette [12]. Explicit proofs of stronger theorems are given by van der Aalst [13] and Zerguini [18].

Theorem 3. *If a fragment F is well-structured, an unstructured concurrent, or an unstructured sequential fragment, then F is sound if and only if all its child fragments are sound.*

Proof. Let F be a fragment. Suppose that all child fragments are sound. Then they can be replaced by activities resulting in a workflow graph, which we call G. By Thm. 2, we have that F is sound if G is sound. Conversely if a child fragment is not sound, it follows by Thm. 2 that F is unsound. We distinguish the following cases:

1. Let G be unstructured sequential. Since there are no joins and forks there is always exactly one token inside the fragment. It directly follows that the fragment is sound.
2. Let G be unstructured concurrent. Since G is acyclic, the edges of G are partially ordered. Local safeness with respect to an edge can be proven by induction over the distance of the edge from the entry edge. Local liveness is shown by proving that a token on some edge e can be removed from e, which in turn can be shown by induction over the distance of e to the exit edge.
3. The well-structured cases are special cases of the corresponding unstructured cases.

Theorem 4. *A complex fragment F is not sound if it satisfies one of the following conditions:*

1. F has one or more decisions (merges), but no merges (decisions) as children in the process structure tree,
2. F has one or more forks (joins), but no joins (forks) as children,
3. F contains a cycle, but has no decisions or no merges as children.

Proof. Let F be a complex fragment. Suppose that F is sound.

3. (a) Let F contain a cycle, but no decisions. There exists a maximal strongly connected component O that contains this cycle. Since there is no decision, there must be a fork f and edges e and e' such that
   - f belongs to O,
   - e is outgoing from f leading to a node inside O,
   - e' is outgoing from f leading to a node outside O.

   Since F is sound, f can be activated and executed (Lemma 1). In the resulting state, there is a token in e and e'. Since there is a path from e to f and F is sound, a state can be reached where f is activated again (Lemma 1). This can be done without removing the token on e', because e' is outside O. When f is executed again, we obtain a state with two tokens on e', which contradicts soundness.

   (b) Let F contain a cycle, but no merges. There exists a maximal strongly connected component O that contains this cycle. In the initial state, no edge in O carries a token. Since F is sound, it follows from Lemma 1 that for each edge there exists a reachable state, where this edge carries a token. Let e be an edge and s, s' be states and n be a node such that
   - e is inside O,
   - s is reachable from the initial state,
   - no edge inside O carries a token in s,

- $s \xrightarrow{n} s'$,
- e carries a token in s'.

As executing n can only produce a token in an outgoing edge of n, e is an outgoing edge of n. Since e is inside O, also n is inside O. Since n is inside O, there exists an incoming edge e' of n such that e' is inside O. n cannot be activated in s, because there are no merges and e' carries no token in s. This contradicts soundness.

1. (a) Let F have at least one decision d, but no merges. If F is cyclic and has no merges, it is unsound as proven above in case 3 (b). Therefore, we assume F has no cycles. Decision d has at least two outgoing edges which have distinct paths to the exit edge. These paths have non-empty disjoint parts in the beginning. Since F has no cycles, there is a node, which is the first node that merges the disjoint paths. This node is either a merge or a join. Since there is no merge, both paths merge in a common join j. Since F is sound, there is a state s that is reachable from the initial state that activates d. When d is executed, we have only one token on one of the disjoint paths from the decision to the join. Because F is acyclic and safe, d is executed only once. Since there are no merges on these disjoint paths, the last edge on a path can get a token only if first edge on that path gets one. Thus, the join can get a token only from one of these paths. This leads into a local deadlock, which contradicts soundness.

   (b) Let F have at least one merge, but no decisions. If F is cyclic and has no decisions, it is unsound as proven above in case 3 (a). Therefore, we assume F has no cycles. The merge has at least two incoming edges which have distinct paths from the entry edge. Since F has no cycles, there is a node, which is the last node that splits the distinct paths into non-empty disjoint parts in the end. This node is either a decision or a fork. Since there is no decision, both paths split in a common fork f. Since F is sound, there is a state s that is reachable from the initial state that activates f. When f is executed, we have two tokens, each being on a different disjoint path from the fork to the merge. Because these paths are disjoint, F is sound and there is no decision, we can move the two tokens up to the merge, independently of each other. The merge can be executed twice resulting in a state violating local safeness.

2. (a) Let F have at least one fork, but no joins. Since there is a fork and F is sound, the fork can be activated and thus there is a reachable state, where at least two edges carry a token. As there are no joins, moving one of the tokens to the exit edge (Lemma 1) cannot require moving the other token(s). Thus, there is a reachable stopping state, which is not the terminal state, which contradicts soundness.

   (b) Let F have at least one join, but no forks. Since there is no fork, F has never more than one token. Since F is sound, the join can be activated, which however needs more than one token — a contradiction.