Variability Patterns for Analyzing Flexible Processes

Kathrin Kirchner[1(✉)] and Ralf Laue[2]

[1] Berlin School of Economics and Law, Alt-Friedrichsfelde 60, 10315 Berlin, Germany
kathrin.kirchner@hwr-berlin.de
[2] Department of Computer Science, University of Applied Sciences of Zwickau, Dr.-Friedrichs-Ring 2a, 08056 Zwickau, Germany
Ralf.Laue@fh-zwickau.de

Abstract. In practice, flexible processes often do not follow a fixed sequence of steps. Instead, process participants can rely on their knowledge and can decide for additional steps, change the execution order or skip a task. If a requirements engineer discusses such a process together with the stakeholders, variability has to be taken into account. The challenge is to translate the stakeholders' explanations into a process model by asking the right questions. In order to cope with this problem, we propose the use of patterns. Patterns are a well-known way in systems engineering to describe solutions for frequently occurring problems. In this paper, we present the idea of business process variability patterns and give examples how they can be used in requirements and process analysis.

Keywords: Flexible process · Process modeling · Process analysis Pattern

1 Introduction

In real-world-scenarios, processes can be highly flexible. Process participants as domain experts decide on the execution of activities as well as their order during process execution. Therefore, it is not possible to fully prescribe such a process beforehand as it can be done for, e.g., a standardized purchasing process. A typical scenario for variability is a medical treatment process. Physicians decide dynamically which treatment is needed depending on the health status of the patient. A certain order of steps is known in advance, e.g., that some preliminary examinations have to be carried out before an operation. The sequence of these examinations might change due to several criteria. In order to document such processes, to optimize and to support them by information technology, the following question arises: How we can support the dialog between requirements engineer and domain experts as effectively as possible?

We propose the use of patterns that deal with typical variable scenarios. The aim of our patterns is to speak about variable parts of a process (model) using a vocabulary that is close to domain experts' understanding instead of speaking in terms such as "gateways" or "timer events" - which would be the

E. Teniente and M. Weidlich (Eds.): BPM 2017 Workshops, LNBIP 308, pp. 623–629, 2018.
https://doi.org/10.1007/978-3-319-74030-0_49

vocabulary of a modeling language (in this case BPMN). The patterns lead to typical questions and suggestions for optimization which can be helpful for a requirements engineer. While the abstraction level of our patterns is higher than it is the case for the workflow patterns [1], it is still lower than it is the case for patterns that address the actual meaning of an activity [2].

In this paper, we draw on experiences from the research project PIGE (Process Intelligence in Healthcare) at the University Hospital Jena (Germany) [3]. We analyzed 47 process models, namely in the area of liver transplantation, living liver donor transplantation, liver cell cancer and hepatocellular carcinoma that were modeled together with medical experts. The examples in Sect. 2.2 are taken from this project.

2 Business Process Variability Patterns

2.1 Selected Related Work

In business process modeling, the most popular work on patterns are workflow patterns [4]. These patterns have been widely used by practitioners, vendors and academics. They are helpful for selecting a workflow system that corresponds to the needs of an organization and for evaluating the expressiveness of modeling languages. Workflow patterns provide an answer to the question which elements a workflow engine or modeling language should support in order to be useful in a given context. However, they do not deal with the topic of process variability.

To address process variability, the patterns need to have a somewhat higher abstraction level than the traditional workflow patterns [4]. For example, we wish to say that certain activities are executed in parallel instead of referring to the pattern Parallel Split in combination with another pattern Synchronization as it would be the case in the terminology of workflow patterns. Also, the patterns should be named in a language that can be used intuitively by domain experts. For example instead of speaking about the interleaved routing pattern, we use the term in any order.

Process variability can be expressed by means of patterns for differences/changes between business process models. Such patterns have been discussed for various purposes and using different terminologies (an example is [1]). However, such change patterns have been developed for other purposes than the patterns in our paper. They describe variability on a rather technical level while the aim of our patterns is to use typical business vocabulary only. For example, our pattern Perform in Any Order corresponds to the change pattern Move Process Fragment in [1]. While our pattern describes a business situation, the change pattern deals with abstract operations on a model. The exception handling patterns in [5] are closer to our work (and were used as an important source for identifying our patterns). However, the focus of [5] is restricted to exception handling which is just one aspect of variability.

2.2 Variability Patterns

We collected typical variable situations in a process that are described by the stakeholders verbally. Based on a literature review and our modeling experience, we found three main classes for variability in business processes: First, the number of executions of an activity can be unknown until execution time. Activities may be optional, i.e. they can be skipped or it can be necessary to repeat an activity until the desired outcome has been achieved. Second, it can turn out that additional activities have to be added to the process. Third, the order between activities might not be known before the process is actually executed.

Using patterns allows a requirements engineer to communicate with stakeholders in a language that is very close to their domain vocabulary. In addition, each pattern leads to typical questions and optimization options. For illustrating the principle, we selected three patterns related to a situation where a series of tests must be passed. Such situations are common in medical processes, but also in other domains (e.g., approval procedures). Depending on the context, the most appropriate pattern has to be found. For its selection, a requirements engineer can already make use of a list of typical questions:

- Which activities lead to the situation where the object has to be tested?
- In which order the tests are usually performed? Why? Which resources are used? Are there dependencies between the tests that restrict the order?
- Which tests can be performed in parallel?
- What are the non-functional requirements for the tests? How important are speed, cost and accuracy?
- Is it always necessary to perform all tests? Are there cases where only the "most important" tests are performed (e.g., to save time)?
- What should be done in case of a failure of a test? (Patterns for possible consequences are discussed in [6].)
- Have any preparatory steps (such as collecting or organizing data) to be done before the actual tests can be carried out? Can data collected in advance become invalid before actually being used in a test?
- What are the percentage of failure and the probability distributions of cost and execution time for each individual test? How we can get access to logs of previously executed tests to answer these questions?
- Should we monitor the parameters described in the previous question at the runtime of future process instances? How should this information be used?

We can associate questions with certain patterns, so that one of the patterns described later can be selected depending on the questions' answers. Till now, we found a number of ten flexibility patterns, but due to space limitations, we present only three patterns here in more detail.

Pattern "Pass All Tests (Ordered)" *Problem:* Some object has to undergo a series of tests. The order in which the tests should be executed is known beforehand. The process can proceed only if all tests have been passed successfully.

As soon as the first test fails, another path in the process flow is taken, and no further tests are necessary.

Example: A prospective liver donor needs to undergo a blood test to investigate the compatibility to the patient's blood. In case the blood is not compatible, the donor is not suitable, and no further tests are necessary. Otherwise, an investigation by a psychologist is planned. Here, too, the donor can be found to be not suitable if there are any contraindications.

Questions: What are the reasons for defining the order of the individual tests? Which tests depend on the output of other tests?

Considerations for Optimization: If no dependencies between the tests exist, the issue of finding the optimal order between the tests is discussed in detail for the next pattern "Pass All Tests (Any Order)". Once the order is fixed, we can ask whether it corresponds to the spatial and organizational layout of the organization in order to avoid long ways for transport and unnecessary handovers.

If preparatory steps (such as collecting data) are executed before the first test starts, we can ask whether indeed all those steps are necessary *before* the sequence of tests. As it can be possible that the preparation for test 2 becomes irrelevant if already test 1 fails, it can be a good decision to prepare for test 2 only after test 1 succeeded.

Pattern "Pass All Tests (Any Order)" *Problem:* Some object has to undergo a series of tests. The order in which the tests should be executed has to be decided at run time. The process can proceed only if all tests have been passed successfully.

Example: In order to decide whether a person is a suitable liver donor for a near relative, a number of investigations and tests are necessary according to an evaluation check list. A transplant operation can only be planned if the whole evaluation procedure is completed successfully.

Questions: The tests can be performed in any order – but is there a typical/preferred order anyway? Why?

Considerations for Optimization: If for each test, the cost for executing the test is known as well as the probability that the test succeeds (i.e. delivers a "positive" result), the question for the optimal ordering of the tests is the well-known filter ordering problem (in the area of business process management also known as optimal ordering of knock-out processes [7]). The tests should be executed in increasing order of the ration between cost of a test and the probability that the test reveals a problem [8]. "Cost" can be replaced by another indicator which should be minimized (e.g. the radiation exposure from medical imaging). If the probabilities that a test succeeds are not known beforehand, they can be learned at runtime from previously executed process instances. If the tests are executed by different roles, it can also be helpful to take into account organizational handovers and transport ways between the performers of the tests. Both should be minimized.

Pattern "Pass All Tests (Parallel)" *Problem:* Some object has to undergo a series of tests. These tests can be done in parallel (to be executed by different actors).

Example: While the prospective liver donor is undergoing medical investigations, medical consultations (meetings among physicians from different disciplines) can take place to discuss the case and discuss possible contraindications.

Questions: What advantages do we expect from performing the tests in parallel?

Are there really no dependencies between the tests such that one test would need (or at least profit from) the results of another test?

How do we ensure that all tests have access to the same up-to-date data of the object under test?

If one of the tests fails, is it desirable to prevent (or even to stop) the execution of the remaining tests?

Considerations for Optimization: In general, this pattern should be chosen if execution time is more important than costs for potentially unnecessary tests. The disadvantage for this pattern is that it can happen that some tests are executed unnecessarily, because another test already reveals an error. Therefore, the requirements engineer should discuss whether/how the failure of a test should be communicated such that the roles performing the other tests do not have to do them anymore (see the discussion of the Cancel Activity pattern in [4]).

2.3 Combining Patterns

When patterns are combined, more questions can emerge. We will illustrate this for the combination of any of the Pass All Tests patterns with the pattern Repeat Activity Until Success: Some activity A must be executed repeatedly until several tests which follow its execution confirm that A has been executed correctly. A single failed test shows that A has to be repeated. Afterward, a new round of tests is necessary until all of them succeed (proving that A was carried out successfully). There are two possible ways to deal with this situation: Either all tests have to be executed again or only the tests that did not already succeed before. To decide between those options, we have to ask whether the rework can introduce new problems concerning features that have already passed a test.

In addition, there might be a time limit for executing A and the corresponding tests. This leads to a third pattern Must Succeed in a Given Time. The requirements engineer would have to ask what happens in the case of a timeout (a typical option would be that a person in a higher position in the organization becomes responsible for achieving the result of A). In case of time-consuming tests, it can also be an option only to perform a subset of the tests in order to exclude the most important risks only.

3 Pattern Application

For the process of a living liver donor in the PIGE project, medical experts were involved in the modeling and discussion of the as-is-process. Some parts of the process are flexible, e.g., the evaluation of possible donors mentioned before. Several examinations have to be carried out before a decision can be made whether the examined person can donate his/her liver. During the modeling of the process, the physicians showed a paper checklist containing all examinations that the possible donor has to undergo. Only if the person's results are positive in every examination, she/he is a suitable candidate for a donation. This calls for our pattern Pass all tests (in any order). According to this pattern, the modeling expert should ask whether a preferred sequence exists that is typically used. This way, additional conditions can be defined.

In the detailed discussion of the evaluation process, it turned out that after the blood compatibility and psychological testing, a number of further investigations and tests are necessary according to the check list. Their sequence is decided by the medical personal. If a mayor contraindication is found, the donor is not suitable, and no further tests are necessary.

Additionally, data was collected from the clinical information systems and analyzed alongside the evaluation procedure. It turned out that especially one examination often leads to a dropout of the possible donor. With this knowledge, the process could be optimized following the optimization suggestions from the pattern (see Sect. 2.2). The examination procedure was changed so that the examination with the high dropout rate is done at the beginning of the evaluation process. This way, the evaluation procedure for non-suitable donors is shorter, and hospital stay for further examinations can be avoided [9].

In the future, we plan to extend our set of variability patterns to cover more variable situations in business processes in other application areas. Additionally, we will work on the representation of our patterns in different modeling languages.

References

1. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data Knowl. Eng. **66**(3), 438–466 (2008)
2. Thom, L.H., Reichert, M., Iochpe, C.: Activity patterns in process-aware information systems: basic concepts and empirical evidence. Int. J. Bus. Process Integr. Manage. **4**(2), 93–110 (2009)
3. Kirchner, K., Malessa, C., Scheuerlein, H., Settmacher, U.: Experience from collaborative modeling of clinical pathways. In: Modellierung im Gesundheitswesen: Tagungsband des Workshops im Rahmen der Modellierung 2014, pp. 13–24 (2014)
4. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.: Workflow patterns. Distrib. Parallel Datab. **14**(3), 5–51 (2003)
5. Staudt Lerner, B., Christov, S., Osterweil, L., Bendraou, R., Kannengiesser, U., Wise, A.: Exception handling patterns for process modeling. IEEE Trans. Softw. Eng. **36**(2), 162–183 (2010)

6. Namiri, K., Stojanovic, N.: Pattern-based design and validation of business process compliance. In: Meersman, R., Tari, Z. (eds.) OTM 2007. LNCS, vol. 4803, pp. 59–76. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76848-7_6
7. van der Aalst, W.M.P.: Re-engineering knock-out processes. Decis. Support Syst. **30**(4), 451–468 (2001)
8. Garey, M.: Optimal task sequencing with precedence constraints. Discrete Math **37**(4), 37–56 (1973)
9. Kirchner, K., Scheuerlein, H., Malessa, C., Krumnow, S., Herzberg, N., Krohn, K., Specht, M., Settmacher, U.: Was ein klinischer Pfad im Krankenhaus bringt. Evaluation klinischer Pfade am Uniklinikum Jena am Beispiel des PIGE-Projekts, Chirurgische Allgemeine Zeitung, **15**(7+8), pp. 475–478 (2014)