Information and Software Technology 75 (2016) 148–170

Contents lists available at ScienceDirect

Information and Software Technology

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

Specification of behavioral anti-patterns for the verification of block-structured Collaborative Business Processes

J. Roa[a,*], O. Chiotti[b], P. Villarreal[a]

[a] UTN-FRSF-CONICET, Lavaisse 610, Santa Fe, Argentina
[b] INGAR - CONICET, Avellaneda 3657, Santa Fe, Argentina

ARTICLE INFO

Article history:
Received 3 July 2015
Revised 11 December 2015
Accepted 3 January 2016
Available online 19 April 2016

Keywords:
Anti-patterns
Collaborative Business Processes
Verification
Formal methods
Control flow
Behavior
Cross-organizational collaborations

ABSTRACT

Context: The verification of the control flow of a Collaborative Business Process (CBP) is important when developing cross-organizational systems, since the control flow defines the behavior of the cross-organizational collaboration. Behavioral anti-patterns have been proposed to improve the performance of formal verification methods. However, a systematic approach for the discovery and specification of behavioral anti-patterns of CBPs has not been proposed so far.

Objective: The aim of this work is an approach to systematically discover and specify the behavioral anti-patterns of block-structured CBP models.

Method: The approach proposes using the metamodel of a CBP language to discover all possible combinations of constructs leading to a problem in the behavior of block-structured CBPs. Each combination is called minimal CBP. The set of all minimal CBPs with behavioral problems defines the unsoundness profile of a CBP language, from which is possible specifying the behavioral anti-patterns of such language.

Results: The approach for specification of behavioral anti-patterns was applied to the UP-ColBPIP language. Twelve behavioral anti-patterns were defined, including support to complex control flow such as advanced synchronization, cancellation and exception management, and multiple instances. Anti-patterns were evaluated on a repository of block-structured CBP models and compared with a formal verification method. Results show that the verification based on anti-patterns is as accurate as the formal method, but it clearly improves the performance of the latter.

Conclusion: By using the proposed approach, it is possible to systematically specify behavioral anti-patterns for block-structured CBP languages. During the discovery of anti-patterns different formalisms can be used. With this approach, the specification of anti-patterns provides the exact combination of elements that can cause a problem, making error correction and result interpretation easier. Although the proposed approach was defined for the context of CBPs, it could be applied to the context of intra-organizational processes.

1. Introduction

Business processes are strategic assets of organizations used to deliver value-added products and services to clients. In order to adapt to the market dynamics, organizations have been encouraged to integrate their business processes by establishing cross-organizational collaborations with their partners, customers, and providers [1]. A cross-organizational collaboration involves undertaking collaborative and cooperative actions for a period of time to achieve common goals, coordinate activities, share information, and define and execute Collaborative Business Processes [2–4].

A Collaborative Business Process (CBP) defines, from a global perspective, the choreography of interactions that take place between organizations involved in a cross-organizational collaboration, as well as the way the cross-organizational process-aware information systems (PAISs) will interact [5,6]. Examples of languages for modeling CBPs are BPMN [7], WS-CDL [8], UMM [9], or UP-ColBPIP [5,6].

The control flow of CBPs defines the behavior of cross-organizational collaborations, therefore its verification is an important aspect for cross-organizational PAISs development. Two important requirements of methods for business process verification

* Corresponding author.
E-mail addresses: jorgemarceloroa@gmail.com, jroa@frsf.utn.edu.ar (J. Roa), chiotti@santafe-conicet.gov.ar (O. Chiotti), pvillarr@frsf.utn.edu.ar (P. Villarreal).

http://dx.doi.org/10.1016/j.infsof.2016.01.001
0950-5849/

are: completeness and performance.*Completeness* means a method should support verification of every business process model, which implies supporting all language constructs. *Performance*, refers to the time a method requires to carry out the verification [10,11]. These two requirements are also valid for CBP verification methods.

Supporting a complete set of constructs may affect the verification performance, and viceversa. Therefore, there is a trade-off between these requirements. Current methods that support the verification of complex constructs [12–15] analyze the state space of models, which could lead to the state space explosion problem [16], affecting verification performance; and methods that can verify processes in linear time [11] do not support complex control flow constructs.

This trade-off can be solved by using anti-patterns [17–19]. A *behavioral anti-pattern* of CBPs is a predefined and well-known situation of a deficient specification of the control flow of CBPs. Once anti-patterns have been specified, CBP verification relies on a pattern matching technique, instead of exploring the state space of models.

Existing behavioral anti-patterns (either for the context of intra-organizational processes [18,20,21] or for CBPs [21]) were specified from well-known problems described in the literature, or from repositories of real case processes. Two main disadvantages of process repositories are: (1) insufficient models; and (2) bias by common problems of business process designers. Therefore, some anti-patterns could not be identified from the repositories, and hence, a verification method based on identified anti-patterns could not satisfy the completeness requirement.

To cope with this issue, in this work we propose an approach to systematically discover an specify behavioral anti-patterns of CBP languages. It consists in discovering all possible combinations of block-structured constructs composed of the least possible number of elements where such combination leads to a behavioral error, so that if one of these combinations of constructs is part of a CBP, then the CBP will have a behavioral error. Once discovered and specified, anti-patterns can be used to detect behavioral errors such as deadlocks, livelocks, and lacks of synchronizations in the control flow of block structured CBP models. In order to validate and evaluate the approach, we specified behavioral anti-patterns of UP-ColBPIP [6], which is a block-structured language for CBP modeling. Although the proposed approach was defined for the context of CBPs, it could also be applied to the context of intra-organizational processes.

This work is structured as follows. Section 2 reviews general concepts and existing work about modeling, structure, and verification of CBPs. Section 3 introduces the concepts of block-structured CBPs and minimal CBPs. Section 4 presents the approach to specify behavioral anti-patterns. Section 5 shows the specification of anti-patterns for UP-ColBPIP. Section 6 presents the evaluation of the proposed approach. Section 7 discusses related work. Finally, Section 8 presents conclusions and future work.

2. Background

2.1. Modeling of CBPs

Currently, there are different languages for modeling and implementing CBPs such as BPMN [7], BPEL [22], UP-ColBPIP [6], WS-CDL [8], among others. These languages have a structural and a behavioral semantics. In the *structural semantics*, each element of the concrete syntax of a language usually represents a construct. Constructs are defined at the language meta-model level, whereas their instances are defined at the model level. A construct may have infinite instances, and such instances can be used to generate a possibly infinite set of models from a given metamodel. The constructs of a given language are defined by a non-empty set of classes (or types) of the language's metamodel, and the relations between constructs' instances in a model are determined and restricted by the metamodel. Since a CBP model is always composed of instances of constructs, in this work we use the term *CBP element* (or just *element*) to refer to an instance of a construct that is part of a given CBP model.

The *behavioral semantics* of a CBP language is defined at a metamodel level as part of the behavioral semantics of each control flow construct and determines how a process model or specification will be executed, i.e. the execution ordering of their elements, which is also referred to as the control flow perspective of process models. The use of control flow constructs is essential to define the behavior of CBPs, since they determine the behavioral semantics of the elements of a CBP model.

The focus of this work is the behavioral semantics. To model behavior, CBP languages usually provide simple control flow constructs such as sequence, split and join to represent concurrency, and decision and merge to represent mutual exclusion; and complex control flow constructs such as advanced synchronizations, cancellation and exception handling, multiple instances, etc.

2.2. Structure of business process models

The combination of business process elements determines the structure of business process models. The metamodel of the language plays an important role in this subject. Two types of business process languages can be distinguished: graph-structured and block-structured [23]. A *graph-structured* language (such as BPMN [7]) enables the definition of business process models by combining its elements in any way, as long as the combination is supported by the language's metamodel. These models are essentially graphs. On the contrary, a *block-structured* language (such as UP-ColBPIP [6]) has constraints defined in its metamodel to provide block-structured constructs. A block starts with a construct that represents the divergence of parallel or alternative paths and ends with a construct that represents the convergence of such paths. This implies that models are block-structured and can be represented as trees [20,24], where each element of a model (except the root) must be nested within another one. Details about block-structured CBP models are presented in Section 3

Despite its unstructured nature, usually graph-structured languages can generate instances of both graph-structured and block-structured models, whereas block-structured languages only generate instances of block-structured models. For example, Fig. 1 shows a block-structured CBP model defined with BPMN (Fig. 1a) and UP-ColBPIP (Fig. 1b). After the start event, the BPMN choreography model contains two *Parallel gateways* that define a block. One (annotated with the text "Parallel split") represents the divergence of the paths, whereas the other one (annotated with "Parallel join") represents their synchronization. In the UP-ColBPIP model, the construct *And* is a block that represents both the split and the join. The first *parallel gateway* of the BPMN model is composed of two diverging paths. One contains a *Loop Activity* annotated with "While", whereas the other one contains two *Exclusive gateways* (annotated with "Decision" and "Merge") that define a block. This block is followed by a *Loop Activity*. In the UP-ColBPIP model this is represented with the constructs *Loop-While, Xor*, and *Loop-Until* respectively. Within the exclusive gateways of the BPMN model there are two other *parallel gateways* defining a block with two interaction activities, one for each parallel path. In UP-ColBPIP this is represented with the construct *And* within the *Xor*. UP-ColBPIP is described in more detail in Section 5.1.