Contents lists available at ScienceDirect

Data & Knowledge Engineering

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

The refined process structure tree

Jussi Vanhatalo [a,b,1], Hagen Völzer [a,*], Jana Koehler [a]

[a] *IBM Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland*
[b] *Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstrasse 38, D-70569 Stuttgart, Germany*

ARTICLE INFO

Article history:
Available online 9 March 2009

Keywords:
Workflow management
Workflow graph parsing
Model decomposition
Subprocess detection
Graph theory

ABSTRACT

We consider a workflow graph as a model for the control flow of a business process and study the problem of workflow graph parsing, i.e., finding the structure of a workflow graph. More precisely, we want to find a decomposition of a workflow graph into a hierarchy of sub-workflows that are subgraphs with a single entry and a single exit of control. Such a decomposition is the crucial step, for example, to translate a process modeled in a graph-based language such as BPMN into a process modeled in a block-based language such as BPEL. For this and other applications, it is desirable that the decomposition be unique, *modular* and as fine as possible, where *modular* means that a local change of the workflow graph can only cause a local change of the decomposition. In this paper, we provide a decomposition that is unique, modular and finer than in previous work. We call it the *refined process structure tree*. It is based on and extends similar work for sequential programs by Tarjan and Valdes [ACM POPL '80, 1980, pp. 95–105]. We give two independent characterizations of the refined process structure tree which we prove to be equivalent: (1) a simple descriptive characterization that justifies our particular choice of the decomposition and (2) a constructive characterization that allows us to compute the decomposition in linear time. The latter is based on the tree of triconnected components (elsewhere also known as the SPQR tree) of a biconnected graph.

1. Introduction

The control flow of a business process can often be modeled as a *workflow graph* [1]. Workflow graphs capture the core of many business process languages such as UML activity diagrams, BPMN and EPCs. We study the problem of *parsing* a workflow graph, that is, decomposing the workflow graph into a hierarchy of sub-workflows that have a single entry and a single exit of control, often also called *blocks*, and labeling these blocks with a syntactical category they belong to. Such categories are *sequence*, *if*, *repeat-until*, etc., see Fig. 1a. Such a decomposition is also called a *parse* of the workflow graph. It can also be shown as a *parse tree*, see Fig. 1c.

The parsing problem occurs when we want to translate a graph-based process description (e.g. a BPMN diagram) into a block-based process description (e.g. BPEL process), but there are also other use cases for workflow graph parsing. For example, Vanhatalo, Völzer and Leymann [2] show how parsing speeds up control-flow analysis. Küster et al. [3] show how differences between two process models can be detected and resolved based on decompositions of these process models. Gschwind et al. [4] use parsing for pattern-based editing operations of process models. We believe that parsing also helps in understanding large processes and in finding reusable subprocesses.

* Corresponding author. Tel.: +41 44 724 8395; fax: +41 44 724 8953.
E-mail addresses: jussi.vanhatalo@ieee.org (J. Vanhatalo), hvo@zurich.ibm.com (H. Völzer), koe@zurich.ibm.com (J. Koehler).
[1] Tel.: +41 44 724 8111; fax: +41 44 724 8953.

0169-023X/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2009.02.015

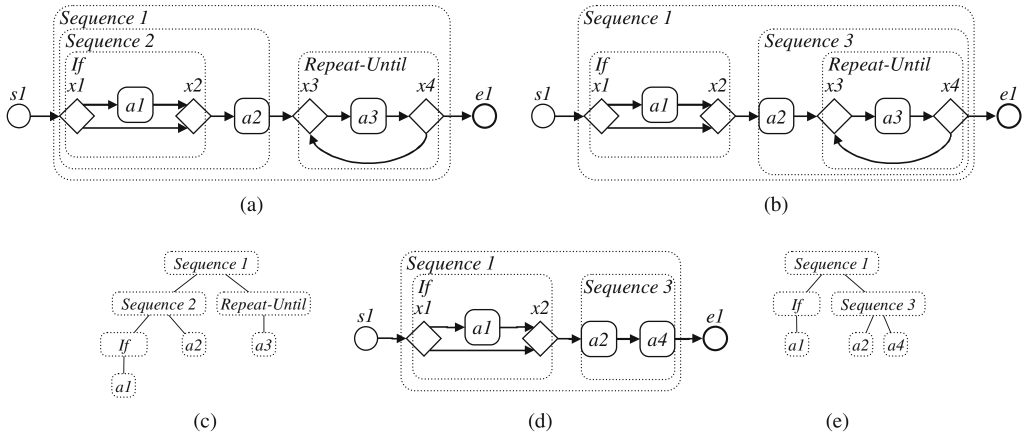

Fig. 1. (a), (b) Two parses of the same workflow graph. (c) Parse tree corresponding to (a). (d) Workflow graph obtained by a local change and its parse. (e) Parse tree corresponding to (d).

For a roundtripping between a BPMN diagram and a BPEL process, it is desirable that the decomposition be unique, i.e., the same BPMN diagram always translates to the same BPEL process. Consider, for example, the workflow graph in Fig. 1a. The translation algorithm proposed by Ouyang et al. [5] is nondeterministic. It may produce one of the two parses shown in Fig. 1a and b, depending on whether the if-block or the repeat-until-block is found first by the parsing algorithm.

One idea to resolve some of this nondeterminism is to define priorities on the syntactic categories to be found [5–7]. For example, if in each step the parsing algorithm tries to find sequences first, then if-blocks and then repeat-until-blocks, we can only obtain the parse in Fig. 1a in our example. However, this can introduce another problem. If we change a single block, say, the repeat-until block by replacing it, e.g. by a single task, we obtain the workflow graph shown in Fig. 1d. Fig. 1d also shows the parse we obtain with the particular priorities mentioned above. The corresponding parse tree is shown in Fig. 1e. It cannot be derived from the tree in Fig. 1c by just a local change, viz., by replacing the Repeat-Until subtree. For a roundtripping between a BPMN diagram and a BPEL process, it would be much more desirable that a local change in the BPMN diagram also result in only a local change in the BPEL process. Replacing a block in the BPMN diagram would therefore only require replacing the corresponding block in the BPEL process. We then call a such decomposition *modular*. The existing approach to the BPMN to BPEL translation problem [5] is not modular. Furthermore, it does not provide, because of the above problems, a specification of the translation that is independent of the actual translation algorithm.

A unique and modular decomposition is provided by the *program structure tree* defined by Johnson et al. [8,9] for sequential programs. It was applied to workflow graphs by Vanhatalo et al. [2] to find control-flow errors. The corresponding decomposition for our first example is shown in Fig. 2. It uses the same notion of a block as Ouyang et al. [5] do, that is, a block is a connected subgraph with a single entry and a single exit edge. But in contrast to the approach of Ouyang et al. [5], non-maximal sequences are disregarded in the program structure tree. For example, Sequence 2 in Fig. 1a [likewise Sequence 3 in subfigure (b)] is non-maximal: it is in sequence with another block.

Another general requirement for parsing is to find as much structure as possible, i.e., to decompose into blocks that are as fine as possible. As we will see (cf. Section 6), this allows us to map more BPMN diagrams to BPEL in a structured way. It has also been argued [5] that the BPEL process is more readable if it contains more blocks. Furthermore, debugging is easier when an error is local to a small block rather than to a large one.

In this paper, we provide a new decomposition that is finer than the program structure tree as defined by Johnson et al. [8,9]. It is based on and extends similar work for sequential programs by Tarjan and Valdes [10]. The underlying notion of a block is a connected subgraph with unique entry and exit *nodes* (as opposed to *edges* in the previous approach). Accordingly, all blocks of the previous approach are found, but more may be found, resulting in a more refined parse tree. Therefore, we call our decomposition the *refined process structure tree* (*RPST*, for short). We prove that our RPST is unique and modular.

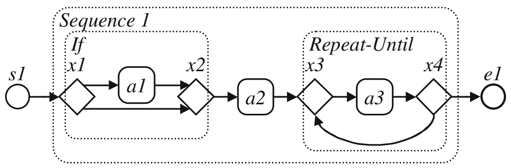

Fig. 2. Modular decomposition of the process from Fig. 1.