Computers in Industry 87 (2017) 52–67

Contents lists available at ScienceDirect

Computers in Industry

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

Recalling the rationale of change from process model revision comparison – A change-pattern based approach

Selim Erol

Institute of Management Science, Department of Industrial and Systems Engineering, Vienna University of Technology, Austria

ARTICLE INFO

Article history:
Received 30 January 2017
Accepted 10 February 2017
Available online xxx

Keywords:
Business process modeling
Change patterns
Pattern detection
Process-aware information systems

ABSTRACT

Industrial enterprises embody a rather large and heterogeneous business process landscape including hundreds to thousands of both manufacturing and supporting processes. To keep records of their business process architecture enterprises use informal and as well formal process descriptions. Formal process descriptions are often referred to as process models and are gaining increasing importance as a basis for process-aware enterprise information systems and automation purposes. Therefore process models are continuously adapted to changing business requirements. Keeping track of model changes is an important requirement to be able to understand past decisions and their impact on the process landscape. Hence, keeping track of changes is not easy if changes are not associated with the original rationale and the order of atomic changes is not preserved anymore. In this paper we present an approach that builds upon the concept of change patterns. For this purpose we systematically examined revision histories from a large process model collection and described them through a pattern language. In addition, we propose an algorithm to detect such change patterns. Our approach has been implemented in a modeling environment and has been evaluated with regard to effectiveness and performance.

1. Introduction

Modern industrial enterprises reveal a rather large business process landscape including hundreds to thousands value-adding manufacturing and as well supporting processes. Therefore enterprises try to keep records of their business process architecture in some way. On the one hand informal process descriptions are used for the goal of documentation, analysis, knowledge transfer and governance. On the other hand business processes are described through (semi-)formal languages (e.g. EPC [1], BPMN [2]) which potentially allow for their automation and orchestration of related software services and applications [3,4]. The latter kind of artifacts, commonly referred to as process models, are created, stored and maintained by means of software-based modeling environments (e.g. ARIS Business Architect[1]). These modeling environments typically consist at least of a model editor and a model repository. The repository is used to store larger collections of process models, enable version management and efficient model retrieval [5]. Such process model repositories often contain hundreds to thousands of process model artifacts and – in case some kind of versioning exists – also a multiple of model versions.

In practice, process models are developed in an iterative manner. From vague ideas of the process in focus to informal process descriptions and finally deployable workflow specifications and so forth [6]. Process models are also continuously adapted to changing business requirements. Keeping track of model changes is typically accomplished by a change log or a revision history. In the first case changes are recorded by storing at least the order of operations, the type and the target object. Thus, a sequence of change operations is obtained that exactly shows the actions a modeler has undertaken to change a model. In the case of the revision history approach snapshots at discrete points in time are taken and stored along with a time-stamp. Model changes are then reproducible by comparing two subsequent revisions revealing only the substantial changes but hiding all intermediate changes that led to the final change. Both approaches have their advantages and disadvantages which have been broadly discussed in literature, e.g. in [7]. In practice, the revision history approach has gained ground in both model and software development tools. Versioning systems like SVN[2] are exemplary implementations of

E-mail address: selim.erol@tuwien.ac.at (S. Erol).

[1] http://www.aris.com.

[2] https://subversion.apache.org.

http://dx.doi.org/10.1016/j.compind.2017.02.003
0166-3615/

the revision history approach. One of the drawbacks of the revision history approach is that the rationale of changes to a process model is not easily recoverable if not documented explicitly or implicitly through a change log. By rationale the idea, intention and reason for a change is meant which leads to an operational change of a process model artifact. The above mentioned problem is due to the fact that atomic changes resulting from a change intention do not have an envelope that indicates the rationale. Rather a change rationale needs to be "guessed" a posteriori from a set of unrelated atomic change operations which have been derived from a simple revision comparison.

To tackle the above mentioned challenges we have chosen an approach that builds upon the concept of change patterns formulated in [8] and extends this concept in various ways. First, by conducting an explorative study of revision histories from a two long-term case-studies or process modeling and the subsequent classification of compound changes through change patterns. Second, by introducing a pattern language allowing the description of typical compound changes (change patterns) in an easy, practicable but still comprehensible way. Second, by formulating algorithms for recovering compound changes from revision comparisons. Finally, through a prototypical implementation in a respective modeling environment and its subsequent evaluation.

In the subsequent sections of this paper the approach for describing change patterns through a pattern language and related detection algorithms are described in detail. In Section 2 we discuss basic concepts of revision history, revision comparison and compound changes. In Section 3 we present the pattern based approach to specifying compound changes. Section 4 is dedicated to the formulation of respective detection algorithms. In the last section a prototypical implementation is presented along with it's evaluation. Results are summarized and discussed in the last section.

2. Basic concepts

2.1. Process model

For the following considerations a simple meta-model for process models has been used. Accordingly, a process model M is a finite set of model elements $\dot{m} \in M$ where each model element is classified into three distinct types of model elements *a*..activity, *g*.. gateway, *e*..sequence flow edge. Instances of such classes form subsets of M: the set of all activities $A = \{a_0. . a_{n-1}\}$, the set of all gateways $G = \{g_0. . a_{m-1}\}$, the set of all edges $E = \{e_0. . e_{k-1}\}$ where $n, m, k \in \mathbb{N}$. Thus, the set of model elements M can be described as a three tuple (A, G, E). Where A is a finite set of activities, G is a finite set of logical gateways and E is a finite set of directed sequence flow edges. E represents different sequences of A and G. To be precise it can specified as a subset $E \subseteq (A \times G) \cup (G \times A) \cup (A \times A) \cup (G \times G)$. Different types of gateways $\{\times, +, \circ\}$ may exist where type $T(g_i) = \times$ refers to an exclusive gateway, type $T(g_i) = +$ refers to a parallel gateway and type $T(g_i) = \circ$ refers to an inclusive gateway. Gateways are also classified according to their role in the control flow. We distinguish the set of split gateways G_s which includes all gateways g_i that have at least two outgoing sequence flow edges and have exactly one ingoing sequence flow edge, formally $G_s = \{\forall g_i \in G \mid OUT(g_i) \geq 2 \wedge IN(g) = 1,\ 0 \leq i \leq k-1\}$ where OUT is a function that counts all outgoing and IN is a function that counts all ingoing edges for a given gateway node and a set of join gateways G_j that is defined as $G_j = \{\forall g_i \in G \mid IN(g_i) \geq 2 \wedge OUT(g_i) = 1,\ 0 \leq i \leq k-1\}$. Fig. 1 shows a UML[3] class diagram of the types of model elements incorporated for subsequent considerations. Note that we use a rather general and relaxed meta-model and formalization. This is due to the requirement that we want to cover as well process models in progress that are not complete in the sense of executability. We also did not include events in the meta-model as we wanted to provide a general meta-model that abstracts from concrete modeling languages and keeps the semantics simple to have more explanatory power. The formal description of the meta-model provided above builds upon the work of [9].

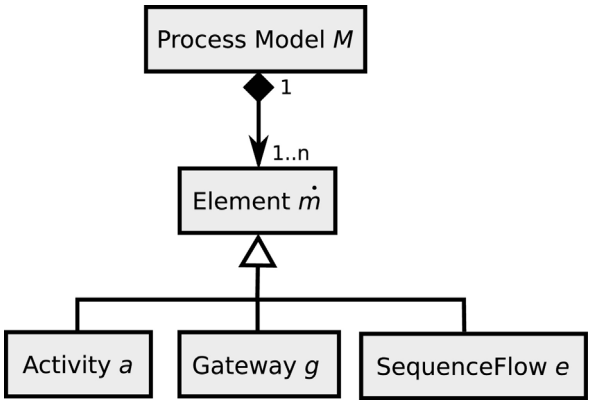

Fig. 1. Meta-model of process models as used in subsequent considerations.

2.2. Process model change

Process models are created through a model editing software and are maintained in process model repositories [10,11,5]. Changes to process models are performed through their graphical representations. Such graphical representations usually consist of shapes and edges of different types to represent the semantics of process model elements. E.g. a rectangular shape with rounded corners to represent a task. When an agent changes a process model then it performs a set of subsequent atomic change operations until a desired new state of the model is reached. These atomic change operations are the result of some change rationale – an agents intent to change a process for some reason. The rationale of change therefore can be associated a set of atomic change operations that follow some order. The set of compound atomic change operations – a compound change – can be defined as a tuple $cc = (O, Rel)$ consisting of an unordered set of atomic change operations $O = \{o_0, o_1, \ldots, o_k\}$ and a set of relations $Rel = \{rel_0, rel_1, \ldots, rel_{m-1}\}$ where $k, m \in \mathbb{N}$ and $m \leq |O| - 1$ that determine the order atomic change operations have been executed. Each atomic change operation $o = o_i(\dot{m}, type, args)$ is defined through it's target model element $\dot{m}$, the *type* of operation ("add", "delete", "modify") and optional arguments *args*.

2.3. Process model revisions and revision comparison

A revision history of a process model represents different states of a process model over time. These different states of a process model result from interactions of some kind of agent with the process model artifact leading to respective changes. The scenario depicted in Fig. 2 shows two model interactions that take place in sequence. The first modeler u_A opens (checks out) a model revision r_i from a repository at point in time $t_{A,s}$, changes the model, and submits (checks in) the changes to the repository at point in time $t_{A,e}$ which leads to model revision r_{i+1}. Subsequently, a second modeler u_B accesses the model revision r_{i+1}, opens it at a point in time $t_{B,s}$, changes it and submits as well his new model revision at point in time $t_{B,e}$ which leads to a revision r_{i+2}. This scenario

[3] http://www.uml.org.