Data & Knowledge Engineering 101 (2016) 43–61

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Converting unstructured into semi-structured process models

Rik Eshuis [a,*], Akhil Kumar [b]

[a] Eindhoven University of Technology, School of Industrial Engineering, P.O. Box 513, 5600MB Eindhoven, The Netherlands
[b] Department of Supply Chain and Information Systems, Smeal College of Business, Penn State University, University Park, PA 16802, USA

ARTICLE INFO

Article history:
Received 7 March 2014
Received in revised form 24 August 2015
Accepted 22 October 2015
Available online 6 November 2015

Keywords:
Business process management
Process models
Semi-structured processes
Model transformation

ABSTRACT

Business process models capture process requirements that are typically expressed in unstructured, directed graphs that specify parallelism. However, modeling guidelines or requirements from execution engines may require that process models are structured in blocks. The goal of this paper is to define an automated method to convert an unstructured process model containing parallelism into an equivalent semi-structured process model, which contains blocks and synchronization links between parallel branches. We define the method by means of an algorithm that is based on dominators, a well-known technique from compiler theory for structuring sequential flow graphs. The method runs in polynomial time. We implemented and evaluated the algorithm extensively. In addition we compared the method in detail with the BPStruct method from literature. The comparison shows that our method can handle cases that BPStruct is not able to and that the method coincides with BPStruct for the cases that BPStruct is able to handle.

1. Introduction

Business processes management (BPM) [1,2] focuses on the automation of business processes using middleware information technology. Business process models play a key role in BPM. They can be conceptual-level designs of business processes, but they can also specify the coordination logic for execution engines that support the operational management of business processes. Communication of business process models between different stakeholders, end users or system engineers is of utmost importance to realize a successful implementation.

An important subclass of process models are structured process models, which consist of nested blocks and each block has a single entry point and a single exit point. Examples of structured languages are the Business Process Execution Language (BPEL) [3] and OWL-S [4], a language for describing semantic web services using the Web Ontology Language (OWL). Structured process models are also required by advanced BPM techniques such as process views in inter-organizational collaborations [5,6], rule-driven implementations of process models [7], and dynamic changes of running processes [8].

A structured process model is similar to a (parallel) program without goto statements. While every unstructured program has a structured equivalent [9], this is not true for unstructured process models, since synchronization links between parallel blocks cannot be expressed in a structured process model [10,11]. Therefore, established BPM languages like BPEL [3] and Adept [8] allow semi-structured process models, which are structured into blocks with additional synchronization links between parallel blocks.

Structured process models are easier to read and understand than unstructured models [12] and contain fewer errors [13]. Structuredness has been proposed as a modeling guideline [14]. But business processes are typically modeled in graph-based

* Corresponding author.
E-mail addresses: h.eshuis@tue.nl (R. Eshuis), akhilkumar@psu.edu (A. Kumar).

http://dx.doi.org/10.1016/j.datak.2015.10.003
0169-023X/© 2015 Elsevier B.V. All rights reserved.

notations such as BPMN [15] or UML activity diagrams [16] that do not syntactically enforce structuredness. A graphical representation enables easy communication with end-users in the organization in which the process is used. Nodes are either activities (tasks) or routing constructs like a choice split or a parallel join while edges represent ordering constraints.

In a structured process graph, each split has a matching join, and the split and join demarcate a subprocess that corresponds to a single entry, single exit block. Since a split does not need to have a matching join, process graphs are usually not structured into blocks. Therefore translations have been proposed to structure graph-based process models [11,17,18]. But these approaches do no construct semi-structured process models.

We aim to define an automated method for converting an unstructured process model into an equivalent semi-structured process model. The method is described as a formal algorithm, using well-known concepts from the field of compiler design [19] for structuring control flow graphs. However, a control flow graph only specifies sequential behavior while a process model also specifies parallel behavior, which complicates the structuring procedure. Another difference is that the approach allows the identification of semi-structured process models in which parallel branches use cross-synchronization.

For every input process model, the method delivers output, but the result is only meaningful (i.e., the output process is equivalent to the input process) if the input process model is correct, i.e., it is free of deadlocks and there is no lack of synchronization (see Section 3). Moreover, the approach only outputs a semi-structured process model if no equivalent structured process model exists.

Compared to existing approaches for structuring unstructured process models [11,17,18], our method is more powerful, since it can generate both structured and semi-structured processes, while existing approach can only generate structured processes and fail for correct process models that have no equivalent structured process. The overall time complexity of the method is polynomial, which ensures that the method scales well to large process models. Other approaches are either as efficient but less powerful [11,17] or have a much higher time complexity [18]. An extensive discussion of related work can be found in Section 7.

To simplify the presentation, we restrict ourselves in the main text to acyclic process models. In the Appendix, we extend the algorithms of the main text to deal with cyclic process models in which each loop has a single entry and a single exit point. Other papers [20–22] already discuss techniques to turn a process model with unstructured loops into an equivalent process model in which each loop has a single entry and a single exit point. These techniques are complementary to our method and can be easily integrated.

The rest of this paper is organized as follows. Section 2 presents a motivating example. Section 3 defines process flow graphs, which formalize business process models, and a functional notation for representing semi-structured processes. To simplify the definition of the structuring algorithm, we apply in Section 4 preprocessing steps to the input process flow graphs. Section 5 defines the structuring algorithm that transforms a process flow graph into an equivalent semi-structured process. The correctness of the algorithm is also analyzed in Section 5. To simplify the exposition, we consider acyclic process flow graphs in Sections 4 and 5. Appendix A explains how the algorithm extends to process flow graphs with loops. Section 6 discusses evaluation of the method. To evaluate feasibility, we have implemented a prototype implementation of the method. To evaluate the utility, we have applied the method to several benchmark examples taken from the literature. Section 7 discusses related work. Finally, Section 8 concludes this paper and gives directions for further work.

2. Motivating example

We motivate the approach with an example process model shown in Fig. 1. The notation is explained in the next section. The process is about handling insurance claims for damaged vehicles. Each received claim is assessed. For small amounts, the claim is accepted without further ado. For large payment amounts, the vehicle is inspected and the payment is determined and approved by a manager. In both cases, the claim is paid and in parallel a decision letter is sent, where the decision is subject to the manager approval for large amounts. In parallel, each assessed claim is monitored and a report is created. Finally, the claim is archived if all previous tasks have been completed.

The resulting process flow graph in Fig. 1 cannot be converted into a structured process since there is a cross-synchronization between two parallel branches (arrow from AND split A3 to AND join A4). Therefore, all existing approaches for structuring process flow graphs [11,17,18] fail for this example.

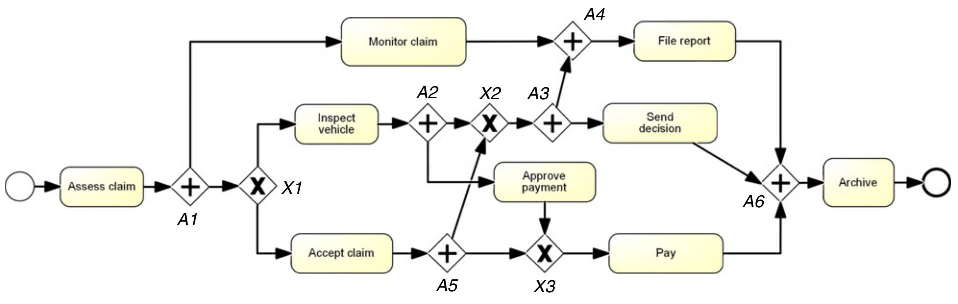

Fig. 1. Process flow graph.