Information Systems 74 (2018) 40–52

Contents lists available at ScienceDirect

Information Systems

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

Discovering hidden dependencies in constraint-based declarative process models for improving understandability

Johannes De Smedt[a,b,*], Jochen De Weerdt[a], Estefanía Serral[a], Jan Vanthienen[a]

[a] KU Leuven Faculty of Economics and Business, Department of Decision Sciences and Information Management, United Kingdom
[b] University of Edinburgh Business School, Management Science and Business Economics Group, United Kingdom

A R T I C L E I N F O

Article history:
Received 4 October 2016
Revised 24 October 2017
Accepted 15 January 2018
Available online 31 January 2018

Keywords:
Declarative process modeling
Declare
Hidden dependencies
Constraint-based process models
Model comprehension
Empirical research

A B S T R A C T

Flexible systems and services require a solid approach for modeling and enacting dynamic behavior. Declarative process models gained plenty of traction lately as they have proven to provide a good fit for the problem at hand, i.e. visualizing and executing flexible business processes. These models are based on constraints that impose behavioral restrictions on process behavior. Essentially, a declarative model is a set of constraints defined over the set of activities in a process. While allowing for very flexible process specifications, a major downside is that the combination of constraints can lead to behavioral restrictions not explicitly visible when reading a model. These restrictions, so-called hidden dependencies, make the models much more difficult to understand. This paper presents a technique for discovering hidden dependencies and making them explicit by means of dependency structures. Experiments with novice process modelers demonstrate that the proposed technique lowers the cognitive effort necessary to comprehend a constraint-based process model.

1. Introduction

Declarative process models (DPMs) have been proposed to counter the limitations of procedural modeling languages to represent and execute processes flexibly [1,2], a desirable feature that is of importance in application areas such as healthcare [3] or services [4]. Instead of modeling predetermined paths of activities, declarative process models typically use constraints or rules to express what can, cannot, and must happen. Every execution sequence that is not strictly forbidden by the constraints can be enacted by the model, which makes it very flexible. Many executions are possible due to the interaction of the constraints over the activities. However, each type of constraint can have distinctive effects on the enabledness of an activity, creating dependencies that are not explicit or visible in the graphical model or even in the execution semantics. The dependencies between constraints and activities that are not explicit or visible in the model [5–7] are so-called hidden dependencies and make declarative models difficult to comprehend [6,8].

This paper proposes a technique capable of revealing hidden dependencies in constraint-based declarative process models by propagating the constraints' properties through the activities of a model and builds upon the prior work in [9]. It extends this work by explicitly addressing how constraints propagate their restrictions over activities, as illustrated in Section 3, and forms the backdrop for building dependency structures for the whole model. They then are used to visualize the dependencies, as well as create textual annotations. In this way, the paper addresses two suggestions for improvement that were found in the user study performed in [6], i.e. 'Simplify combination on constraints' by introducing an extra layer of annotation that explains their relations, and 'Make hidden dependencies explicit' by capturing them in a visual model and textual annotations. The technique has been implemented for the Declare language [10], one of the most-widely used declarative process languages, and is implemented in the newly-developed Declare Execution Environment,[1] a tool that supplements an existing Declare model with visual and textual annotations and shows which behavior is allowed or disallowed by the model and why.

The technique has been tested in an empirical evaluation in which 146 novice modelers participated. The results are reported in greater detail with a more in-depth statistical analysis using more factors, and a bigger sample than in [9]. The evaluation shows that hidden dependencies pose a significant burden for the modelers to understand the full behavior of a model, and demon-

* Corresponding author at: University of Edinburgh Business School, Management Science and Business Economics Group, United Kingdom.

E-mail addresses: Johannes.DeSmedt@kuleuven.be, Johannes.DeSmedt@ed.ac.uk (J. De Smedt).

[1] http://www.processmining.be/declareexecutionenvironment.

https://doi.org/10.1016/j.is.2018.01.001
0306-4379/© 2018 Elsevier Ltd. All rights reserved.

strates that our technique actually has an impact on the cognitive complexity and improves the understandability of declarative process models by uncovering these dependencies. Using the proposed technique, modelers were better capable of getting a holistic view on the model they had to interpret, and were better capable of answering questions regarding the behavior of the model correctly.

The structure of the paper is as follows. First, in Section 2, the concept and background of DPMs are summarized and relevant characteristics are explained. In Section 3, the concept of a constraint-based DPM and the constraint propagations in Declare are formalized. In Section 4, the construction of constraint dependency structures is elaborated on. In Section 5, the tool is introduced which we developed to implement the ideas. Section 6 reports on the results of an experiment performed on users of the tool, and Section 7 concludes the paper with an overview of the results and future work.

2. State-of-the-art

In this section, the state-of-the-art of declarative process models and the research on their understandability is discussed.

2.1. Languages and approaches

There exist numerous types of declarative process modeling languages in literature. The term surfaced with the proposal of EMBrA^{2}CE [2], DecSerFlow, and Condec [1,11]. The former proposes an extension of Semantics and Business Rules Vocabulary [12] to model business processes, while the latter proposed a framework grounded in Linear Temporal Logic (LTL)-based constraints and later was transformed into the Declare language [10]. A similar approach was followed for DCR Graphs [13], a framework also based on a set of constraints, which is smaller but when used in a composite way offers at least the same expressiveness as LTL [14]. These contributions focus mainly on the control flow, though data-aware extensions exist, e.g., [15]. Other research focuses on a declarative specification of artifacts, including Guard Stage Milestone [16]. In recent years, however, Declare has seen the biggest surge in terms of research interest in many application areas such as process mining [17], application development [18], and process verification [19]. Therefore, it will be used to illustrate the example in this work. Other languages also support Declare for model checking as well, i.e. in SCIFF [4]. As a framework, it aims to support different semantics besides LTL, such as regular expressions [20,21], and R/I-nets [22]. Furthermore, the template base is extensible, allowing Declare to support any constraint that can be defined with the same semantics. An overview of the Declare semantics in LTL and regular expressions are given in Table 1.

Declare uses a graphical notation that depicts activities as boxes and constraints as edges of various types. All such edges are annotated with their respective name, except for unary activities for they are either similar to cardinalities, or contain the constraint name. A major downside of the framework however, is that the graphical notation and execution model are separated. This downside was overcome in DCR Graphs by representing the states explicitly in the form of markings.

2.2. Understandability of declarative process models and hidden dependencies

The departure of defining explicit control flow as in Business Process Model and Notation [23] or Petri nets [24], has urged a number of researchers to study the understandability and usability of DPMs in general. In [8,25] , the comparison with the procedural process modeling paradigm was made in order to get a grasp on the benefits and downsides of using either approach, especially from the viewpoint of the reader and modeler. A case study with practitioners showed that, rather than using a full declarative approach, a hybrid approach suits interests best [26]. Since the control flow in declarative models is often underspecified, there is a vast number of execution scenarios which might impede the user's understanding of the actual behavior of a DPM. An investigation into the size and understandability of models was performed using the Alaska Simulator in [27], which presents a process as a journey and was used in a user study for solving planning problems. Later, a test suite for DPMs was introduced in [7,28], looking into how users and modelers make use of the interplay and changes of constraints in a model. A comprehensive study of the understandability factors, the notation, usage of hierarchy [29], and interpretation strategies of users, was performed in [30]. In the latter works, it became apparent that *hidden dependencies* and the interplay of constraints clearly impede the understandability and raise the cognitive load of the reader and modeler. A hidden dependency in software was defined in [5,31] as 'a relationship between two components such tat one of them is dependent on the other, but that the dependency is not fully visible'. They raise cognitive complexity [32] due to intertwining parts of software, and often are urged to be added visually to raise understandability [33].

2.3. Alleviating understandability issues: discovering hidden dependencies

As will be illustrated below, hidden dependencies arise in DPMs when activities propagate their cardinalities, or are prevented from executing at a certain point in time, i.e., by *negative* constraints in Declare or *exclude* relations in DCR Graphs. In this paper, mitigation is sought for by making all dependencies between constraints that are not explicit in the model itself (i.e. hidden) visible in the form of textual annotations and dependency graphs. This approach is general to the extent that new constraints in any type of semantics can be added, as long as the basic principles of constraint-based models listed are followed.

3. Preliminaries

In this section, the concept of a DPM and its constraints is formalized.

3.1. Constraint-based declarative process models

A declarative process model $DM = (A, \Pi)$ can be defined as follows.

- A is a set of activities or atomic propositions from the alphabet Σ,
- $\Pi(A)$ is a set of constraints defined over the activities,
- $\S(\pi), \pi \in \Pi$ is a function assigning semantics to a constraint, and
- $\Phi = \bigwedge_{\pi \in \Pi} \S(\pi)$ is the model comprised of the conjunction of constraints, given that the language used to express the constraints is closed for common properties such as concatenation, intersection, Kleene star, and so on, as is the case for regular expressions and LTL.

For every activity $a \in A$ and timestamp $t \in \mathbb{N}$, we define

- $L : (a, t) \rightarrow \mathbb{N}$ the lower bound of the amount of occurrences of an activity at time t,
- $U : (a, t) \rightarrow \mathbb{N}$ the upper bound of the amount of occurrences of an activity at time t,
- E: $(a, t) \rightarrow \{0, 1\}$ a function keeping track of the enabledness of an activity at time t,
- O: $(a, t) \rightarrow \{0, 1\}$ a function indicating whether an activity fired at time t, and