Survey paper

Refactoring large process model repositories

Barbara Weber [a,*], Manfred Reichert [b], Jan Mendling [c], Hajo A. Reijers [d]

[a] *Department of Computer Science, University of Innsbruck, Technikerstraße 21a, 6020 Innsbruck, Austria*
[b] *Institute of Databases and Information Systems, University of Ulm, Germany*
[c] *Humboldt-Universität zu Berlin, Germany*
[d] *School of Industrial Engineering, Eindhoven University of Technology, The Netherlands*

ARTICLE INFO

Abstract: With the increasing adoption of process-aware information systems, large process model repositories have emerged. Typically, the models in such repositories are re-aligned to real-world events and demands through adaptation on a day-to-day basis. This bears the risk of introducing model redundancies and of unnecessarily increasing model complexity. If no continuous investment is made in keeping process models simple, changes will become more difficult and error-prone over time. Although refactoring techniques are widely used in software engineering to address similar problems, so far, no comparable state-of-the-art has evolved in the business process management domain. Process designers either have to refactor process models by hand or are simply unable to apply respective techniques at all. This paper proposes a catalogue of process model "smells" for identifying refactoring opportunities. In addition, it introduces a set of behavior-preserving techniques for refactoring large process repositories. The proposed refactorings enable process designers to effectively deal with model complexity by making process models better understandable and easier to maintain. The refactorings have been evaluated using large process repositories from the healthcare and automotive domain. To demonstrate the feasibility of the refactoring techniques, a proof-of-concept prototype has been implemented.

© 2010 Elsevier B.V. All rights reserved.

Contents

---

* Corresponding author.
   E-mail address: Barbara.Weber@uibk.ac.at (B. Weber).

1. Introduction

Process-aware information systems (PAISs) have become an integral part of enterprise computing and are used to support business processes at an operational level [99]. In contrast to conventional information systems, PAISs strictly separate process logic from application code, relying on explicit *process models* that provide the schemes for process execution. This enables a separation of concerns, which is a well established principle in Computer Science to increase maintainability and to reduce costs of change [13].

1.1. Problem statement

Process repositories are the central store of process models in PAISs. In large companies, such repositories can easily contain several thousands of process models [72]. Such sheer numbers give rise to several quality issues. Over time new process models emerge, existing ones need to be adapted to changing requirements, and new process model variants are created to align processes to a particular context (e.g., specific regulations in one of the countries where the company operates). While support for model changes is quite well understood from a research perspective both in terms of process model configuration [74,71] and adaptation of running process instances [58,60], a notable research gap exists concerning quality assurance in process repositories.

This gap is underlined by two facts: First, regarding model construction companies try to delegate process modeling tasks to operational staff that has little or no modeling competence [72]. Thus, it is not surprising that process model repositories tend to contain a rate of unsound models that ranges from 3.3% up to 37.5% [40]. This rate is a severe roadblock to process model usage. Second, it is well known from software engineering research that computer programs degenerate over time when code is modified or added by different developers [54]. Since numerous users may work on a single process repository, we can expect that process repository evolution faces similar challenges as software program evolution does; i.e., maintenance will become increasingly difficult over time if no techniques for quality improvement are provided.

While methods and tool support are still limited in process modeling, there has been considerable progress in software engineering (SE) on related problems. So-called *refactoring techniques* have been widely used to ensure that code bases remain maintainable [52,17]. Refactoring enables programmers to restructure a software system without altering its behavior. Thus, it is typically used to improve code quality by removing duplication, improving readability, simplifying software design, or adding flexibility [2]. Examples of SE refactoring techniques include the renaming of a class to foster understandability and the extraction of a new method from an existing code block to reduce redundant code fragments and to increase readability. In the SE domain, *code smells* are widely used for identifying refactoring opportunities [50] (e.g., duplicate code or very long methods).

It has been noted by various authors that process models and computer programs are similar in various respects [23,90]. In [92], the following parallels are singled out:

- Both types of artifacts provide a *procedural view* on the processing of information. Within each described step, either within a process model or a computer program, one or more outputs are produced on the basis of one or more inputs.
- A process model has a *compositional structure* that is similar to that of a computer program. A computer program can be split up into modules or classes. Every module consists of a number of statements, and every statement references variables and constants. Likewise, a process model contains activities, each of these being composed of elementary operations, which in turn use one or more pieces of information to produce new information.
- Both a process model and a computer program can be used as *script for enactment*. When instantiating either of these, an execution flow of their elements is invoked that unfolds in accordance with this static representation. This flow may involve consecutive executions, concurrency, conditional routings, etc.

Considering these similarities, it is not surprising that some authors even refer to process modeling as "programming in the large" [90]. Our line of reasoning now is that the idea of refactoring, well-known in the area of software engineering, may well be an attractive direction to investigate in the context of process model usage in PAISs.

1.2. Contribution

This article adapts the concept of refactoring from SE to process modeling. Our contribution is twofold. Firstly, we provide an extensive discussion of *process model smells* facilitating the identification of refactoring opportunities. Secondly, we introduce *refactoring techniques* that provide remedies for these smells. A refactoring technique improves upon the *internal quality* of a model such that it becomes easier to read and maintain, but it does not affect the model's semantics or external behavior. The techniques

are proposed as a means to assist process designers, but the final decision whether or not to apply a refactoring in a specific situation is always at their discretion. In that sense, the proposed refactoring techniques support a modeler's task without making it superfluous.

The presented smells and refactorings are not complete in a mathematical sense: It can be easily imagined that refactorings might be added in future or that existing refactorings will be refined. The presented list, however, is duly empirically validated on utility considerations. Using a range of existing process repositories from the healthcare and automotive domain, we are able to show that all identified refactorings are frequently needed in practice. A similar argument of relevance is also used in SE as constructive criterion for design patterns [18]. Beyond that, we provide a second constructive validation in terms of a prototypical implementation, which demonstrates how the different refactorings can be offered to process designers in an easy-to-use fashion.

While some isolated refactorings are discussed in [15], our contribution is the first comprehensive account of the refactoring concept for process models. At the same time, process models typically comprise different perspectives including control-flow, data flow, and resource allocation. Our contribution is restricted in the sense that it purely focuses on the control-flow perspective. This is, however, not a fundamental limit: It can be imagined how the current set of refactorings can be extended to cover other dimensions as well.

This paper further complements previous work on *process redesign* and *process adaptation*. Both refactoring and *process redesign* may require model transformations. However, the scope of process redesign is broader and goes beyond structural adaptations. It is primarily business-driven and aims to improve one or more performance dimensions of a process (e.g., time, quality, costs) [61]. Therefore, redesign often affects the external quality of a PAIS and its results are visible to the customer. In contrast, refactoring techniques primarily impact the internal quality of a PAIS, ensure conceptual integrity, and foster maintainability. Similar to refactorings, *process adaptations* [58,60,97] refer to structural changes of a process model (e.g., using change patterns) [95]. In contrast to refactorings, they usually affect process model behavior.

The paper is structured as follows. Section 2 presents a generic meta model we assume for a process repository. We use an illustrative example to introduce process modeling as well as refactoring concepts. We then define the scope of the process model refactorings we consider. Section 3 presents a set of empirically supported process model smells for detecting refactoring opportunities. Section 4 describes 11 refactoring techniques which enable process designers to improve the quality of process models and provide remedies for the process model smells as discussed in Section 3. Section 5 demonstrates the applicability of the refactorings based on a prototypical implementation and provides a realistic use scenario. Section 6 discusses related work, before Section 7 concludes the paper.

2. Preliminaries

We first introduce general concepts providing the foundation of this paper. Section 2.1 presents the meta model we assume for a process repository. Section 2.2 defines refactoring and aligns it to process modeling concepts.

2.1. Process repository meta model

The most essential entities in a *process repository* are *process models*. Several *process modeling languages* have been defined including Event-driven Process Chains (EPCs), Business Process Modeling Notation (BPMN), and Workflow Nets. They have distinctive elements and sometimes display subtle differences in semantics. In the following, we aim to abstract from these differences, and focus on basic commonalities of these languages instead [84]. Accordingly, we define a *process model* as a set of activities and gateways that are connected by control-flow arcs. Gateways can be either split nodes (i.e., nodes with one incoming and multiple outgoing arcs) or join gateways (i.e., nodes with multiple incoming and one outgoing arc). There are three different types of splits and joins. The XOR-split defines a decision point where one outgoing branch becomes activated and the XOR-join the respective merge. The AND-split introduces concurrent processing of all outgoing branches while the AND-join synchronizes its incoming branches. The OR-split represents a non-exclusive choice in the sense that one, multiple, or all outgoing arcs can be activated. The OR-join guarantees proper synchronization of those branches that have become active.

There are relationships that span different process models as well. Most relevant is the *subprocess* relationship that refers from an activity of one process model (parent) to another process model as a whole (child). This signifies that the subprocess implements the activity, i.e., every time the activity gets activated, it is the subprocess that has to be executed. We denote such an activity as a *complex activity*. We require the parent-child process relationship to be acyclic such that we have different *process model trees* linking parent and child process models. Subprocesses constitute a powerful concept for describing the common parts of different process models.

Fig. 1a illustrates the content of a very simple process repository at a certain point in time. There are five process models S, S1, S2, S3, and S4. Model S includes an AND-split after activity A. Accordingly, B, C and D, and also E and F can be executed concurrently. The AND-join synchronizes the different paths. M is a complex activity pointing to subprocess S3 that executes activity sequence X, Y and Z. Model S1 uses this subprocess as well. S1 and S2 also contain the same process fragment, which is built upon an AND-split and AND-join. Finally, S2 contains complex activity K that refers to S4. As can be seen, parts of the different models are redundant in the sense that they cover exactly the same process logic.

Process models can either be created from scratch or through adaptation of a *reference process model*, i.e., by means of configuration. From such a reference model S_{ref}, several *process model variants* $V_1, \ldots, V_n$ can be derived based on a restricted set of high-level change patterns [58,95].[1] Thereby, for a given variant model V we denote the minimal number of high-level changes needed to transform the reference model S_{ref} into V as *change distance* $\sigma(S_{ref}, V)$ between S_{ref} and V. Furthermore, a minimal sequence of high-level changes needed to transform model S_{ref} into V is denoted as *bias* (Δ) between S_{ref} and V.[2] The total set of all variant models (i.e., variants for short) derived from a reference process is called a *process model family*.

Fig. 1b shows a reference process model S_{ref} and four process variants $V_1, \ldots, V_4$ derived from it; e.g., to configure S_{ref} into V_2 we need to insert Y and delete Activity G; i.e., we obtain distance $\sigma(S_{ref}, V_2) = 2$ and bias $\Delta(S_{ref}, V_2) = [$ Insert Y after C, Delete G].

Based on a given process model, at run-time new *process instances* can be created and executed according to this model. The latter is reflected by the execution *traces* of these instances, which log information about events relating to the start and completion of process activities [85].

---

[1] Examples of change patterns include the insertion, deletion and movement of activities within a process model.

[2] Generally, it is possible to have more than one minimal sequence of change operations to transform S_{ref} into V, i.e., given two process models their bias does not need to be unique (see [36] for a detailed discussion on this).

Fig. 1. Core concepts.

2.2. Process models and refactorings

The term "refactoring" was coined by Opdyke [52] and refers to "the process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure" [17]. As such, refactoring neither resolves errors nor adds functionality, but improves understandability and maintainability through *behavior-preserving* model transformations. Therefore, refactoring differs from model transformations applied when redesigning or adapting processes, since these transformations are typically not behavior preserving. Refactoring can be classified as both *endogenous* (i.e., transformations between models expressed in same language) and *horizontal* (i.e., source and target model reside at same level of abstraction) [49]. Refactorings constitute small changes with little value when applied in isolation, but these become valuable when combined with other refactorings [2]. Thus, model refactoring constitutes an iterative process which enables designers to improve the quality of a process repository. According to [50] we define refactoring by a procedure consisting of a number of distinct activities:

1. Identify refactoring opportunities
2. Determine which refactoring(s) shall be applied
3. Ensure that the applied refactoring(s) preserve model behavior
4. Apply the refactoring
5. Assess the effect of the refactoring on quality characteristics of the process model repository (e.g., understandability, maintainability)

In SE, the incentive to consider a particular refactoring is the detection of a *code smell* [17]. Code smells are indicators of bad *code quality* and in our application domain of bad *process model quality*. Let us revisit our process repository from Fig. 1a to illustrate this point. We have already stated that the repository contains several redundancies. Using "refactoring terms" we can now describe these as *process model smells*: models S1 and S2 both include the common process fragment[3] G1, which is a slight variation of process fragment G contained in the left model S (i.e., although process fragments G and G1 are not structurally equivalent, they expose the same behavior). Using refactoring techniques, we can extract these commonalities from the different models (cf. Fig. 2). This results in a new model S5, which represents the logic of process fragment G1 and G, respectively, and which becomes a subprocess of the refactored models S, S1, and S2. Due to the reduced redundancy, the resulting process models should now be easier to maintain, but still expose the same behavior.

Whether the occurrence of a smell really means that a model must be refactored is not a black or white decision. The value of a particular refactoring often involves the trade-off between different quality characteristics. For example, extracting (redundant) process fragments from one or several process models decreases the overall model size on the one hand, but potentially increases the number of process models (with low number of activities). This aspect is captured in our approach that aims to *assist* the process designer rather than to fully automate the refactoring process.

---

[3] In the context of this paper a (process) fragment denotes a subgraph of a process model with single-entry and single-exit node.
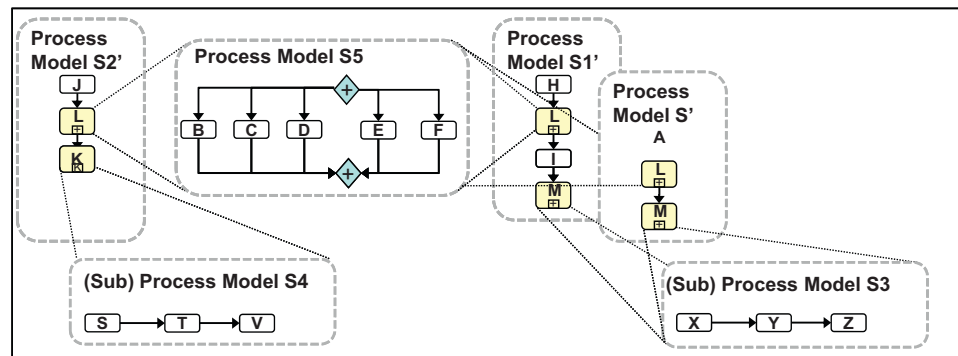
Fig. 2. Model repository after refactoring (cf. Fig. 1a).

In the following, we approach process model refactorings from two angles, and with reference to the five refactoring steps as summarized above. First, we explain how refactoring opportunities can be identified. In this context, we introduce a catalogue of process model smells that signal low process model quality. The guidelines are supported both by an empirical evaluation and a study of the relevant literature. Thereby, we assume that process designers determine – in the same spirit as software engineers do for code refactorings – whether or not a refactoring shall be applied taking various trade-offs into account. Second, we describe a set of behavior-preserving refactoring techniques which can be used to improve overall quality of the process model repository without changing actual process behavior. Again we assume that process designers assess the effect of the applied refactoring.

3. Identifying refactoring opportunities

In the SE domain *code smells* are the most popular method for identifying refactoring opportunities [50]. Picking up this metaphor, Section 3.2 introduces a list of *process model smells* serving as indicators for low process model quality. These smells were identified based on a large collection of process models from different domains (cf. Fig. 3). Such an empirical approach seems justified given the lack of an established theory that captures how modeling artifacts come into being. Additional support was gathered from a literature study that focuses on the understandability and maintainability of process models (cf. Section 3.1). Section 3.3 summarizes our results on process model smells.

3.1. Research methodology

We first describe the selection criteria for our process model smells, the data sources providing the empirical evidencefor them, and the procedure we applied for their identification.

3.1.1. Selection criteria

We consider process model smells for assisting designers in detecting opportunities for process model refactoring. Our focus is on smells which can be addressed by *behavior-preserving* refactoring techniques; i.e., our refactorings do not change the actual behavior of the process models to which they are applied. In addition, the smells should not be restricted to a specific process modeling language.

3.1.2. Data sources and data collection

The following collections of process models have been used as sources for the identification of process smells (cf. Fig. 3 for an overview). The data sources were carefully selected to cover processes from several domains (i.e., healthcare and automotive engineering) and to mitigate the risk that the identified process model smells are specific for a particular domain. Moreover, we ensured that the selected data sources comprise processes with different characteristics. The process models range from very small ones (with just a few activities) to very large models (with hundreds of activities). The sources include single process models, but also families of process models expressed in different process modeling languages. Finally, the selection of data sources also considered aspects like the presence of a sufficiently large number of models and full access to the respective process model repository.

Source 1. In a large *healthcare* project we analyzed five core processes of a women's clinic as documented in its organizational manual: birth and postnatal care, inpatient chemotherapy treatment, outpatient chemotherapy treatment, ovarian carcinoma surgery, and keyhole surgery [59]. In total, these five core processes consist of 70 process models, which are expressed either in terms of Event Process Chains or UML Activity Diagrams. Each process model contains 2–18 activities.

Source 2 comprises 46 process models (with up to 40 activities) representing medical guidelines and clinical pathways in internal medicine.

Data Source	Domain	Scenarios	Number of Models	Reference
Source 1	Healthcare	Birth and postnatal care Inpatient chemotherapy treatment Outpatient chemotherapy treatment Ovarian carcinoma surgery Keyhole surgery	70 process models	[59]
Source 2	Healthcare	Clinical guidelines and pathways in internal medicine	46 process models	
Source 3	Healthcare	Clinical guidelines for urinary stone diagnosis	1 process model with 98 instances	
Source 4	Healthcare	Handling of medical procedures (i.e., requesting, scheduling, performing and validating medical examinations)	84 process model variants	
Source 5	Automotive	Vehicle development	1 process model	[6]
Source 6	Automotive	Electronic change management	60 process models	[19]
Source 7	Automotive	Vehicle repair	900 variants	[24]

Fig. 3. Data sources for identifying process smells.

Source 3 consists of a clinical guideline for urinary stone diagnosis as implemented in a PAIS (1 process model with 98 process instances).

Source 4 comprises process models from a clinical centre, i.e., 84 process model variants with 7–17 activities for the handling of medical procedures; i.e., activities for requesting, scheduling and performing medical examinations as well as for validating their results.

Source 5 is a core process in vehicle development: product planning [6]. The process model (plotted on a 1.5 m × 5 m wallpaper) comprises several hundreds of activities for planning production facilities and resources with complex inter-dependencies, and the flow of about 50 relevant documents. Further, there exists a process handbook with detailed activity descriptions.

Source 6 refers to a case study on electronic change management (ECM) from the *automotive industry*. ECM process models were partially published by the German Automotive Industry [19]. Our material comprises 60 process models expressed in different notations like Event Process Chains and UML Activity Diagrams (with 2–32 activities).

Source 7 is a vehicle repair process from the automotive domain [24]. Overall, there exist around 900 variants of this process, 68 of which are documented in explicit process models (in a BPMN-like language).

3.1.3. Procedure for process model smell identification

We first created a list of candidate process smells by taking an existing list of code smells from the SE domain as starting point [17]. Since the focus of this paper is on the control-flow perspective, we only considered smells which are related to this perspective. In addition, we used the outcomes of an extensive literature study to support the importance of the proposed smells and to underline that the process model smells are really indicators of bad model quality. Next we thoroughly analyzed the above mentioned material to find empirical evidence for our process model smells and – if necessary – extended the candidate list of process model smells. Since we want our process model smells to help detecting common quality problems in process models, we required each of the smells to be observed at least *three times* in the different models from our sources. Therefore, only those smells, for which enough empirical evidence exists, are included in the final list of process model smells.

3.2. Process model smells

In the following we present the identified process model smells (cf. Fig. 4). Each smell is briefly described and then illustrated using material from the aforementioned data sources.[4] We subsequently discuss each smell along its supporting literature, also explicitly addressing related process model quality metrics. Forward references are provided to the refactoring techniques in Section 4 that can be used to address the respective smells.

3.2.1. PMS1: Non-intention Revealing Naming of Activity/Process Model

Description. Activities in a process model are normally tagged with textual labels. However, improper labels may not reveal the intended content or purpose to readers. This makes the model more difficult to understand.

Illustration. When analyzing the 70 process models from Source 1 we identified significant inconsistencies regarding activity names and labelling styles. For example, 16 process models

Process Model Smells
PMS1 - Non-intention Revealing Naming of Activity / Process Model
PMS2 - Contrived Complexity
PMS3 - Redundant Process Fragments
PMS4 - Large Process Models
PMS5 - Lazy Process Models
PMS6 - Unused Branches
PMS7 - Frequently Occurring Instance Changes
PMS8 - Frequently Occurring Variant Changes

Fig. 4. Catalogue of process model smells.

contained activities dealing with the scheduling of medical procedures (e.g., surgeries, medical examinations, drug administrations). Although all these activities had similar intentions, different labels and labelling styles were used (e.g., "make appointment", "appointment", "schedule examination", "fix day", "agree on surgery date", and "plan"). This, in turn, caused considerable efforts when reusing the models later in the context of a large harmonization project (see the illustration of the "Lazy Process Models" smell in Section 3.2.5).

Discussion. In literature, many guidelines exist stressing the importance of appropriate activity namings in process models [77,39,78]. Furthermore, empirical evidence exists that negative effects can occur if inappropriate labels are used. In an experimental study [41,43] the impact of different *grammatical* styles for activity labelling was investigated. When being asked to single out the labels in a process model that were ambiguous, respondents often referred to labels that did *not* first mention a verb, followed by an object. In contrast, labels that followed this "verb-object" style (e.g., *Determine Loan Conditions*) were rated as being significantly more useful. In addition, *length* of a text label can be an issue, as was established in another experiment [46]. While activities not following the "verb-object" style can be automatically detected [34], non-intention revealing labels have to be manually identified by process designers.

Relevant Refactorings. RF1 (Rename Activity), RF2 (Rename Process Model), RF7 (Re-label Collection).

3.2.2. PMS2: Contrived Complexity

Description. It is often possible to express a piece of control-flow logic within a process model in different ways. However, one alternative may be more difficult to comprehend for humans than another, despite their equivalence with respect to the (partial) execution traces they produce. Using the more complex alternative may negatively affect model understanding, and thus make maintenance of the model more difficult.

Illustration. In the model repositories from all considered data sources we were able to identify process models with unnecessarily complex control-flow structures, which could be simplified without changing the models' behavior. Examples of such complications include unnecessary AND-splits/-joins in connection with parallel branchings and superfluous control arcs expressing order relations that could be transitively derived by a set of other control arcs. Fig. 5a gives an example of unnecessary logical connectors; its simplified version is shown in Fig. 5b. It is worth mentioning that another factor impacting the difficulty humans had in respect to the comprehension of process models in the considered sources concerns the *layout* of the process model.

Discussion. Various studies have investigated the impact of structural model properties on model understandability. For example, [9] is centered around an adaptation of the cyclomatic number (one of the most widely used SE metrics) for business processes. Other research has analyzed process model understandability as aspect of maintainability, and has identified several correlations [8,1]. Further metrics take their motivation from
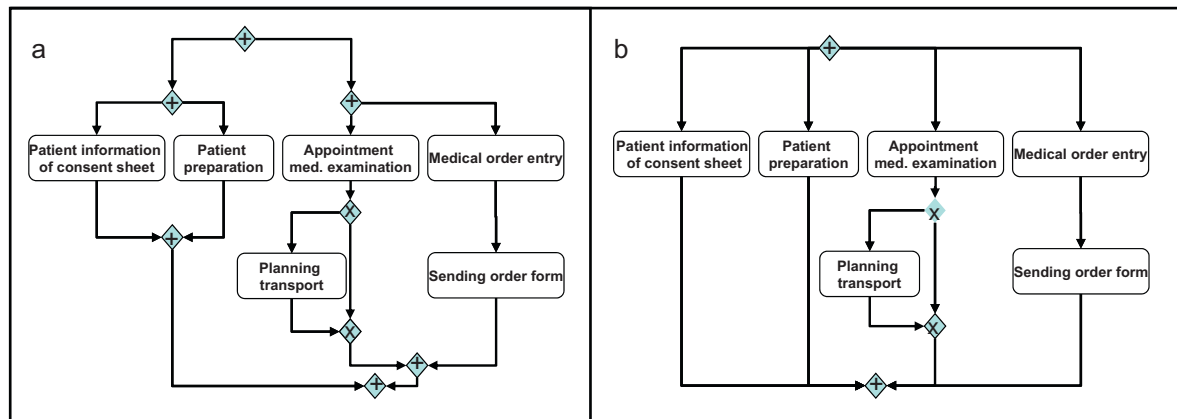
---

[4] Even though we illustrate each smell by way of an example, we have observed it multiple times when analyzing the numerous process models from our data sources.

Fig. 5. Process fragment from healthcare case.

cognitive research [91] or are based on concepts of modularity [93,88]. Most notably, an extensive set of metrics has been validated as factor influencing both error probability [48] and understandability [42]. The various validations show that factors like *structuredness* of a process model (i.e., the proper nesting of its gateways) and its *density* (i.e., the number of connections between its model elements) are influential. Both aspects can be manipulated by restructuring a process model; e.g., [91] presents three different, but trace-equivalent process models displaying different degrees of connectivity between model elements. Similarly, [75] proposes a metric for *structural appropriateness*, which can be used to determine how different models compare in their ability to capture a process in a compact and meaningful way.

Relevant Refactoring. RF3 (Substitute Process Fragment).

3.2.3. PMS3: Redundant Process Fragments

Description. Both within a single and across different process models, there may be fragments capturing the same control-flow logic. Whenever it is required to change this logic (e.g. due to changes in regulation or policy), the change must be propagated across all these occurrences. When overlooking some of them or when applying any of the changes incorrectly, inconsistencies arise which make successive maintenance even more problematic.

Illustration. Source 1 comprises 70 process models of a women's clinic. Despite their diversity the models contained many redundant fragments, which in most cases covered repetitive procedures relevant in a more general context. Examples include patient admission and discharge, medical reporting, and medical order handling (e.g., ordering drugs or a medical examination). Discussions with process owners showed that redundancies had been partially introduced through copying and pasting fragments from existing models when defining new ones. Furthermore, over time these cross-model redundancies led to problems in model maintenance due to oversized models as well as model inconsistencies.

Discussion. A common reason for redundancies entering process models is that multiple model variants are created for different scenarios [73]. Process parts may then be applied in a copy-paste fashion, which is indeed also the case for the illustration we provided above. As a consequence, even simple changes might require manual re-editing of process variants [35]. Accordingly, the advice to avoid redundancy in process models is widespread [3,29]. For example, [29] extracts typical modeling errors analyzing hundreds of process models. The researchers suggest that each activity, whenever possible, should only be defined once and be made available in some sort of global repository to avoid execution errors and to improve model understandability. In [89] the

footprint similarity metric is proposed to detect highly similar process models or process model parts, which can be used to detect this smell.

Relevant Refactorings. RF4 (Extract Process Fragment), RF5 (Replace Process Fragment by Reference), RF8 (Remove Redundancies).

3.2.4. PMS4: Large Process Models

Description. With an increasing number of activities process models become more difficult to understand and maintain.

Illustration. The product planning process from Source 5 comprises several hundreds of activities for planning production facilities. Interviews with process owners revealed that the current model contains several flaws, is known in its entirety to only very few experts, and is partially outdated. Moreover, the model is considered as being too large and difficult to maintain.

Discussion. Beyond the sources available to us, various instances of process models that have grown to a very large size have been recorded in literature. For example, the model in [79] initially consisted of more than 800 activities, but this number grew with 17% in an observed time period of two years. Even though it is natural for process models to grow in size along with their increased use, it is by now well-known that size of a process model is connected to understandability and correctness issues. An empirical study of a set of over 600 process models in an industrial repository provides evidence that larger, real-world process models tend to have more formal flaws (such as deadlocks or unreachable end states) than smaller ones [48]. Moreover, an empirical study investigating the effect of using modularity in process models (i.e., use of complex activities referring to subprocesses) has indicated that this eases understanding [63]. Some considerations are available on when a process model would have to be split up into subprocesses. In this context, practitioner books recommend modularizing process models with more than 5–15 [27] or 5–7 activities [77]. According to [44] models with more than 50 elements have an error probability of 50%. To support the process designer in finding this process model smell, *process model size* can be used as a metric. Based on the above described insights a process model size of 50 elements should be regarded as upper bound. However, modularization might also be effective for a smaller process model.

Relevant Refactoring. RF4 (Extract Process Fragment).

3.2.5. PMS5: Lazy Process Models

Description. Inclusion of many small process models will increase the overall number of models in a process repository.

This is bad for maintenance and it will make model retrieval more difficult.

Illustration. 15 out of 60 process models of Source 6 comprised only 3 or less activities. All these models were referred to by exactly one superordinated process. This rather large number of small process models aggravated both model maintenance and model training, and it was additionally accompanied by inconsistencies. Therefore, model harmonization, removal of redundancies, and reduction of the number of models were considered as key contributions towards improved model management by the involved stakeholders.

Discussion. Use of complex activities referring to subprocesses is known to improve the understanding of process models in comparison with models merely using atomic activities [63]. Clearly, decompositions which are too extreme (i.e., which result in many tiny process models) are not optimal in terms of maintenance and usability. While there is no source that specifies an optimal, lower bound for the number of activities in subprocesses, guidelines suggest that this number should range from 5 to 7 [77]. A metric that could be used to identify this smell is the *number of activities per subprocess*.

Relevant Refactoring. RF6 (Inline Process Fragment).

3.2.6. PMS6: Unused Branches (Unused Code in SE)

Description. Process models may specify behavior that never occurs in reality; i.e., such models are too large and complex for their purpose. This will have negative consequences for their understandability and maintainability.

Illustration. An analysis of the 46 process models representing clinical guidelines from Source 2 showed that some of the models contained branches that were never executed and which, therefore, unnecessarily inflated the models. Interestingly, in several cases the execution of the unused branches depended on a particular medical context (e.g., pregnancy). Since that particular context had already been covered by another, more specific process variant, the respective branches remained unconsidered.

Discussion. The problem of unused branches is closely linked to the issue of "overfitting" [83], which refers to situations where a process model contains behavior not found in a series of observations of the actual process. Clearly, if one can observe a process for only a limited amount of time or only with respect to few different instances, it does make sense for designers to create a process model that attempts to generalize those observations.

However, it is relatively easy to generate models that are too general, as shown in [75]. In this work the degree to which a model represents reality and does not become too generic is captured as metric, referred to as *behavioral appropriateness*.

Relevant Refactoring. RF10 (Remove Unused Branches).

3.2.7. PMS7: Frequently Occurring Instance Changes

Description. When executing a particular process instance it may become necessary to deviate from the logic predefined in the process model. A high frequency of such changes can, however, be problematic. It may indicate that the actual process model does not properly reflect the real process, which undermines its role as communication instrument.

Illustration. In patient treatment, clinical guidelines play an important role [32]. They aim at supporting physicians by providing recommendations for medical decision making and patient treatment based on existing evidence. However, physicians are not supposed to follow the process set out by a guideline step-by-step. Instead, they must estimate the patients' chances and risks, and ensure that their decisions are consistent with the patients' states (i.e., the specific treatment process depends on medical knowledge as well as on case-specific decisions). Consequently, physicians frequently adjust the treatment process defined by a guideline to the specific situation of the patient (i.e., the process is adapted at instance level). As example, consider a clinical guideline for urinary stone diagnostics taken from Source 3 (cf. Fig. 6a). This process has been implemented using the ADEPT2 adaptive process management system [11]. Physicians deviate from this process quite frequently, for example, in case a patient is pregnant or has an increased blood sugar level. In the former case, an additional lab test ("Blood glucose") is added and activity "Abdominal X-ray" is exchanged by activity "MRT". In the latter case, lab tests are added (i.e., "HbA1C" and "Blood glycemic profile"). Fig. 6b depicts the guideline taking pregnancy and diabetes into consideration.

Discussion. A much investigated PAIS feature concerns the deviation from predefined process logic during run-time. There are various reasons why exceptions occasionally occur that necessitate such changes [80]. Interestingly, a study of processes in the chip design industry [51] found that actual instance changes are often highly similar. This has been confirmed in other domains like healthcare, e-negotiation, and transportation [96]. When exceptions occur frequently, it is desirable to pull similar instance
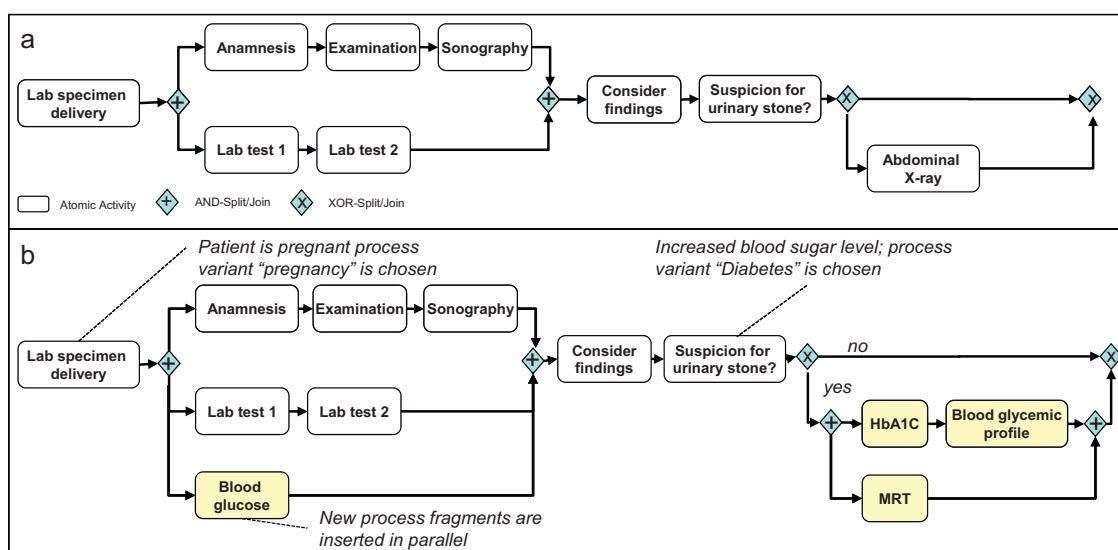

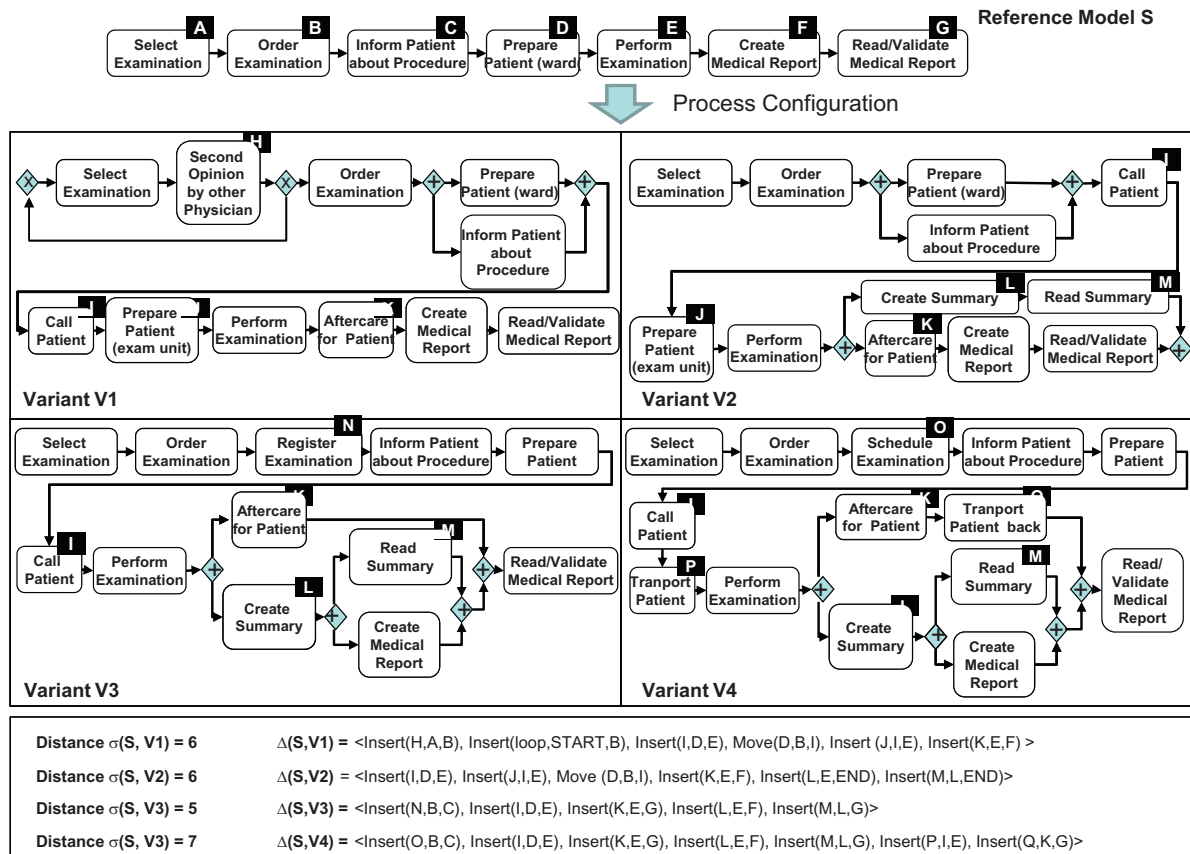**Fig. 6.** Example of clinical guideline "Urinary Stone Diagnostics".

Fig. 7. Examples of configured process variants for handling medical procedures.

changes up to the process type level. On the one hand, this improves semantic quality of the process model (i.e. it decreases the gap between modeled and real-world process). On the other hand, it reduces the need for future instance changes. This is advantageous, because a proper instance change might be rather difficult to achieve due to various constraints to be taken into account. ProCycle [96,67], for example, has been explicitly developed to support such discovery of desirable process model changes. To automatically detect this smell, the *number of instance changes* could be used as a metric.

Relevant Refactoring. RF11 (Pull Up Instance Change).

3.2.8. PMS8: Frequently Occurring Variant Changes

Description. Ongoing creation of multiple model variants leads to an enlargement of the size of the model repository aggravating its maintenance.

Illustration. In one of our case studies in a large clinical centre (cf. Source 4) we have identified more than 80 process variants for handling medical procedures (e.g., X-ray inspections or cardiological examinations). Fig. 7 depicts four variant models of Source 4 and their distances to a documented reference process model. Despite the high similarity of the four variants they are captured in separate process models. Discussions with process owners have shown that in the past even simple changes (e.g. due to new regulations or reengineering efforts) required error-prone, manual re-editing of a large number of logically related process variants. Over time, this had led to degeneration and divergence of the respective process models, which aggravated their maintenance significantly. As a consequence, costly manual refactorings became necessary. We observed similar problems with respect to model maintenance in Source 7.

Discussion. The number of process models in real-life repositories can be substantial [38,62]. One of the common reasons for this is the creation of multiple model variants for different scenarios

[73]. In [89], an industrial repository of 74 sales and distribution process models was investigated. Alone in this sample 50 pairs of process model variants were identified. This indicates the uncontrolled profusion that can take place when creating process variants. This exact issue is the subject of methods as described in [62,38,35,37], which aim to search and match process variants towards the creation of more generalized models. In this way, the size of model repositories can be controlled. For example, in the setting of a large financial organization it was possible to combine 15 different variants of the same offering into one process model, which was well-received by the users that maintain the repository [62]. The profusion of process variants can be determined by inspection of their *change distances*.

Relevant Refactoring. RF9 (Generalize Variant Change).

3.3. Summary of process model smells

Above, we have identified and discussed eight frequent process model smells. Fig. 8 summarizes this discussion including references to related work in the literature. For each smell, the source process model collections are mentioned in which we observed their occurrence. Each of the smells was at least supported by its occurrence in three different models. The column metrics mentions the indicators that are useful to detect the smell. Finally, in the last column, we point to Relevant Refactoring techniques. These techniques will be discussed in more detail in the next section.

4. Refactoring techniques

In the following, we describe 11 refactoring techniques which enable process designers to improve the quality of their models and to cope with the discussed model smells (cf. Fig. 9). An analysis of our sources, the process repositories from the healthcare and

Smell	Sources	Literature	Metrics	Refactorings
PMS1	1-7	[41,43,46,77,78]	verb-object style	RF1, RF2, RF7
PMS2	1-7	[1,8,9,42,48,88,90,91]	cyclomatic number, structuredness, density, structural appropriateness	RF3
PMS3	1, 4, 5, 6	[3,89]	footprint similarity	RF4, RF5, RF8
PMS4	5	[42,44,48,63]	size	RF4
PMS5	6	[63,77]	#activities / subprocess	RF6
PMS6	2, 3, 7	[75,83]	behaviorial appropriateness	RF10
PMS7	3	[51,97]	#instance changes	RF11
PMS8	4, 6, 7	[35,37,62]	change distance	RF9

Fig. 8. Summary of the discussion of the various smells

automotive domain, and the additional literature study have clearly shown that all identified refactorings are frequently needed in practice (cf. Section 3).

For each of the proposed refactorings we describe its intent and the process smell(s) it addresses, give illustrations, provide a description of the refactoring operation (with pre- and postconditions), and sketch its implementation. We organize our refactorings into three groups. The first one is introduced in Section 4.2 and contains *refactorings for process model trees*. Refactorings in this category can be applied to a single model or to entire process model trees (i.e., hierarchies of process models). The second group additionally relies on the support for reference process models. It provides a *refactoring that can be applied to a collection of process variants*. More precisely, this refactoring helps process designers in finding a process reference model that is close to the given variant collection (cf. Section 4.3). Finally, the third group describes two refactorings, which support *model evolution* by considering process history data (Section 4.4). Since respective refactorings make use of history data (i.e., execution traces), their application requires the presence of a run-time environment.

4.1. Preliminaries

In our context, refactorings constitute model transformations which are behavior-preserving if certain preconditions are met.

Several of our refactorings can be implemented based on process change patterns as introduced in [95,68]. However, since change patterns are usually not behavior-preserving our refactorings are imposing the required preconditions to ensure this.

Most refactorings are not applicable to arbitrary process fragments, but are restricted to single-entry, single-exit (SESE) regions or sequences of SESE regions (e.g., RF3, RF4, RF5 and RF8). Fig. 10 shows process model S and its decomposition into SESE regions (for details on SESE decomposition see [93]). SESE regions can be nested (e.g., R3, R4, R5 and R6 are contained in R2), sequentially composed (e.g., regions A, R2 and M), or disjoint (e.g., regions R3 and R4). In Fig. 10a refactorings are applicable to region R2, but not to a selection comprising regions R3, R4 and R5.

The resulting process structure tree PST_S is depicted in Figure Fig. 10b. Nodes in such a tree represent SESE regions of the process model, while edges represent the nesting of regions (see [93] for details).

Models S and S′ are called *structurally equivalent* if and only if they have the same process structure tree.

Definition 1 (Structural Equivalence).
Two process models S and S′ are structurally equivalent iff PST_S = $PST_{S'}$.

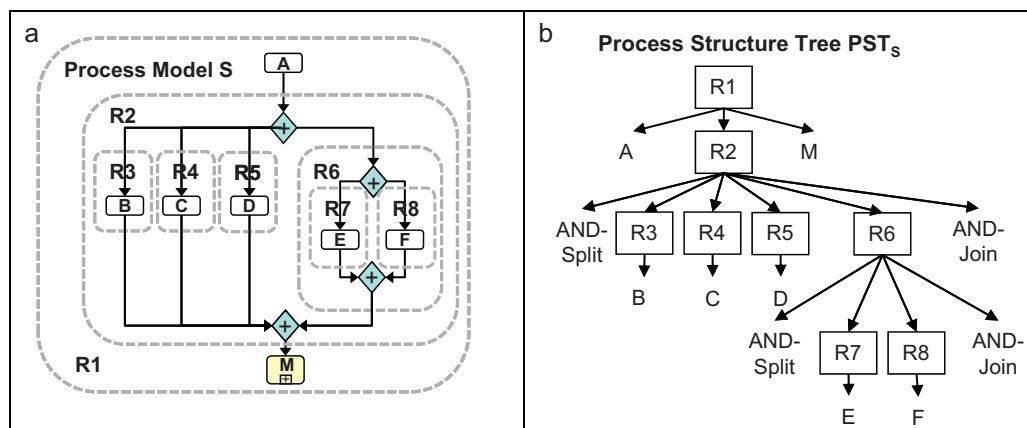|	
RF1: Rename Activity	RF7: Re-label Collection
RF2: Rename Process Schema	RF8: Remove Redundancies
RF3: Substitute Process Fragment	RF9: Generalize Variant Changes
RF4: Extract Process Fragment	RF10: Remove Unused Branches
RF5: Replace Process Fragment by Reference	RF11: Pull Up Instance Change
RF6: Inline Process Fragment	

Fig. 9. Refactoring catalogue.

Fig. 10. Process model with SESE regions.

To reason about behavior-preservation of the refactorings, it is essential to settle the notion of equivalence to be applied. Many such notions exist (e.g., trace equivalence, bisimulation, branching bisimulation). This paper will use *trace equivalence* as the main formal notion. For example, $\sigma_1 = \langle A, B, C, D, E, F, M \rangle$ and $\sigma_2 = \langle E, B, D, C, A, F, M \rangle$ both constitute traces producible on model S from Fig. 1a. Models S (cf. Fig. 1a) and S' (cf. Fig. 2) are called *trace equivalent* since the same set of traces can be produced based on S as well as on S'.

Definition 2 (Trace Equivalence).

Let $\mathcal{PS}$ be the set of all process models. Let further $\mathcal{A}$ be the total set of activities or – more precisely – activity labels based on which models $S \in \mathcal{PS}$ are specified (without loss of generality we assume unique labelling of activities). Let further $\mathcal{Q}_S$ denote the set of all possible execution traces producible on model $S \in \mathcal{PS}$. A trace $\sigma \in \mathcal{Q}_S$ is given by $\sigma = \langle a_1, \ldots, a_k \rangle$ (with $a_i \in \mathcal{A}$) where the temporal order of a_i in σ reflects the order in which activities a_i were completed over S. Two process models S and S' are trace equivalent iff $\mathcal{Q}_S = \mathcal{Q}_{S'}$.

Many of the described refactorings do not only affect a single process model, but an entire *process model tree*. To determine whether two – potentially hierarchical – process models S and S' are trace equivalent, the respective process model trees need to be expanded. To this end, each complex activity needs to be replaced by the (sub) process model it refers to. Consequently, the trace of an activity does not contain the complex activity directly, but the trace of the associated subprocess. A possible execution trace for model S in Fig. 1a is $\sigma_1 = \langle A, B, C, D, E, F, X, Y, Z \rangle$.

For refactorings RF10 and RF11 the notion of *trace equivalence* is not applicable since the behavior *producible* on the changed process model is altered. Therefore, we use *state compliance* [65] as formal notion instead. In the given context, it indicates whether the actual trace of a process instance could have been produced on the changed process model as well. More precisely, if all instances of a process model are state compliant with its refactored model version (i.e., the traces are re-producible on this model), its *observed* behavior remains unchanged.

Definition 3 (State Compliance).

Let I be a process instance with execution trace σ. Let further S be a process model. Then: I is state compliant with S iff σ is producible on S.

4.2. Refactorings for process model trees

First, we describe 8 refactorings for process model trees. Refactoring R F1 (Rename Activity) can be applied if the name of an activity is not intention revealing. Similarly, *RF2 (Rename Process Model)* enables designers to alter the name of a model. Using *RF3 (Substitute Process Fragment)* process designers can substitute a fragment within a model by another one which is simpler in structure, but has the same behavior. *RF4 (Extract Process Fragment)* enables designers to extract a process fragment into a subprocess to remove model redundancies, to foster reuse, and to reduce model size. By applying *RF5 (Replace Process Fragment by Reference)* a process fragment can be replaced by a complex activity referring to a (sub) process model containing the respective fragment. *RF6 (Inline Process Fragment)*, in turn, can be applied to collapse the hierarchy by inlining a fragment. *RF7 (Re-Label Collection)* is a composed refactoring, which supports re-labelling of selected activities within a collection of process models. Finally, *RF8 (Remove Redundancies)* enables the combined use of RF4 and RF5 in order to remove redundant fragments from multiple models in a model collection at once.

4.2.1. RF1/RF2 (Rename Activity/Process Model), RF7 (Re-label Collection)

Description. With RF1 the name of an activity can be changed if it is not intention revealing. If an activity occurs several times in a process model, all occurrences of that activity will be renamed. RF1 is comparable to the *Rename Method* refactoring in SE [17]. Similarly, RF2 enables designers to rename a model S into S'. A similar refactoring in SE is called *Rename Class* [7]. RF7, in turn, is a composed refactoring for re-labelling a particular activity in all models of a model collection. For this, RF1 is applied to all models containing the activities to be re-labelled.

Addressed Process Smell. Altogether these refactorings can be used to address smell *PMS1 (Non-Intention Revealing Naming)*.

Pre-conditions. RF1 requires that no activity from S is labelled with the new name. RF2, in turn, requires that no process model with label S' exists.

Implementation. Labels which are not following the "verb-object" style can be automatically refactored using techniques described in [33].

Behavior-Preservation. Renaming an activity does not alter the actual behavior of the model as only its label is changed; i.e., trace equivalence can be guaranteed when taking changed labels into account appropriately. To guarantee that RF2 does not alter process behavior all references to S need to be updated. Obviously, trace equivalence can be used as a formal notion for RF2 ensuring that behavior of the model collection remains unchanged.

Effects. Applying RF1 enables process designers to improve model understandability through more intention revealing labels and consequently to reduce errors and to decrease costs of change (cf. Section 3.2.1).

Illustration. Regarding the illustration provided for PMS1 (cf. Section 3.2.1), RF1 was used for harmonizing activity labels before extracting fragments and replacing them by subprocess references.

4.2.2. RF3: Substitute Process Fragment

Description. Using RF3, a fragment G can be substituted by another fragment G' with simpler structure, but showing same behavior (cf. Fig. 5). The *Substitute Algorithm* refactoring known from SE [17] is comparable to RF3.

Addressed Process Smell. Scenarios in which RF3 is useful include unnecessarily complex parallel branchings or superfluous control-flow arcs due to transitive relations, i.e., RF3 addresses *PMS2 (Contrived Complexity)*.

Pre-conditions. RF3 requires G and G' to be trace equivalent SESE fragments or sequences of SESE fragments.

Implementation. RF3 can be implemented based on process change pattern *Replace Process Fragment* as described in [95,68].

Behavior-Preservation. Guaranteed based on pre-conditions.

Effects. Substituting a fragment by a simpler one enables designers to improve model quality along several dimensions: removing unnecessary parallel branchings and control-flow arcs does not only increase model clarity, but also decreases model size and control-flow complexity (CFC).

Illustration. Fig. 5a gives an example of a process model with unnecessary logical connectors. Its simplified version (after applying RF3) is shown in Fig. 5b.

4.2.3. RF4 (Extract Process Fragment), RF5 (Replace Process Fragment by Reference), RF8 (Remove Redundancies)

Description. RF4 can be used to extract a process fragment G from any model S (e.g., to eliminate redundant fragments or to reduce size of model S). Applying RF4 results in the creation of a new (sub) process model S' implementing the fragment. In addition, in S the original fragment is replaced by a complex

activity referring to S'. RF5, in turn, is used to replace a process fragment by a trace-equivalent subprocess model. Finally, RF8 is a composed refactoring based on RF4 and RF5. It can be applied to a collection of models $S_1, \ldots, S_n$ in order to remove redundancies. For this, RF4 is applied to one of these models to extract the redundant fragment. To all other models, RF5 is applied for replacing the respective fragment by a reference to the (sub) process model created before. The intent of these refactorings is similar to *Extract Method* as known from SE [17].

Addressed Process Smell. RF4, RF5 and RF8 are potential remedies for process model smells *PMS3 (Redundant Process Fragment)* and *PMS4 (Large Process Model)*.

Pre-conditions. To guarantee that RF4 does not alter the behavior of the model tree, the fragment to be extracted must be a SESE region or a sequence of SESE regions (cf. Fig. 10). For applying RF5, the SESE fragment to be replaced and the corresponding (sub-) process model need to be trace-equivalent.

Implementation. RF4 can be implemented using change pattern *Extract Process Fragment* [95]. RF5, in turn, can be implemented based on change pattern *Replace Process Fragment* [95].

Behavior-Preservation. Guaranteed by pre-conditions.

Effects. Extracting parts of a process model often results in reduced control-flow complexity (CFC). Similarly, in SE the *Extract Method* refactoring is suggested as remedy for high cyclomatic complexity [20]. RF4 and RF5 can also be used to reduce the size of large models and overall number of nodes in the process repository by removing redundancies. Furthermore, removing redundancies reduces costs of future process changes.

Illustration. Regarding the illustration that was provided for PMS3 (see Section 3.2.3), it was possible to extract 9 redundant fragments relevant for more than one process model and to map them to separate (sub-) process models (RF4) (e.g., process models for admitting patients, creating discharge summaries, or handling medical orders). Taking the extracted process models, 30 redundant process fragments within the 70 process models could be replaced by references to the corresponding (sub-) process models (RF5). These refactorings led to a significant reduction of redundancies, a decrease of model sizes (while increasing the total number of process models), an increase of model consistency, and better overall maintainability.

4.2.4. RF6: Inline Process Fragment

Description. RF6 can be used to collapse the hierarchy of a model by inlining the process fragment, e.g., if it is not justifying its induced overhead. Similarly, in SE *Inline Method* [17] enables programmers to inline the body of a method. By inlining a fragment $S1$ into S the complex activity referring to $S1$ is substituted by the fragment corresponding to $S1$.

Addressed Process Smell. This refactoring can be applied to address *PMS5 (Lazy Process Model)*.

Pre-conditions. RF6 can be applied to complex activities (i.e., activities referring to a (sub) process model).

Implementation. RF6 can be implemented based on the *Inline Process Fragment* change pattern described in [95].

Behavior-Preservation. Trace equivalence can be used as formal notion.

Effects. RF6 enables designers to collapse the hierarchy of a process model tree resulting in a decrease of levels. Note that model size and control-flow complexity might increase when applying RF6.

Illustration. The models described in the illustration of PMS5 (see Section 3.2.5) can be significantly improved by applying RF6. In particular, the total number of process models can be reduced from 60 to 26 (containing 2–18 activities). Especially, the number of very small models (i.e., models with 2 or 3 activities) can be decreased from 15 to 4 models.

4.3. Refactoring of process variants

Another challenge is to manage the process variant models belonging to the same process family (cf. Fig. 1b). Typically, the model of a process variant is directly or indirectly derived through configuration from a given reference process model S_{ref}, i.e., by applying a sequence of change operations to S_{ref} (see Fig. 7 for an example from the healthcare domain). As discussed in the context of process model smell *PMS8 (Frequently Occurring Variant Changes)*, in many cases the process variants have to be maintained by their own, and even simple changes affecting multiple variants (e.g. due to new laws or re-engineering efforts) require error-prone, manual re-editing of a large number of related process variants. Over time this leads to a degeneration and divergence of the models, which further aggravates maintenance.

In general, the configuration of new variants or the adaptation of existing ones can be done most effectively if the reference model is kept close to the given variant collection. This, in turn, can be achieved if the *average change distance* between the reference process model S_{ref} and its corresponding variant models $V_1, \ldots, V_n$ is kept minimal; i.e., the average number of high-level change operations needed to transform S_{ref} into variant models $V_i, i = 1, \ldots, n$ is minimal [36,37]. In order to ensure this, continuous efforts have to be made to evolve the reference model accordingly. Otherwise, more and more redundant changes would have to be performed to different variant models in order to keep them aligned with the real-world processes. As example consider again Fig. 7. Obviously, the depicted variant models contain redundant changes (e.g., insertion of activity `Call Patient`) which should be pulled up to the reference model in order to reduce future configuration efforts and to decrease average distance between reference model and process variants.

Though the variant models of a process family are similar, unnecessary differences of their control flow structure can make refactorings RF4 and RF5 inapplicable in many situations. Therefore, an additional refactoring technique is needed, which supports designers in maintaining reference models.

4.3.1. RF9: Generalize Variant Changes

Description. RF9 enables designers to pull changes, which are common to several variants, up to the reference model (similar to *Pull Up Method* and *Push Down Method* known from SE [17]). This enables process designers to remove redundancies and to decrease costs of future changes. As example consider Fig. 7 where the depicted variant models have several changes in common. To each variant model, for instance, activities `Inform Patient about Procedure`, `Call Patient` and `Aftercare for Patient` have been added. When pulling respective changes up to the reference model the average distance between reference model and process variants can be reduced. Consider our example from Fig. 7 for which we can derive an optimized reference model by applying RF9 to the given variant collection. This optimized reference model is depicted on the top of Fig. 11. As can be further seen from the bottom of Figs. 7 and 11, respectively, average change distance between reference model and variants decreases when evolving the old reference model S to the new one (i.e. to S^*).

Addressed Process Smell. This refactoring can be applied to address *PMS8 (Frequently Occurring Variant Changes)*.

Implementation. RF9 necessitates a framework for managing reference models and the variants derived from them. First of all, techniques for analyzing process variants and for identifying process variant changes to be pulled up to the reference model are needed. In this context, we apply a family of advanced mining algorithms as described in [37,35]. Using the clustering approach as introduced in [35], a reference model S'_{ref} can be derived by mining a set of process variants $V_1, \ldots, V_n$ such that average
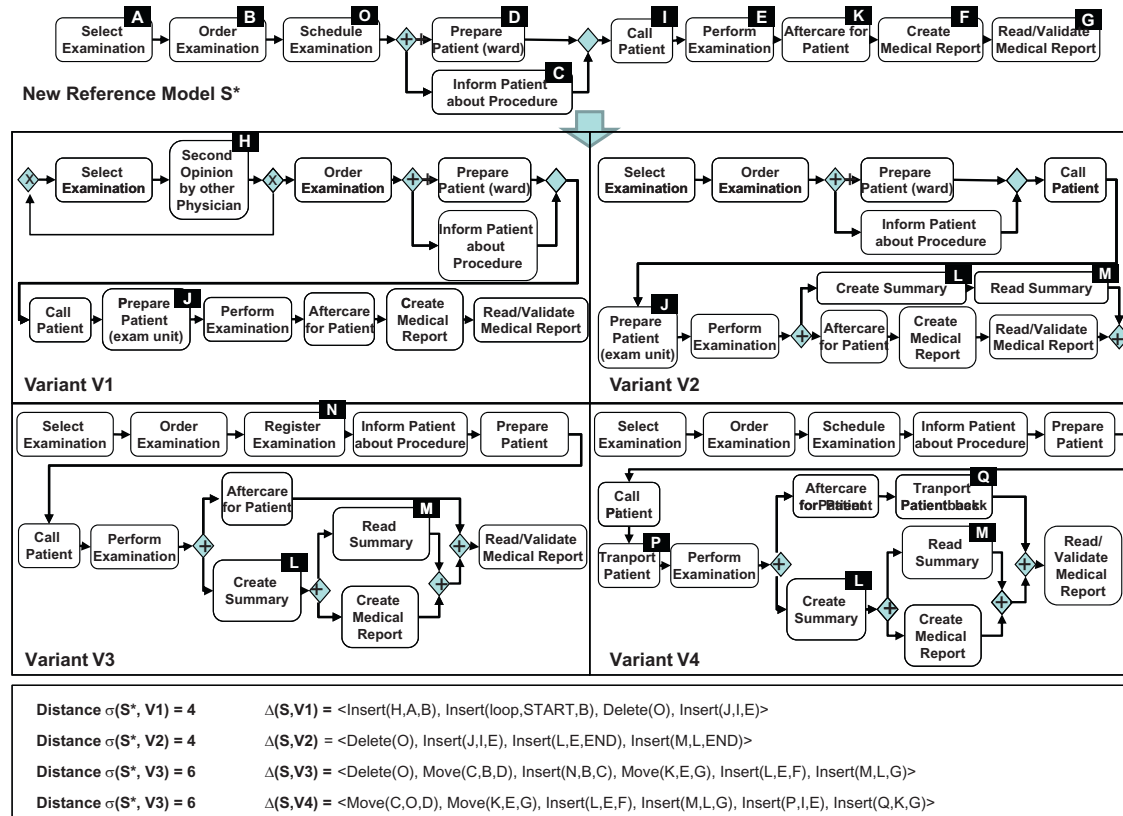
Fig. 11. Newly mined reference model.

distance between S'_{ref} and the variants becomes minimal. The heuristics approach described in [37], in addition, allows to take the old reference model into account as well; i.e. we are able to also control the (maximal) distance between old reference model and newly discovered one, which helps to avoid spaghetti-like process models.[5] Furthermore, when evolving a reference model S_{ref} accordingly, all variants need to be re-linked from S_{ref} to S'_{ref}, and for each variant its bias needs to be re-calculated in respect to S'_{ref} [96]. Finally, effective techniques are needed for internally representing a reference model and its variants.

Behavior-Preservation. Note that RF9 does not alter the variant behavior. Applying the updated bias of a variant V_i to S'_{ref} results in same variant-specific model as it can be obtained when applying the old bias to S_{ref}.[6] Thus trace equivalence can be used as formal notion.

Effects. The average change distance between a reference process model and its variants is reduced.

Illustration. We applied RF9 to the healthcare scenario as provided in the context of PMS8 and Source 4, respectively. In total, 84 process model variants were considered. Based on their relevance (i.e., the relative frequency with which process instances from the variant models were created), we assigned weights to the variant models ranging from 0.1% to 8.67%. Considering this, the original reference process model S was a simple process model comprising 7 activities (see Fig. 7 for S and four exemplarily chosen process variants). When evaluating this model in respect to the 84 variants, average change distance between S and the variant models corresponded to 5.3; i.e., per average we needed to apply 5.3 high-level change patterns (e.g., to add, delete or move

activities) to configure a variant model out of S. Applying RF9 to the collection of 84 variants resulted in a new reference model S^* (cf. top of Fig. 11) with average weighted distance of 2.79 between S^* and the process variants; i.e., RF9 performed well for this case and contributed to the inclusion of important changes in the new reference model and thus to less configuration efforts in future.

4.4. Refactorings for model evolution

This section describes refactoring techniques, which become applicable when process models are executed by PAISs and historic data on process instances is available in execution or change logs [85,66]. These logs can be analyzed and mined to discover potential refactoring options. In this context *RF10 (Remove Unused Branches)* enables process designers to remove unused paths from a process model (cf. *PMS6 – Unused Branches*) and *RF11 (Pull Up Instance Change)* enables generalization of frequent instance changes by pulling them up to the process type level (cf. *PMS7 – Frequently Occuring Instance Changes*). Several mining methods for discovering such situations already exist [85,37]. We therefore do not look at respective techniques, but use them for realizing refactorings based on historical data.

4.4.1. RF10: Remove Unused Branches

Description. RF10 enables designers to remove non-executed process fragments from a model S. While unused branches can be automatically detected, RF10 is not automatically applied, but designers have to ensure that the misalignment between model and log was not caused by design errors or an execution log not covering all relevant traces.[7]

---

[5] A technical description of these algorithms can be found in [35,37] and is out of the scope of this paper.

[6] This also becomes evident from the examples depicted in Figs. 7 and 11, respectively.

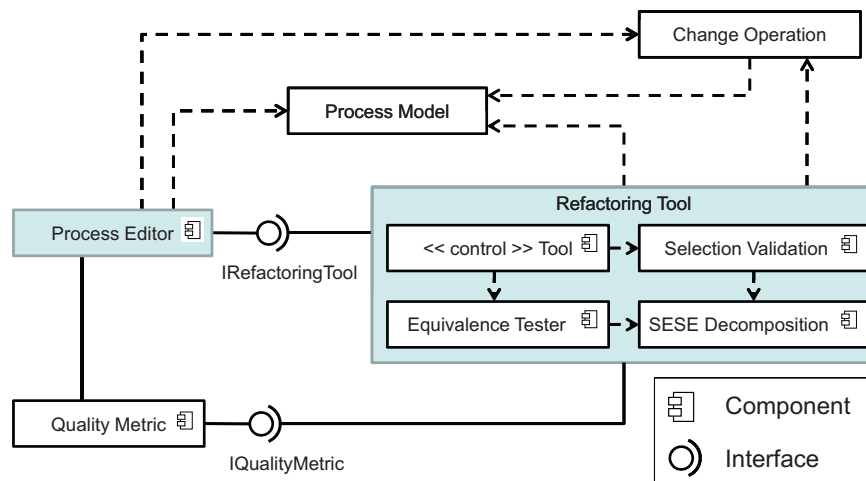[7] Note that we require access to an execution log in order to be able to identify unused branches.

Fig. 12. Architecture of refactoring tool component.

Addressed Process Smell. This refactoring provides a remedy for smell *PMS6 (Unused Branches)*.

Implementation. RF10 can be implemented based on the change pattern *Delete Process Fragment* [95,68] and standard process mining techniques [85].

Behavior-Preservation. Trace equivalence is not suitable as formal basis for RF10 since behavior p roducible on the respective process model is altered by RF10. Instead, we use the notion of *state compliance*. RF10 can be applied to S if the traces of all instances on S are re-producible on the new model (i.e., *observed* behavior remains unchanged). Generally, state compliance can be guaranteed when removing previously unused execution paths.

Effects. Applying RF10 decreases model size and control flow complexity.

Illustration. Regarding the illustration provided for PMS6 (cf. Section 3.2.6), we were able to apply RF10 to some of the 46 process models in order to remove branches that had never been chosen for execution in any of the logged process instances. However, the deletion of such unused branches had to be explicitly approved by process owners (e.g., to avoid the removal of relevant exceptional paths that were not covered in another way within the total collection of the 46 process models).

4.4.2. RF11: Pull Up Instance Change

Description. RF11 can be used to generalize frequently occurring instance changes by pulling them up to the process type level (similar to RF9 where variant changes are generalized).

Addressed Process Smell. RF11 serves as a remedy for smell *PMS7 (Frequently Occurring Instance Changes)*.

Implementation. Implementation of RF11 is similar to the one of RF9, but requires change logs to reconstruct the models of the respective process instance collection.

Behavior-Preservation. Trace equivalence cannot be used to exclude errors when applying RF11. By pulling changes from instance level to type level, producible behavior is always altered. Therefore, state compliance (cf. Definition 3) is used as formal notion. Like RF9, RF11 has the potential for full automation.

Effects. Like for RF9, the goal is to reduce average and total change distance between the process model and instance-specific models; e.g., to learn from instance changes and to reduce future need for adapting instances [96].

Illustration. Since instance adaptations occurred frequently in the scenario introduced in the context of PMS7 (cf. Section 3.2.7), it was decided to consider them in a specific variant of the original guideline (cf. Fig. 6b).

5. Implementation

To demonstrate the feasibility of our refactoring techniques we implemented a proof-of-concept prototype and applied it to existing process models. Section 5.1 describes the architecture of our prototype, while Section 5.2 demonstrates it based on a walk-through scenario.
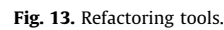
5.1. Architecture

Our refactoring tool has been implemented as an Eclipse RCP application on top of the SecServ platform.[8] Our prototype provides support for all refactorings depicted in Fig. 9. To support users in applying behavior-preserving refactorings, our prototype only enables refactorings which fulfill the relevant preconditions. In addition, users are supported in assessing the effects of a particular refactoring on selected quality metrics. Fig. 12 shows the architecture of our refactoring tool component. The *Refactoring Tool component* is integrated in a loosely coupled manner into the *Process Editor component* of SecServ via the extension point mechanism of the Eclipse RCP platform.

Both the *Refactoring Tool component* and the *Process Editor component* rely on the *Process Model component* containing all the model elements from the process model repository and the *Change Operation component* containing all supported change patterns (e.g., Insert Process Fragment, Delete Process Fragment and Move Process Fragment).

The *Tool component* is the central part of the *Refactoring Tool component* containing the functionality of the refactorings. The *Tool component* relies on the *Selection Validation component*, which checks whether a particular refactoring can be executed depending on the currently selected elements (i.e., it checks its pre-conditions). In the graphical user interface of the *Process Editor component* only those refactorings become enabled which fulfill the pre-conditions.

The *Tool component* also relies on the *Equivalence Tester component*, which checks whether two process models or process model fragments are structurally equivalent, i.e., have the same process structure tree (cf. Definition 1) or expose the same behavior (cf. Definition 2). Both the *Selection Validation component* and the *Equivalence Tester component* rely on the *SESE decomposition component* which computes SESE regions for a given process model and process model fragment, respectively, and which constructs a corresponding process structure tree.

Fig. 13. Refactoring tools.

To evaluate the effect of a particular refactoring with respect to selected quality metrics the *Refactoring Tool component* uses the *IQualityMetric interface* provided by the *Quality Metric component*. Note that the *Quality Metric component* is easily extensible, i.e., further quality metrics can be added by implementing the *IQualityMetric interface*.

As illustrated in Fig. 13, the current version of our refactoring tool component provides support for all refactorings described in this paper. In addition, the component can be easily extended with additional refactorings by implementing the *IRefactoringTool interface* and by adding a new extension. From a technical point of view the provided refactoring tools can be divided into three groups. First of all, *ElementRefactoringTools* modify a single activity. These refactorings are not enabled if the selected element is a gateway or an edge (i.e., RF1, RF6 and RF7). Second, *FragmentRefactoringTools*, in turn, are only applicable to SESE fragments (i.e., RF3, RF4, RF5, and RF8). To support users in applying behavior-preserving refactorings these refactorings are only enabled when a SESE is selected. Finally, *ProcessModelRefactoringTools* modify the whole process model. As a consequence, the selection of elements has no effect on the availability of these refactorings (i.e., RF2, RF9, RF10, and RF11).

5.2. Walk-through scenario

To better illustrate the main functionalities of our prototype we describe a walk-through scenario. This scenario is based on a simplified version of the pre take-off process for a general aviation flight under visual flight rules (VFR) and is briefly described in the following.

Before conducting a general aviation flight the pilot first has to check the weather. Optionally, the pilot can then file the flight plan.

This is followed by a preflight inspection of the airplane. For large airports the pilot calls clearance delivery to get the engine start clearance. If an airport has a tower control the pilot has to contact ground to get taxi clearance, otherwise she has to announce taxiing. This is followed by taxiing to run-up area and run-up inspections ensuring that the airplane is ready for flight. If the airport has a tower, the tower is contacted to get take-off clearance, otherwise take-off intentions have to be announced. Finally, the pre take-off process finishes with the take-off of the airplane.

During the pre-flight inspections the pilot can detect problems with the airplane. If the problems are severe, the flight is immediately cancelled. Otherwise, if the airplane can move under its own power, it drives to the repair station. Alternatively, the airplane is either towed to the repair station or a mechanician comes to the airplane in order to deal with the problem. After the repair the flight is re-started with checking the weather.

Fig. 14 depicts the pre take-off process as described above including several process model smells. First, the model does not strictly follow the verb-object style of naming activities, which relates to PMS1 (*Non-Intention Revealing Naming of Activity*). For example, activity `Repair` violates this convention. Second, the process model contains a redundant process model fragment (i.e., Fragments 1 and 2), which is an example of PMS3 (*Redundant Process Model*). Third, the depicted model is rather large and complex constituting an example of PMS4 (*Large Process Model*).

To address these smells, the process designer loads the model into our refactoring tool. To remove PMS1 (*Non-Intention Revealing Naming of Activity*) the process designer selects the respective activity and renames it to `Repair Airplane` using RF1 (*Rename Activity*). To deal with PMS4 (*Redundant Process Fragment*), the designer selects Fragment 1 and chooses RF4 (*Extract Process Fragment*) from the list of enabled refactorings to extract Model
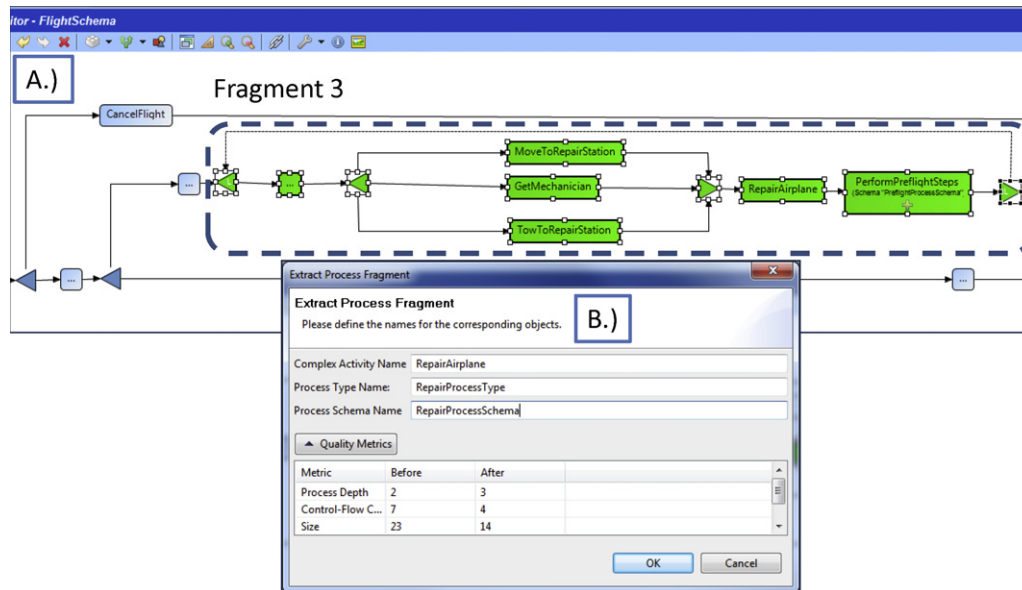

Fig. 14. Original model of pre take-off process.
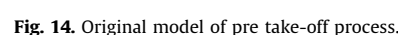
Fig. 15. Model before extracting repair process.

PreightProcessSchema (RF4 is enabled since the selection constitutes a SESE fragment and thus fulfills the pre-conditions for RF4). In a next step, the designer selects Fragment 2 and chooses refactoring RF5 (*Replace Process Fragment by Reference*) to replace this fragment with a reference to Model PreightProcessSchema (cf. Fig. 17).

Although these two changes have already reduced the size of the pre take-off process by eight nodes, the process model is still rather large and complex (PMS 6 – *Large Process Model*). Therefore, the designer decides to also apply refactoring RF4 (*Extract Process Fragment*) to Fragments 3–6, since each of them comprises several activities logically belonging together. For example, Fragment 3 deals with the reparation of an airplane if issues are detected during the preflight inspections.

Fig. 15 shows a screen of our prototype illustrating the extraction of all activities dealing with the reparation of the airplane (selected activities in Fig. 15A) into a distinct Model RepairProcessSchema (cf. Fig. 16B). Before conducting the

change, the designer is informed about the effects of this refactoring on the pre take-off process in terms of selected metrics (e.g., reduction of model size by nine) (cf. Fig. 15B). By pressing the OK button the change is performed and the updated model is stored in the repository. The result of this refactoring is depicted in Fig. 16.

Fig. 17 illustrates the model after applying all refactorings described above. A summary of the conducted refactorings and their effects on quality metrics is depicted in Fig. 18.

6. Related work

Refactoring techniques for improving software design were first proposed by Opdyke [52]. He suggested a set of refactorings for C++ which are semantic preserving if certain preconditions are met. The first notable refactoring tool has been the Refactoring Browser [7] for Smalltalk, which automatically performs the refactorings proposed by Opdyke plus some additionally techniques [69]. As all
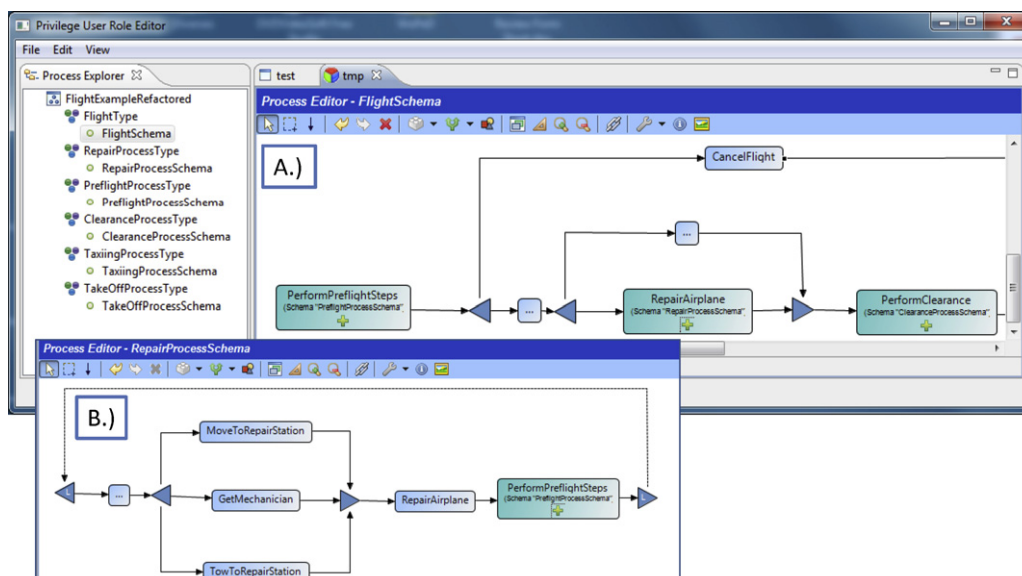

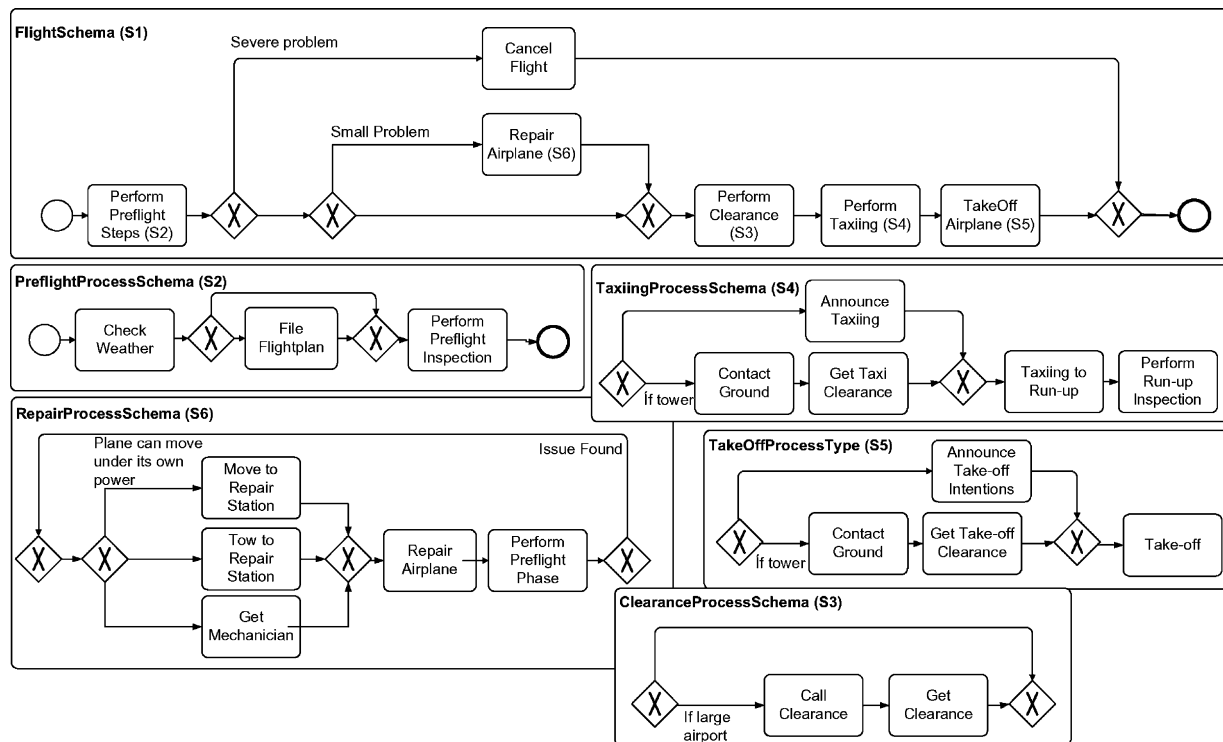**Fig. 16.** Model after extracting repair process.

Fig. 17. Model after refactoring.

refactorings provided by this tool constitute behavior-preserving transformations it is ensured that no errors or information losses are introduced. Tool support for languages like C++ and Java have recently emerged. The provided refactorings usually cannot be proven to be completely behavior-preserving. Therefore, refactorings need to be backed up by automated regression tests to detect behavioral changes in the software and to avoid errors [17]. Most of our refactorings, in turn, make use of pre-conditions to ensure that the behavior of the process models from the repository is not altered.

Closely related to the refactoring techniques considered in this paper are program and code optimizations, which constitute behavior-preserving program transformations [4]. In contrast to refactorings which focus on improvement of design artifacts, code optimization aims to improve the execution efficiency at runtime. While this topic has not yet been systematically investigated in the context of process-aware information systems, there are techniques and considerations available that relate to this topic [28]. Code optimization is an issue for the deployment of an executable process model to a process engine [25]. For instance, it is more efficient to transform graph-oriented links in a BPEL process into sequences if possible, because checking of link status is more resource intense than simply jumping to the next activity in a sequence when executing a process. In this paper, though, we stick to the perspective of a process designer aiming to organize process models in a comprehensible way.

Similar to program refactorings, process model refactorings constitute transformations, which are behavior-preserving if certain preconditions are met. Existing approaches focus on UML model transformations [81], while refactoring has not been elaborated in detail for business process models. There exist a few approaches which provide specific refactorings in a narrow context (e.g., a particular process modeling formalism). In [16] refactoring techniques for Event-driven Process Chains are described. Unlike our refactorings, these proposals require additional modeling elements. Refactoring techniques have also been discussed in connection with model merging [30]. The proposed transforma-

tions aim at improved process design, but are not necessarily behavior-preserving. The ADEPT process management system, in turn, applies simple refactorings in the context of process changes to avoid smell PMS2 [58]; i.e., the structure of a process model is simplified after the application of a sequence of changes, while preserving model behavior. Finally, [6] discusses graph transformations similar to RF4 and RF6 in the context of process views.

Several refactoring techniques are discussed in [15], however, their scope is slightly different from our refactorings; e.g., refactorings "Merge Process Ends", "Join Process Ends" and "Close Branches" focus on model completion and refactoring "Automatically Order Branches" deals with layouting issues. Similarly, [64] discusses options for re-layouting process models in order to increase model comprehension.

Behavior-preserving model transformations have been proposed in [5] to make Petri nets more compact. Several approaches for deriving structured models from unstructured ones are discussed in the context of BPMN to BPEL transformations. These include refined process structure tree decomposition [93], untangling unstructured loops [100], and further transformation rules [88,53,56,55]. Synthesis can be used to transform a Petri net via a transition system into another behavior-equivalent Petri net. Respective techniques allow to eliminate unnecessary net elements (e.g., silent activities, unnecessary places) [10] or to discard OR-joins from process models [47] (and can therefore be used to address smell PMS2).

The specific requirements of capturing process model variants have been addressed in different modeling approaches. In this work, we assume a generic process model to capture the behavioral alternatives of different variants. This is similar to the approach taken in work on the configuration of process models [22,86,87,24], while other approaches define dedicated variation elements on the process modeling language level. Such languages include Configurable EPCs [74,31], aggregated EPCs [62], and the variant rich process models [76], which pick up ideas and concepts from modeling of software product families and feature diagrams [26,94]. The definition of generic process models relates to

Before Refactoring	Refactorings	After Refactoring
PMS1 – 1 violation of verb-object style	RF1 (1x)	All activities labeled according to verb-object style
PMS3 – Fragments 1 and 2 are redundant	RF4 (1x) RF5 (1x)	No redundant fragments Reduction of model size by 8
PMS4 – Process model with 42 nodes (22 activities, 18 gateways, 1 start node, 1 end node)	RF4 (4x)	Flight Schema (S1): size 12 Preflight Process Schema (S2): size 7 Clearance Process Schema (S3): size 6 Taxiing Process Schema (S4): size 9 Take-Off Process Schema (S5): size 8 Repair Process Schema (S6): size 11

Fig. 18. Overview of conducted refactorings.

identifying similar process models with overlapping behavior [89,14,12,98] and integrating them into a single model [57,45,21,70].This integration is a particular instance of RF8, which aims to remove redundancies.

Process model smells are closely related to anti-patterns [29,82], since both describe indicators for low process model quality. However, most of the described patterns constitute real modeling errors (potentially leading to deadlocks) and cannot be resolved through behavior-preserving refactorings. As an instantiation of RF1, the work in [33] defines an automatic approach to reformulate activity labels using techniques from natural language processing. Finally, existing BPM tools only provide limited refactoring support. Renaming of activities and process models is supported by most tools (e.g., ARIS). However, more advanced refactoring support has been missing in most existing tools so far. Some support for process model refactorings is provided by the IBM Pattern-based Process Model Accelerators extension for WebSphere Business Modeler [15].

7. Summary and outlook

With the increasing adoption of PAISs and the emergence of large process repositories systematic support for model management is getting increasingly important. We introduced 8 process model smells, supported by empirical evidence in the form of several large case studies, to assist process designers in detecting symptoms of low process model quality. Moreover, we proposed 11 refactorings specifically suited for large process repositories. To demonstrate the feasibility of the proposed refactoring techniques we provided a proof-of-concept prototype to support users in both identifying refactoring options and applying behavior-preserving or compliance-ensuring refactorings.

The smells and refactorings we propose, along with the presented tool support, should be seen as a means to support organizations to better deal with their emerging process model repositories. As mentioned, it is increasingly realistic that business users without any deep modeling skills will be developing process models. While there are several benefits that can be associated with this practice, we argue that the quality of process models – regardless of their originators – will deteriorate over time. Therefore, we argue that there will always be a need for a small proportion of people with more advanced modeling knowledge to counter-balance this detorioration; the proposed techniques in this paper aim to support experienced modelers during process modeling. The analogy from the business domain is that while business professionals nowadays use advanced IT tools that required specialized use only some years ago (e.g. web development tools, databases, etc), the role of system administrators has not died out and has arguably become more important than before – precisely because of the lack of deep IT knowledge with casual users.

Even though we considered several different data sources from the healthcare domain and from automotive engineering having

different characteristics, it cannot be ruled out completely that other domains might show different characteristics which are missing from the current set of models. The goal of our paper is not to provide a complete list of process model smells. Instead our claim is to provide a list of process smells which can be typically found in practice and which provide indication for poor process model quality. To further validate our process model smells as well as refactoring techniques expert interviews are planned.

Future work includes the evaluation of our proof-of-concept prototype using case studies. We further plan to integrate the presented techniques with other repository services including process model adaptation [95], process model evolution [65], and process change mining [65]. Our overall goal is to provide integrated repository support for the management of process models throughout the entire process life cycle.

References

[1] E.R. Aguilar, F. García, F. Ruiz, M. Piattini, An exploratory experiment to validate measures for business process models, in: Proc. RCIS'07, 2007.
[2] K. Beck, Extreme Programming Explained, Addison Wesley, 2000.
[3] J. Becker, M. Kugeler, M. Rosemann, Process Management: A Guide for the Design of Business Processes, Springer, 2003.
[4] J. Bentley, Writing Efficient Programs, Prentice Hall Ptr, 1984.
[5] G. Berthelot, Transformations and decompositions of nets, in: W. Brauer, W. Reisig, G. Rozenberg (Eds.), Advances in Petri Nets 1986, Part I, Springer, 1987.
[6] R. Bobrik, Configurable visualization of complex process models, Ph.D. thesis, University of Ulm, 2008.
[7] J. Brant, D. Roberts, Refactoring Browser, st-www.cs.uiuc.edu/users/brant/refactoringbrowser/.
[8] G. Canfora, F. García, M. Piattini, F. Ruiz, C. Visaggio, A family of experiments to validate metrics for software process models, Journal of Systems and Software 77 (2) (2005) 113–129.
[9] J. Cardoso, chap. Evaluating workflows and web process complexity, in: Workflow Handbook 2005, Future Strategies, Inc., 2005, pp. 284–290.
[10] J. Cortadella, M. Kishinevsky, L. Lavagno, A. Yakovlev, Deriving petri nets from finite transition systems, IEEE Transactions on Computers 47 (8) (1998) 859–882.
[11] P. Dadam, M. Reichert, The ADEPT project: a decade of research and development for robust and flexible process support—challenges and achievements, Computer Science – Research and Development 23 (2) (2009) 81–97.
[12] R. Dijkman, M. Dumas, B.F. van Dongen, R. Käärik, J. Mendling, Similarity of business processmodels:metrics and evaluation, Information Systems 36 (2) (2011) 498–516.
[13] E. Dijkstra, A Discipline of Programming, Prentice-Hall, 1976.
[14] M. Dumas, L. García-Bañuelos, R.M. Dijkman, Similarity search of business process models, IEEE Data Engineering Bulletin 32 (3) (2009) 23–28.
[15] C. Favre, T. Gschwind, J. Koehler, et al., Faster and better business process modeling with the IBM pattern-based process model accelerators, in: Proc. BPMDemos2009, 2009.
[16] P. Fettke, P. Loos, Refactoring von Ereignisgesteuerten Prozessketten, in: Proc. EPK'02, 2002.
[17] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.
[18] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Addison-Wesley, 1994.
[19] German Association of the Automotive Industry (VDA), Engineering Change Management. Part 1: Engineering Change Request (ECR), vol. 1.1., Doc. No. 4965, December 2005.
[20] A. Glover, Refactoring with Code Metrics, www.ibm.com/developerworks/java/library/j-cq05306/, 2006.
[21] F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, Merging event-driven process chains, in: Proc. CoopIS'08, vol. 5331 of LNCS, Springer, 2008.

[22] F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, M.L. Rosa, Configurable workflow models, International Journal of Cooperative Information Systems 17 (2) (2008) 177–221.

[23] A. Guceglioglu, O. Demirors, Using software quality characteristics to measure business process quality, in: Proc. BPM'05, 2005.

[24] A. Hallerbach, T. Bauer, M. Reichert, Capturing variability in business process models: the Provop approach, Journal of Software Maintenance and Evolution: Research and Practice 22 (6–7) (2009) 519–546.

[25] R. Hauser, J. Koehler, Compiling process graphs into executable code, in: Proc. GPCE'04, 2004.

[26] S. Hsiao, E. Liu, A structural component-based approach for designing product family, Computers in Industry 56 (1) (2005) 13–28.

[27] N. Kock Jr., Product flow, breadth and complexity of business processes: an empirical study of 15 business processes in three organizations, Business Process Re-engineering & Management Journal 2 (2) (1996) 8–22.

[28] J. Koehler, R. Hauser, Untangling unstructured cyclic flows—a solution based on continuations, in: Proc. CoopIS'04, 2004.

[29] J. Koehler, J. Vanhatalo, Process anti-patterns: How to avoid the common traps of business process modeling, Tech. Rep. Report RZ-3678, IBM Zurich Research Lab, 2007.

[30] J. Küster, J. Koehler, K. Ryndina, Improving business process models with reference models in business-driven development, in: BPM'06 Workshops, 2006.

[31] M.L. Rosa, M. Dumas, A.H.M. ter Hofstede, J. Mendling, Configurable multi-perspective business process models, Information Systems.

[32] R. Lenz, M. Reichert, IT support for healthcare processes—premises, challenges, perspectives, Data & Knowledge Engineering (1) (2007) 39–58.

[33] H. Leopold, S. Smirnov, J. Mendling, Refactoring of activity labels in business process models, in: 15th International Conference on Applications of Natural Language to Information Systems (NLDB 2010), 2010.

[34] H. Leopold, S. Smirnov, J. Mendling, Recognizing activity labeling styles in business process models, Enterprise Modelling and Information Systems Architectures – International Journal (EMISA Journal), in press.

[35] C. Li, M. Reichert, A. Wombacher, Discovering reference process models by mining process variants, in: Proc. ICWS'08, 2008.

[36] C. Li, M. Reichert, A. Wombacher, On measuring process model similarity based on high-level change operations, in: Proc. ER'08, 2008.

[37] C. Li, M. Reichert, A. Wombacher, Discovering reference models by mining process variants using a heuristic approach, in: Proc. BPM'09, 2009.

[38] R. Lu, S. Sadiq, Managing process variants as an information resource, in: Proc. BPM'06, 2006.

[39] T. Malone, K. Crowston, G. Herman, Organizing Business Knowledge: The MIT Process Handbook, MIT Press, 2003.

[40] J. Mendling, Empirical studies in process model verification, in: Proc. ToPNoC II, 2009.

[41] J. Mendling, H.A. Reijers, How to define activity labels for business process models? in: Proc. AIS SIGSAND'08, 2008.

[42] J. Mendling, H.A. Reijers, J. Cardoso, What makes process models understandable? in: Proc. BPM'07, 2007.

[43] J. Mendling, H.A. Reijers, J. Recker, Activity labeling in process modeling: empirical insights and recommendations, Information Systems 35 (4) (2010) 467–482.

[44] J. Mendling, H.A. Reijers, W.M.P. van der Aalst, Seven process modeling guidelines (7PMG), Information and Software Technology 52 (2) (2009) 127–136.

[45] J. Mendling, C. Simon, Business process design by view integration, in: Proceedings of BPM Workshops 2006, vol. 4103 of LNCS, 2006.

[46] J. Mendling, M. Strembeck, Influence factors of understanding business process models, in: Proc. BIS'08, 2008.

[47] J. Mendling, B.F. van Dongen, W.M.P. van der Aalst, Getting rid of or-joins and multiple start events in business process models, Enterprise IS 2 (4) (2008) 403–419.

[48] J. Mendling, H. Verbeek, B.F. van Dongen, W.M.P. van der Aalst, G. Neumann, Detection and prediction of errors in EPCs of the SAP reference model, Data & Knowledge Engineering 64 (1) (2008) 312–329.

[49] T. Mens, P.V. Gorp, A taxonomy of model transformation, Electronic Notes in Theoretical Computer Science 152 (2006) 125–142.

[50] T. Mens, T. Tourwe, A survey of software refactoring, IEEE Transactions on Software Engineering 30 (2) (2004) 126–139.

[51] M. Minor, A. Tartakovski, D. Schmalen, R. Bergmann, Agile workflow technology and case-based change reuse for long-term processes, International Journal of Intelligent Information Technologies 4 (1) (2008) 80–98.

[52] W.F. Opdyke, Refactoring: A program restrucuring aid in designing object-oriented application frameworks, Ph.D. thesis, Univ. of Illinois, 1992.

[53] C. Ouyang, M. Dumas, W.M.P. van der Aalst, A.H.M. ter Hofstede, J. Mendling, From business process models to process-oriented software systems, ACM Transactions on Software Engineering and Methodology 19 (1) (2009) doi:10.1145/1555392.1555395.

[54] D. Parnas, Software aging, in: Proc: ICSE'94, 1994.

[55] A. Polyvyanyy, L. García-Bañuelos, M. Dumas, Structuring acyclic process models, in: Proc. BPM'10, 2010.

[56] A. Polyvyanyy, L. García-Bañuelos, M. Weske, Unveiling hidden unstructured regions in process models, in: Proc. CoopIS'09, 2009.

[57] G. Preuner, S. Conrad, M. Schrefl, View integration of behavior in object-oriented databases, Data & Knowledge Engineering 36 (2) (2001) 153–183.

[58] M. Reichert, P. Dadam, ADEPT$_{flex}$ – Supporting dynamic changes of workflows without losing control, Journal of Intelligent Information Systems 10 (2) (1998) 93–129.

[59] M. Reichert, P. Dadam, B. Schultheiss, I. Konyen, Analysis of healthcare processes in a woman's clinic. DBIS No. 27, 28, 29, 16, 15, 14, 7, 6, 5 (1996–1997).

[60] M. Reichert, S. Rinderle-Ma, P. Dadam, Flexibility in process-aware information systems, Transactions on Petri Nets and Other Models of Concurrency II, vol. 5460 of Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, 2009, pp. 115–135.

[61] H.A. Reijers, Design and control of workflow processes: business process management for the service industry, Springer, 2003.

[62] H.A. Reijers, R. Mans, R. van der Toorn, Improved model management with aggregated business process models, Data and Knowledge Engineering 68 (2) (2009) 221–243.

[63] H.A. Reijers, J. Mendling, Modularity in process models: review and effects, in: Proc. BPM'08, 2008.

[64] S. Rinderle, R. Bobrik, M. Reichert, T. Bauer, Business process visualization—use cases, challenges, solutions, in: ICEIS (3), 2006.

[65] S. Rinderle, M. Reichert, P. Dadam, Correctness criteria for dynamic changes in workflow systems—a survey, Data and Knowledge Enginnering 50 (1) (2004) 9–34.

[66] S. Rinderle, M. Reichert, M. Jurisch, U. Kreher, On representing, purging, and utilizing change logs in process management systems, in: Proc. BPM'06, 2006.

[67] S. Rinderle, B. Weber, M. Reichert, W. Wild, Integrating process learning and process evolution—a semantics based approach, in: Proc. BPM'05, 2005.

[68] S. Rinderle-Ma, M. Reichert, B. Weber, On the formal semantics of change patterns in process-aware information systems, in: Proc. ER'08, 2008.

[69] D. Roberts, J. Brant, R. Johnson, A refactoring tool for Smalltalk, Theory and Practice of Object Systems (4) (1997) 253–263.

[70] M.L. Rosa, M. Dumas, R. Uba, R.M. Dijkman, Merging business process models, in: Proc. CoopIS'10, vol. 6426 of LNCS, Springer, 2010.

[71] M.L. Rosa, J. Lux, S. Seidel, M. Dumas, A.H.M. ter Hofstede, Questionnaire-driven configuration of reference process models, in: Proc. CAiSE'07, 2007.

[72] M. Rosemann, Potential pitfalls of process modeling: Part A, Business Process Management Journal 12 (2) (2006) 249–254.

[73] M. Rosemann, J. Recker, C. Flender, Contextualisation of business processes, International Journal of Business Process Integration and Management 3 (1) (2008) 47–60.

[74] M. Rosemann, W.M.P. van der Aalst, A configurable reference modelling language, Information Systems 32 (1) (2007) 1–23.

[75] A. Rozinat, W.M.P. van der Aalst, Conformance testing: measuring the fit and appropriateness of event logs and process models, in: BPM'05 Workshop, 2006.

[76] A. Schnieders, F. Puhlmann, Variability mechanisms in e-business process families, in: Proc. BIS'06, 2006.

[77] A. Sharp, P. McDermott, Workflow Modeling: Tools for Process Improvement and Application Development, Artech House, 2001.

[78] B. Silver, BPMS watch: Ten tips for effective process modeling, http://www.bpminstitute.org/articles/article/bpms-watch-ten-tips-for-effective-process-modeling.html (2009).

[79] M. Soto, A. Ocampo, J. Munch, The secret life of a process description: a look into the evolution of a large process model, in: Proc. ICSP'08, 2008.

[80] D. Strong, S. Miller, Exceptions and exception handling in computerized information processes, ACM ToIS 13 (2) (1995) 206–233.

[81] G. Sunye, D. Pollet, Y.L. Traon, J. Jezequel, Refactoring UML models, in: Proc. UML'01, 2001.

[82] N. Trcka, W.M.P. van der Aalst, N. Sidorova, Data-flow anti-patterns: discovering dataflow errors in workflows, in: Proc. CAiSE'09, 2009.

[83] W.M.P. van der Aalst, Business alignment: Using process mining as a tool for delta analysis and conformance testing, Requirements Engineering Journal 10 (3) (2005) 198–211.

[84] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A. Barros, Workflow Patterns, Distributed and Parallel Databases 14 (1) (2003) 5–51.

[85] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, A. Weijters, Workflow mining: a survey of issues and approaches, Data and Knowledge Engineering 27 (2) (2003) 237–267.

[86] W.M.P. van der Aalst, M. Dumas, F. Gottschalk, A.H.M. ter Hofstede, M.L. Rosa, J. Mendling, Correctness-preserving configuration of business process models, in: Proc. FASE 2008, 2008.

[87] W.M.P. van der Aalst, M. Dumas, F. Gottschalk, A.H.M. ter Hofstede, M.L. Rosa, J. Mendling, Preserving correctness during business process model configuration, Formal Aspects of Computing 22 (3–4) (2010) 459–482.

[88] W.M.P. van der Aalst, K. Lassen, Translating unstructured workflow processes to readable BPEL: theory and implementation, Information and Software Technology 50 (3) (2008) 131–159.

[89] B.F. van Dongen, R. Dijkman, J. Mendling, Measuring similarity between business process models, in: Proc. CAISE'08, 2008.

[90] I. Vanderfeesten, J. Cardoso, J. Mendling, H. A. Reijers, W.M.P. van der Aalst, Quality metrics for business process models, BPM & Workflow Handbook, 2007.

[91] I. Vanderfeesten, H.A. Reijers, J. Mendling, W.M.P. van der Aalst, J. Cardoso, On a quest for good process models: the cross-connectivity metric, in: Proc. CAiSE'08, 2008.

[92] I. Vanderfeesten, H.A. Reijers, W.M.P. van der Aalst, Evaluating workflow process designs using cohesion and coupling metrics, Computers in Industry 59 (5) (2008) 420–437.

[93] J. Vanhatalo, H. Voelzer, J. Koehler, The refined process structure tree, Data and Knowledge Engineering 69 (8) (2009) 793–818.

[94] C. Verdouw, A. Beulens, J. Trienekens, T. Verwaart, Towards dynamic reference information models: Readiness for ICT mass customisation, Computers in Industry 61 (9) (2010) 833–844.

[95] B. Weber, M. Reichert, S. Rinderle-Ma, Change patterns and change support features—enhancing flexibility in process-aware information systems, Data and Knoweldge Engineering 66 (2008) 438–466.

[96] B. Weber, M. Reichert, W. Wild, S. Rinderle-Ma, Providing integrated life cycle support in process-aware information systems, International Journal of Cooperative Information Systems (IJCIS) 18 (1) (2009) 115–165.

[97] B. Weber, S.W. Sadiq, M. Reichert, Beyond rigidity—dynamic process lifecycle support, Computer Science – Research and Development 23 (2) (2009) 47–65.

[98] M. Weidlich, J. Mendling, M. Weske, Efficient consistency measurement based on behavioural profiles of process models, IEEE Transactions on Software Engineering.

[99] M. Weske, Business Process Management: Concepts, Methods, Technology, Springer, 2007.

[100] W. Zhao, R. Hauser, K. Bhattacharya, B. Bryant, F. Cao, Compiling business processes: untangling unstructured loops in irreducible flow graphs, International Journal of Web and Grid Services 2 (1) (2006) 68–91.

Barbara Weber is assistant professor at the University of Innsbruck (Austria). Barbara is a member of the Quality Engineering (QE) Research Group and head of the Research Cluster on Business Processes and Workflows at QE. Barbara holds a Habilitation degree in Computer Science and Ph.D. in Economics from the University of Innsbruck. Her main research interests are agile and flexible processes, integrated process lifecycle support, intelligent user support in flexible systems and process modeling. Barbara has published more than 60 refereed papers, for example, in Data & Knowledge Engineering, Science of Computer Programming and Journal of Software Maintenance and Evolution. Moreover, Barbara is organizer of the successful BPI workshop series.

Manfred Reichert holds a PhD in Computer Science and a Diploma in Mathematics. Since January 2008 he has been appointed as full professor at the University of Ulm. Before he was working as associate professor at the University of Twente (UT) in the Netherlands. At UT he was also leader of the strategic research orientations on "E-health" and on "Applied Science of Services", and member of the Management Board of the Centre for Telematics and Information Technology (CTIT). His major research interests are next generation process management technology (e.g., adaptive processes, process lifecycle management, data-driven processes, mobile processes), service-oriented architectures (e.g., service interoperability, service evolution), and advanced applications for ICT solutions (e.g., e-health, automotive engineering). Together with Peter Dadam he pioneered the work on the ADEPT process management system. Manfred has been participating in numerous research projects in the BPM area and contributed numerous papers. Further, he has co-organized international and national conferences and workshops. Manfred was PC-Co-Chair of the BPM'08 Conference in Milan and General Co-Chair of the BPM'09 Conference in Ulm.

Jan Mendling is a junior-professor with the Institute of Information Systems at Humboldt-University of Berlin, Germany. His research areas include business process management, conceptual modelling and enterprise systems. He has published more than 100 research papers and articles, among others in IEEE Transactions on Systems, Man and Cybernetics, IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering and Methodology, Information Systems, Data & Knowledge Engineering, Formal Aspects of Computing, and Information and Software Technology. He is a member of the editorial board of three international journals. He is program co-chair of the International Conference on Business Process Management 2010 and initiator of the BPMN workshop series.

Hajo Reijers is an associate professor in the Information Systems Group at Eindhoven University of Technology and an affiliated professor with the TIAS/Nimbas Business School of Tilburg University. Before that, he worked as a management consultant for Accenture and Deloitte. His research interests cover business process redesign, business process modeling, workflow management technology, and simulation. He has published over 75 refereed papers, for example in the Journal of Management Information Systems, Information systems, Data & Knowledge Engineering, and Organization Studies. He recently served as the co-chair of the 2009 edition of the International Conference on Business Process Management.