universität wien

DISSERTATION / DOCTORAL THESIS

Titel der Dissertation / Title of the Doctoral Thesis

Representation and Analysis of Change Operations for Process Instances

verfasst von / submitted by

Dipl.-Ing. Georg Kaes, B.Sc.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der Technischen Wissenschaften (Dr. techn.)

Wien, 2018 / Vienna 2018

Studienkennzahl laut Studienblatt / degree programme code as it appears on the student record sheet:	A 786 880
Dissertationsgebiet laut Studienblatt / field of study as it appears on the student record sheet:	Informatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Math. oec. Dr. Stefanie Rinderle-Ma

Abstract

Supporting flexibility in business processes is vital to almost any business area: In nursing homes and in the medical sector, domain experts such as nurses and doctors should be able to adapt the therapy processes of their patients, due to, e.g., a change in the patient's medical condition. In manufacturing environments, process instances need to be changed if a supplier cannot deliver the required raw materials, or some machine is out of order. In general, process change operations facilitate the adaptation of process instances in order to fit the requirements of the current situation. In many cases, such process changes have to be planned, analyzed, and executed manually by domain experts from the respective field of work. When planning necessary changes, or analyzing the effects of past changes, historic information about previously conducted change operations may be required, which is available in process log files. However, the data contained in such process log files often proves to be problematic for domain experts who have no experience with processes at a technical level: The information is often encoded in machine readable representations, or in representations which only allow for a limited set of analysis questions. Additionally, the data is often not tailored to the specific analysis question of the domain expert; hence, the vast amount of information present in the process log can be overwhelming. Hence, providing representation and analysis methods for change logs tailored towards usage by domain experts is crucial. Process change operations affect process instances at a variety of different levels. One major challenge lies in highlighting the information which is important for the current analysis question. Additionally, change logs potentially contain a vast amount of change operations for thousands of process instances. Especially for scenarios, where a domain expert has to analyze and decide quickly for a change operation, representations need to allow to focus on the data of interest. So far, only few appproaches address user support in process change analysis. Hence, in this thesis, we present representation and analysis methods for single process change operations, as well as sequences of change operations, and their application. One cornerstone is provided by the concept of change trees, which enable domain experts to analyze, among others, the diversity of the applied process change operations in a change log. By using process change similarity measures for comparing

single change operations, a more detailed analysis of the diversity of the applied process change operations is possible. The approaches presented in this thesis are evaluated at three different levels: First, a prototypical implementation shows the technical feasibility of the concepts developed. Second, we analyze how our approaches could be used in different domains, including the nursing, wellness, educational and business sectors, thus analyzing the applicability of this thesis. Third, we evaluate our approaches with user-based experiments, thus evaluating the usability of the presented representation and analysis techniques.

Overall, this thesis presents concepts for bridging the gap between change information presented on a technical level at the system side to adequat representations and analysis methods for domain experts at the user side.

Zusammenfassung

Flexible Geschäftsprozesse sind heutzutage aus beinahe allen Branchen nicht mehr wegzudenken: In Pflegeheimen oder dem Medizinbereich beispielsweise, sollen Domänenxperten wie Pfleger oder Ärzte die Therapiepläne ihrer Patienten flexibel an die sich ständig ändernden Gegebenheiten anpassen können: Wann immer sich der Gesundheitszustand des Patienten verschlechtert, muss der entsprechende Therapieplan flexibel angepasst werden können. Ein weiteres Beispiel stammt aus der Produktionsbranche: Hier müssen Produktionsprozesse flexibel an die sich ständig ändernden Gegebenheiten angepasst werden können: Wenn beispielsweise eine Maschine ausfällt, oder ein Lieferant die notwendigen Ressourcen nicht rechtzeitig liefern kann, muss der verantwortliche Manager die betroffenen Prozessinstanzen entsprechend adaptieren können. In vielen Fällen müssen solche Prozessänderungen von Domänenexperten händisch analysiert, geplant, und durchgeführt werden. Basierend auf historischen Daten, welche in Prozessänderungslogs zur Verfügung stehen, kann der Experte sehen, welche Änderungen in der Vergangenheit durchgeführt worden sind. Allerdings kann die Art und Weise, wie solche historische Daten aufbereitet werden, für Domänenexperten ohne technisches Grundwissen problematisch sein: Die Logs sind oft in einer maschinell lesbaren Repräsentation verfügbar; weitere existierende Repräsentationen eignen sich oft nur für bestimmte Analysen. Des Weiteren sind die Daten oft nicht auf die Fragestellung des jeweiligen Domänenexperten zurechtgeschnitten; folglich kann sich eben dieser durch den Überfluss an für die aktuelle Fragestellung irrelevanten Daten überfordert fühlen. Aus diesen Umständen folgt die essentielle Notwendigkeit, Repräsentationsformen und Analysemethoden für Prozessänderungslogs zur Verfügung zu stellen, welche von Domänenexperten eingesetzt werden können. Prozessänderungen beeinflussen Prozessinstanzen auf verschiedensten Ebenen. Dieser Umstand führt dazu, dass Analyse- und Repräsentationsformen, die zum Ziel haben, Domänenexperten bei der Analyse von Prozesslogs zu unterstützen, die jeweils relevanten Informationen entsprechend hervorheben müssen. Als weitere Erschwernis kommt hinzu, dass Prozesslogs oft tausende Änderungsoperationen beinhalten können, welche von vielen verschiedenen Instanzen stammen. Gerade in Situationen, in denen der Domänenexperte schnell zu einem Ergebnis

kommen muss, liegt eine besondere Herausforderung darin, Repräsentationen zu schaffen, die einen Fokus auf die Daten, die gerade von Interesse sind, erlauben. Aktuell existieren nur wenige Ansätze, die das Ziel der Userunterstützung während der Analyse von Prozessänderungen verfolgen. Aus diesem Grund präsentieren wir in dieser Dissertation verschiedene Repräsentations- und Analysewerkzeuge sowohl für einzelne Prozessänderungen, wie auch für Sequenzen von Änderungen, Prozesslogs, und deren Anwendungen. Ein Grundpfeiler liegt im Konzept sogenannter *Change Trees*, welche es Domänenexperten unter anderem erlauben, die den im Prozessänderungslog enthaltenen Änderungsoperationen zugrundeliegende Diversität zu analysieren. Weiters können Experten durch das Konzept der Ähnlichkeitsmetriken für Prozessänderungen diese Diversität für einzelne Prozessänderungen noch detaillierter, und aus verschiedenen Blickwinkeln analysieren.

Die hier präsentierten Ansätze werden auf drei verschiedene Arten evaluiert: Eine prototypische Implementierung der verschiedenen Repräsentations- und Analysetechniken zeigt die technische Machbarkeit. Im Rahmen einer Anwendungsanalyse überprüfen wir, inwiefern sich die entwickelten Ansätze in verschiedenste Domänen integrieren lassen. Außerdem evaluieren wir ausgewählte Ansätze mit Anwendern, um deren Benutzerfreundlichkeit zu testen.

Zusammenfassend verfolgt diese Dissertation das Ziel, eine Brücke zwischen den im Prozessänderungslog zur Verfügung stehenden Informationen, welche auf einer technischen Ebene präsentiert werden, und den Domänenexperten, welche adäquate Analyse- und Repräsentationswerkzeuge benötigen, zu schlagen.

Acknowledgements

First, I would like to use this opportunity to thank my supervisor, Univ.-Prof. Dipl.-Math. oec. Dr. Stefanie Rinderle-Ma, for her encouragement, and great support during the last years. Also, I would like to thank Prof. Barbara Weber for agreeing on reviewing this thesis.

I would like to thank my colleagues from the research group Workflow Systems and Technology at the University of Vienna, especially Kristof Böhmer, Conrad Indiono, Walid Fdhila, Manuel Gall, Florian Stertz, Karo Winter, Jürgen Mangler, Erich Schikuta, Martin Polaschek, Helmut Wanek, Monika Hofer-Mozelt, and Tanja Schwind for their support and the last years. This research group is a great team, and without their support, great talks and feedback, this thesis would not have become what it is today.

Last but not least, I want to thank my family, who always supported me throughout the last years.

Publications

Parts, concepts and ideas presented in this thesis have appeared in the following publications:

Conference Proceedings and Workshop Papers

Improving the Usability of Process Change Trees based on Change Similarity Measures (Georg Kaes and Stefanie Rinderle-Ma). In: Enterprise, Business-Process and Information Systems Modeling (BPMDS 2018), Springer, pp. 147-162, 2018

Generating Data from Highly Flexible and Individual Process Settings through a Game-based Experimentation Service (Georg Kaes and Stefanie Rinderle-Ma). In: Datenbanksysteme für Business, Technologie und Web (BTW 2017), Gesellschaft für Informatik, pp. 331-350, 2017

On the Similarity of Process Change Operations (Georg Kaes and Stefanie Rinderle-Ma). In: Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CAiSE 2017), Springer, pp. 348-363, 2017

Multidimensional Process Mining: Questions, Requirements, and Limitations (Thomas Vogelgesang, Georg Kaes, Stefanie Rinderle-Ma and Hans-Jürgen Appelrath). In: Proceedings of the CAiSE'16 Forum, at the 28th International Conference on Advanced Information Systems Engineering (CAiSE 2016), CEUR Workshop Proceedings, pp. 169-176, 2016

Mining and Querying Process Change Information Based on Change Trees (Georg Kaes and Stefanie Rinderle-Ma). In: Proceedings of the 13th International Conference on Service-Oriented Computing (ICSOC 2015), Springer, pp. 269-284, 2015

Flexibility Requirements in Real-World Process Scenarios and Prototypical Realization in the Care Domain (Georg Kaes, Stefanie Rinderle-Ma, Ralph Vigne, Jürgen Mangler). In: On the Move to Meaningful Internet Systems (OTM 2014 Workshops), Springer, pp. 55-64, 2014

Demo Papers

ACaPlan - Adaptive Care Planning (Georg Kaes, Jürgen Mangler, Florian Stertz, Ralph Vigne, Stefanie Rinderle-Ma). In: Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015), CEUR Workshop Proceedings, pp. 11-15, 2015

Technical Reports

The NNN Formalization: Review and Development of Guideline Specification in the Care Domain (Georg Kaes, Jürgen Mangler, Stefanie Rinderle-Ma, Ralph Vigne). In: Computing Research Repository (CoRR), arXiv.org, arXiv:1404.2162, 2014

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Business Process Management (BPM) is the art and science of overseeing how work is performed in an organization to ensure consistent outcomes" [1]. According to Gartner, enterprises which use business process management as a method for improving their processes can gain huge cost savings [2]. While business process models contain sets of activities which describe how the work should be performed, process instances contain information related to the execution of these process models. During such executions, the requirements of the process environment may change, thus making the adaptation of process instances necessary. The term *process flexibility* refers to this ability of business processes to adapt to ever changing requirements.

1.1 Motivation

In most business domains, business processes are not static: They need to react to changing requirements, thus adapting to ever changing situations. Flexible business process management systems allow users to change their processes according to the current requirements, thus providing the basis for flexible Process Aware Information Systems (PAIS) [3]. For such process changes, the people who are involved in business process change are an important factor. *"When the people side of change is ignored or poorly managed, the project and the organization take on additional costs and risks. From this perspective, effective change management is a cost avoidance technique and risk mitigation tactic [4]."* Hence, as a basis for a successful and effective process change management, the people involved in adapting process instances need to be able to analyze the changes which have been applied to a certain process instance.

But why is information about previously applied process change operations important? Let us illustrate the need for analyzing process change logs with two

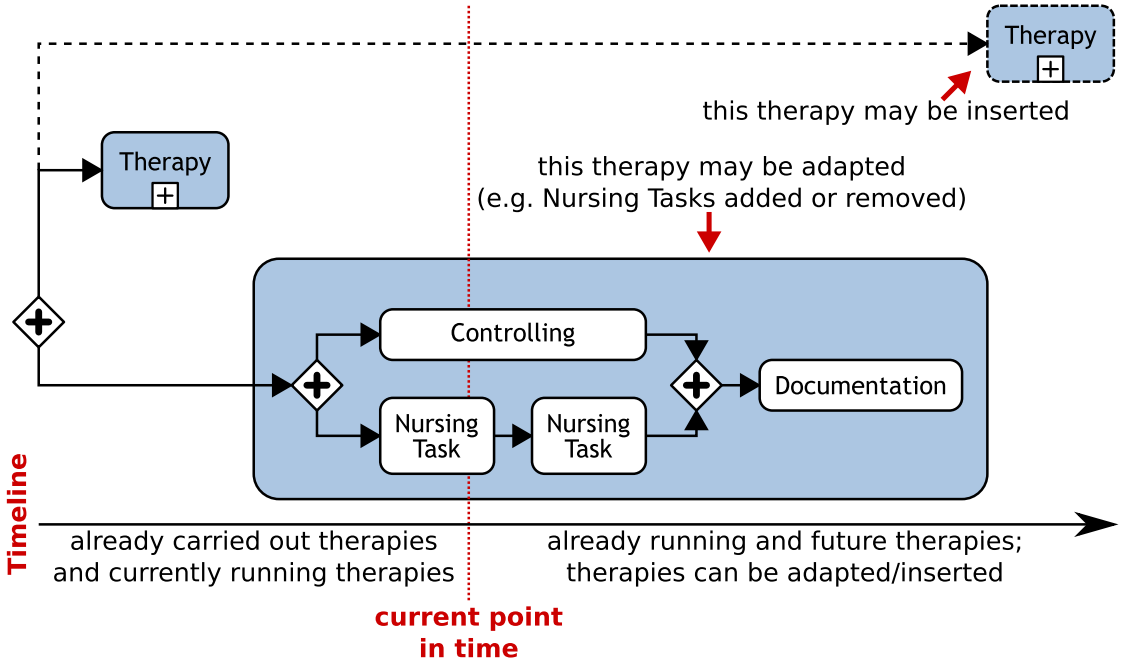


Figure 1.1: Adaptive Process Instance - Example (BPMN notation)

examples stemming from two different domains: Figure 1.1 shows an example of a process instance. It describes the therapy plan of a patient in a nursing home, containing all activities which have to be executed in order to maintain or improve his health. While the process instance has already been executed for a some days or even weeks, possible adaptations may become necessary in the future: If the patient this process instance belongs to catches the flu, some activities of existing therapies may have to be postponed or removed completely, or new activities have to be inserted.

In a nursing environment, the nurses and doctors who interact with a patient need to be aware of the changes made to his individual therapy process instance. Such information is important due to several reasons: Analyzing the history of applied change operations can be relevant, e.g., due to legal reasons: Especially in nursing and medical environments, each activity which has been executed for a patient, including the adaptation of therapy plans, has to be recorded appropriately. Thus, if the health state of a patient deteriorates, the nurses and doctors involved in the treatment can prove that they have planned and executed the corresponding therapy plan to the best of their knowledge. For such analysis, a representation which highlights the chronological order in which change operations have been applied can provide the necessary data. When executing the therapy plan of a patient, the nurses and doctors who are directly involved in this therapy,

can analyze how and why the patient's therapy plan has evolved to its current state: Using such information, they can assess the current state of the process instance, and possibly prevent existing resource bottlenecks by reassigning resources to more important activities. When planning future adaptations, a nurse may want to inspect historic change information to analyze which adaptations have reached the required effects in similar situations in the past: If a change operation which has been applied in a similar situation before is known to have the desired effects on the patient's health, she may use this same change operation again. For example, if a patient with diabetes and a certain set of allergies runs a fever, she can inspect historic information about this and similar patient's therapy processes to see which change operations have achieved the desired results in the past. When comparing change operations which may be applied again, the similarity of those conducted change operations can provide valuable insight: For example, using a similarity metric which compares the required resources, the nurse can analyze how resource assignments will change depending on the choice of the necessary process adaption.

A different example stems from the manufacturing domain: Imagine a manufacturing setting, where a production process describes all activities required for the assembly of a certain product. In such a setting, a production manager needs to adapt the process instances whenever some problem occurs: If, for example, process activities need to be replaced due to some supplier not being able to deliver the required resources on time, the production manager has to adapt the affected process instances accordingly. Historic information about previously applied changes need to be available when analyzing the production of a product. Historic information can be used, for example, to estimate the overall costs of future production process instances. When analyzing the current situation of running production process instances, historic information can also be used to analyze the effects of previously applied process change operations. For such an analysis, a representation which shows previously conducted process change operations in chronological order could be used to highlight the evolution of the relevant process instances: Thus, the production manager can analyze step by step how the process instance has evolved from the unchanged process schema to its current state. Using such information, a production manager can analyze which change operations had which effects on the affected process instances, thus explaining possible problems such as delays or bottlenecks. Additionally, the production manager can utilize information about the effects of previously applied process change operations when planning future adaptations: If a supplier cannot deliver the required goods on time, the manager can then consult historic information in order to see which changes have been executed previously in similar situations. By comparing possible change operations from various perspectives, he can then gain additional

insight into which change operations might be suited best for his current situation: If, for example, the problem has to be solved in less than a week, he can analyze all possible change operations, and discard those where historic information has shown that they will probably require more time.

Overall, we argue that an analysis of historic information can be beneficial for analyzing the past, and the present, and for planning the future of a set of process instances for the following reasons:

- *Analyzing the history of process instances:* Analyzing previously conducted process change operations can be important for a number of use cases, e.g., for compliance checks [5]. Such compliance checks can be necessary for various reasons, e.g., due to legal obligations such as a documentation obligation, which exist among others in the medical or nursing domain. Another example stems from contracts between business partners in a collaborative process setting, where changes of one partner might affect the compliance of their contracts with other partners [5]. We argue that for such an analysis, a representation which highlights the chronological order in which change operations have been applied in the past, can provide required information: Using such a representation, domain experts can analyze how a set of process instances has evolved over time, and deduce the reasons for the applied process changes. Thus, they could be used as a basis to deduce for any point in time in the past, whether threats to the process compliance may have emerged due to conducted change operations [6, 7].

- *Analyzing the current state of process instances:* When analyzing the current state of a set of process instances, a representation which highlights the chronological order of applied change operations can provide valuable insight: Such a representation would enable the domain expert to analyze how the process instances have evolved from the unchanged schema up to their current state. We argue that using such information, one can, for example, analyze how resources in a company are used, and deduce the reasons they have been assigned to process activities which have been adapted: By combining a representation which highlights the chronological order between the conducted change operations with appropriate analysis techniques, the domain expert can analyze how and why the process instances have evolved to their current state step by step: Based on such information, problems like resource bottlenecks could be explained.

- *Planning future change operations:* When planning necessary process adaptations, valuable insight can be gained from analyzing historic data: By analyzing which change operations have been applied in similar situations in the past, in combination with their effects, a domain expert could deduce

which adaptation may achieve the desired effects for his current problem [8]. We argue that for such an analysis, being able to compare historic change operations from various perspectives can provide interesting information for the domain expert: When analyzing a set of change operations which have led to the desired effects in the past, *similarity metrics* can help the domain expert to decide which change operations could be appropriate for his current situation. Similarity metrics highlight which change operations are similar regarding a certain process perspective, e.g., the required resources or time [3]. If, for example, time is critical, the domain expert can compare all relevant change operations regarding the temporal perspective, and choose the change operation where the least amount of time is required.

The concepts presented in this thesis are application-independent. They will be illustrated mainly based on the nursing and manufacturing domains. As we show in Chapter 6, the presented concepts can be applied to other settings as well.

But what do all these use cases have in common? In each of these described situations, a domain expert has to interact with information about previously applied process change operations. Hence, relevant information has to be presented in a way which helps him analyze the data from various perspectives.

1.2 Problem Statement and Research Gap

In many of these settings, change operations should not be conducted automatically. Especially in the health sector, when changing therapy plans of a patient in a nursing home or in a hospital, nurses and doctors may be forced by legal requirements to manually adapt process instances to adhere to the current requirements. But also in other domains, it can be necessary that users change process instances manually. If a production manager recognizes that a certain order cannot be completed in time, or needs a higher prioritization, he may need to decide manually for a change.

Data present in process log files can be used as a basis for analyzing previous changes, for evaluating the evolution of a set of process instances up to their current state, and for planning future adaptations. However, this information can be hard to analyze for users who have little or no technical experience with business processes. Change logs, which contain data about previously applied change operations, or execution logs, which contain information about process activities which have been executed, are often encoded in machine readable representations. These representations provide a solid basis for existing analysis techniques, however, are only of limited use to people with no or little technical experience. So far, existing analysis techniques of process change logs only allow for a limited set

of analysis questions: Using existing representation and analysis methods, for a set of process instances it can be hard to analyze the chronological order of applied process change operations, especially when the same change operation occurs multiple times, or in multiple instances.

For example, change processes [9] are a great tool for representing causal dependencies between applied change operations, but can fail at highlighting their chronological order. However, being able to analyze the chronological order of conducted change operations can be a great asset when analyzing historic information: Using a representation which highlights the chronological order, domain expert can easily inspect, analyze and compare the evolution of a set of process instances. This can be of interest for nurses who have to compare which changes have led to the current state of the therapy process instances for a set of patients who experience the same problems, e.g., due to a legally necessary audit, or they can use this information as a support when planning necessary change operations: We argue that information about change operations which have been applied in similar situations in the past, can be used by domain experts such as nurses as a basis for planning necessary adaptations.

Besides, the data encoded in change log files usually describes what has been changed to a certain set of process instances, e.g., it contains all change operations which have been applied to all process instances which are based upon the same schema. This can be problematic if a domain expert who wants to analyze what has been done in a certain situation requires only a certain part of this data: A doctor would see all change operations which have been applied to all therapy plans in a nursing home, while she only wants to see those which have been applied in order to treat a certain problem, e.g., a patient catching the flu. A production manager would see all change operations which have been applied to the production process instances, without being able to focus on those change operations which have been applied because a supplier did not deliver on time. Hence, users can be overwhelmed in deciding on changes to process structures, e.g., exceptions in their daily routine [10]. This *information overload* [11] can make it very hard for users who have to focus on a specific part of the process change log.

The effects of conducted process change operations can be seen when affected process activities are executed. In contemporary literature, the analysis of process execution logs, which store such information about executed activities, has been discussed extensively [9, 12, 13, 14]. However, a link between the conducted process change operations, and their effects from the corresponding process execution log, is still missing. *ProCycle* [8] provides integrated user support for adapting process instances, however, information about the effects of a process change operation is encoded in a description text of the corresponding change operation. In contrast to this approach, linking activity executions with their corresponding process change

operations can provide domain experts with additional insight into the effects of those change operations based on historic execution data. Based on such a representation, the production manager could analyze the effects of conducted adaptations based on historic data, e.g., if the required resources were delivered on time in the manufacturing domain.

Overall, the goal of this thesis is to bridge the gap between the change information which is present in process change logs at a technical level on the system side, and the domain experts who interact with change operations on the user side.

1.3 Research Questions

The goal of this thesis is to provide representation and analysis methods both for single process change operations, as well as change traces and change logs, which bridge the gap between the system and the user side. Overall, three different levels of granularity need to be addressed:

(a) *Single change operations* are the basic building blocks of a process change log. They contain information about one single change event.

(b) *Change traces* contain a sequence of change operations which have been applied to one process instance. For each process instance, there exists exactly one change trace.

(c) *Change logs* consist of a set of change traces, which belong to a set of process instances based on the same process schema.

When developing representation and analysis methods for process change operations, each of these levels of granularity need to be addressed. Based on the identified problem statement and research gap, we define the following research questions for this thesis:

Q1 **How can domain experts compare process change operations, such that a detailed analysis of single process change operations is possible?**

Single process change operations are the basic building blocks of a process change log. The ability to compare two or more process change operations is a relevant research topic for various applications: For example, when a nurse needs to decide which change operation to apply to a patient's therapy process, she may want to analyze the effects of applicable process change operations. Using such information, she can then decide on one of the change

operations where the desired effects have been reached. When analyzing historic data, one can compare conducted change operations, thus looking for change operations which were different compared to the rest of the change log. When discussing the fundamentals for comparing two single change operations, different aspects have to be considered: While the attributes of a change operation can be used as a basis for comparison, the effects of a change operation can also be used to answer interesting analysis questions, such as *Have the same or different resources been required?* or *Has the same time been required or saved when the change operations have been applied?*. When analyzing how change operations can be compared, it has to be explored which aspects are relevant for conducting such a comparison, how such similarity metrics can be used, and when they should be applied.

Q2 **How can we overcome the information overload problem for representing process change logs? Based on which data?**

Change traces contain the change operations which have been applied for a process instance; change logs group several change traces, e.g., a change log contains all change traces for process instances of the same schema. When analyzing process change logs, only parts of the change log may be required for certain analysis questions. In such a case, it can be beneficial to filter and zoom to the required data [11], such that information which is not related to this analysis question cannot overwhelm the domain expert who analyzes the process change log. Here, a first analysis tackles the question which data should be used for filtering and zooming to the relevant data. Besides, it would be interesting to analyze if and how the concept of process change traces can be used as an additional means to transport information more efficiently.

Q3 **Which properties are required for a representation of process change logs, which highlights the evolution of a set of process instances from their original schema to their current state?**

When analyzing the evolution of a set of process instances, the chronological order of applied change operations provide interesting information. By analyzing sequences of applied change operations in their chronological order, domain experts can see which change operations have led to which other change operations, and ultimately to the current state of the affected process instances. While process change traces usually contain change information in their chronological order, we argue that they are not an ideal representation for highlighting the evolution of a set of process instances: For each instance, there exists exactly one trace, which can make it difficult to analyze similarities between different change traces. Hence, a new representation needs

to be developed which highlights the chronological order of applied change operations as concisely as possible.

Q4 **What are possible application scenarios for process change operations? Can the concepts developed in this thesis also be applied to settings where change operations occur frequently? How can users benefit from the concepts developed in this thesis?**

Process change operations are used in different scenarios. Here, an analysis over different fields of business where change operations are used on a regular basis, which focuses on how the concepts developed in this thesis can be applied, would provide first insights into the applicability of the presented representation and analysis methods. While change operations occur in many business domains, some process settings require more flexibility than others. For example, the nursing domain can be seen as such a setting: Here, every time a patient's therapy plan changes, a change operations is applied. We argue that an analysis about the data which is usually present in such a *flexible and individual process settings* would provide first insights into the applicability of the presented techniques to such settings. Additionally, user-based experiments would help analyze how users could interact with the presented techniques.

1.4 Research Methodology

Overall, the goal of this thesis is to bridge the gap between domain experts who analyze process change logs on the user side, and the technical representation of process change logs on the system side. For this task, different representation and analysis methodologies need to be planned, developed, and evaluated properly. Peffers et al. have described a design science methodology for information systems in their paper *"A Design Science Research Methodology for Information Systems"* [15]. This thesis adapts the research approach of the design science methodology: In their paper [15], the authors define six activities which should be conducted to perform design science research in information systems. In the following, we explain how these activities have been implemented in this thesis.

1. **Problem identification and motivation:**

   As a first step, the problem should be identified and properly motivated. Representation and analysis tools for process change operations which bridge the gap between domain experts and their technical implementation are the main objective of this work. While the required information is available in existing representations, these representations only allow for a limited set of

analysis questions, or may be too technical for a domain expert who has no experience with process change operations at a technical level. Additionally, with the current state of the art, the information overload problem can hinder a successful analysis of process change logs.

2. **Define the objectives for a solution:**

   Based on the identified problem, a realizable objective for a given solution needs to be defined [15]. The overarching goal of this thesis is to provide representation and analysis tools for domain experts to answer more and more complex analysis questions by incorporating historic information from process change and execution logs. Based on the shortcomings of existing representation and analysis techniques, we have identified goals both for single process change operations, as well as for change traces and change logs. The ability to compare single change operations provides a feasible objective for single change operations, which would enhance the analyzability of process change traces. Additionally, allowing users to focus on relevant information, and highlighting the chronological order of applied process change operations are important aspects for analyzing change traces and change logs.

3. **Design and development:**

   After identifying the objectives for a solution, the artifacts which provide this solution have to be developed. For the objective of defining and analyzing change operation similarity, different similarity metrics have been developed. These metrics address both the static aspects, i.e., the attributes of the corresponding change operations, as well as their dynamic aspects, i.e., their effects. The change tree is our representation which highlights the chronological order of applied change operations. In combination with other analysis techniques, such as filtering and change operation similarity, powerful analysis techniques emerge, which allow the domain expert to focus on the relevant information.

4. **Demonstration:**

   The developed artifacts should be applied to the problem setting, in order to show that it solves the given problem. We demonstrate the applicability of the designed artifacts in two ways: First, by implementing the representation and analysis techniques, we demonstrate the technical feasibility. Secondly, we show how the artifacts can be applied to different application domains, thus solving the problems discussed in this chapter: We demonstrate, how more and more complex analysis questions can be answered with the data present in the corresponding process log files, thus bridging the gap between

the domain expert on the user side, and the technical representation on the system side. Additionally, we set a special focus on *flexible and individual process settings*, where process change operations are the main driver of the process design.

5. **Evaluation:**

   After demonstrating the use of the artifact, it should be evaluated *how well the artifact supports a solution to the problem"* [15]. In this thesis, this is done by user-based experiments. We have conducted two experiments which show the advantages of analyzing process change operations with the approaches developed in this thesis.

6. **Communication:**

   After evaluating, the artifacts should be communicated, so that researchers and practitioners can use the results as a basis for future work, or apply them in their target domain. The concepts developed in this thesis have been communicated to the scientific community in conference publications as well as one technical report. A full list of the publications can be found at the beginning of this thesis.

1.5 Contributions and Structure

The representation and analysis techniques developed throughout this thesis are presented as outlined in Figure 1.2. We divide them in three distinct layers: In the first, most basic, layer, we discuss process change operations, as those are the basic building blocks of change traces and change logs. Here, we present change similarity metrics, thus allowing domain experts to compare process change operations ($\rightarrow$ Q1). The second layer discusses how representation and analysis techniques can be applied to process change traces and logs, thus addressing the research gap described above. Here, we present filtering and projection techniques for overcoming the information overload problem for process change logs ($\rightarrow$ Q2), and the change tree as a representation which highlights the evolution of a set of process instances ($\rightarrow$ Q3). In the last layer, we discuss the application of the presented approaches to different process settings, and present the evaluation ($\rightarrow$ Q4).

As a first contribution, we discuss how process change operation similarity metrics can be identified. While similarity metrics for process models and instances already exist, comparing single change events is not possible yet. In Chapter 2, we lay the basic for this contribution by first discussing existing methods for analyzing single process change operations. Based on this analysis, we show how process

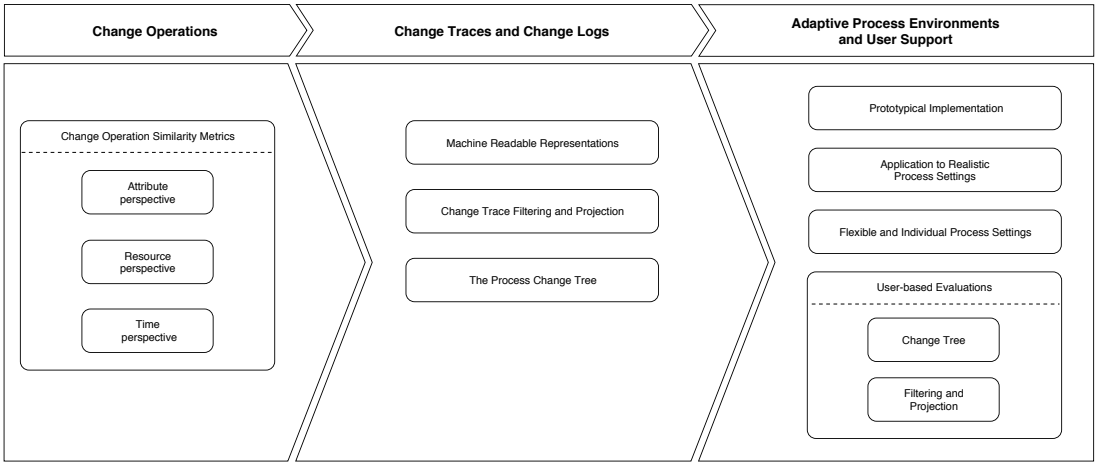


Figure 1.2: Overview over this thesis

change similarity metrics can be used to compare process change operations, and how these concepts can be applied to realistic settings.

Process change traces and logs are built on the basis of process change operations: In Chapter 3, we first analyze existing representation and analysis methods for change logs, and present a new machine readable representation for change logs. This is followed by techniques for filtering and projecting change traces, which can be used to tackle the information overload problem.

Based on the analysis of the shortcomings of existing process mining techniques for change logs, we present the *Change Tree* in Chapter 4. This data structure aims at highlighting the chronological order of applied process change operations. In this chapter, we also demonstrate how the change tree can be extended with additional representation and analysis techniques: We show how to reduce the number of nodes present in a change tree by the application of similarity metrics, and we analyze possible effects of such an application. Additionally, we present how the filtering and projection techniques, which have been presented in Chapter 3 for overcoming the information overload problem, can be applied to change trees.

As a first part of our demonstration of the developed assets, we provide the proof of technical feasibility by means of a prototypical implementation, the APES (Adaptive Process Environment Support) prototype in Chapter 5.

Next, we discuss the application of the developed concepts for different domains. This chapter is split in two parts: First we demonstrate how various log analysis tool developed in this thesis can be applied to realistic data sets from real world environments. In the second part, we describe how applications which aim at user support could be designed for the nursing and wellbeing domain, using the concepts developed in this thesis.

In some process settings, flexibility is a core feature of the process design: In Chapter 7, we analyze *flexible and individual process settings (FIPS)*. After an introduction to this set of settings, we analyze which data items they have in common, and how user support could be designed. Based on this analysis, we present a game-based experimentation service for process adaptations. The goal of this experimentation service is to provide an environment for testing approaches for user support.

Finally, we present two user-based experiments which analyze how the concepts presented in this thesis perform when they are used by users: In our first experiment, we compare how users who analyze change operations using change logs perform in contrast to users who analyze change logs based on change trees. In our second experiment, we show an example for the application of the filtering and projection techniques we have presented for overcoming the information overload problem.

Overall, the goal of this thesis is to provide representation and analysis techniques which allow a successful analysis of process change logs for domain experts.

Chapter 2

Similarity Analysis of Single Process Change Operations

Process Change Operations describe what has been changed in a process instance. Figure 2.1 positions this chapter in the structure of the thesis: Since single change operations are the basic building blocks for all data structures relevant to this thesis such as change traces and change logs, they are discussed first. In this chapter, first, an introduction and a formal definition are given to this topic. This is followed by an analysis of existing analysis techniques for single process change operations. Based on this analysis, we discuss process change similarity as an important analysis method. For this, first an overview over the topic, as well as different notions of the similarity between multiple process change operations are given. This chapter is based upon the Paper *"On the Similarity of Process Change Operations"* [16].

2.1 Overview

Process flexibility is a key concept of business process management [3, 10, 8]. Being able to adapt a process instance whenever the situation requires to do so enables practitioners to flexibly change the design of a process instance to their current requirements. Examples for this can be found in different domains: In the nursing domain, a patient's therapy process instance contains all tasks which need to be done in order to increase or maintain his current health state. Whenever a new problem arises, e.g., he catches the flu, this one patient's process instance needs to be changed, in order to react to the problem accordingly. Depending on which activities currently are present in his process instance, different change operations need to add, remove, or change existing process activities. For example, outdoor activities will be postponed until he feels better; instead, additional drugs may

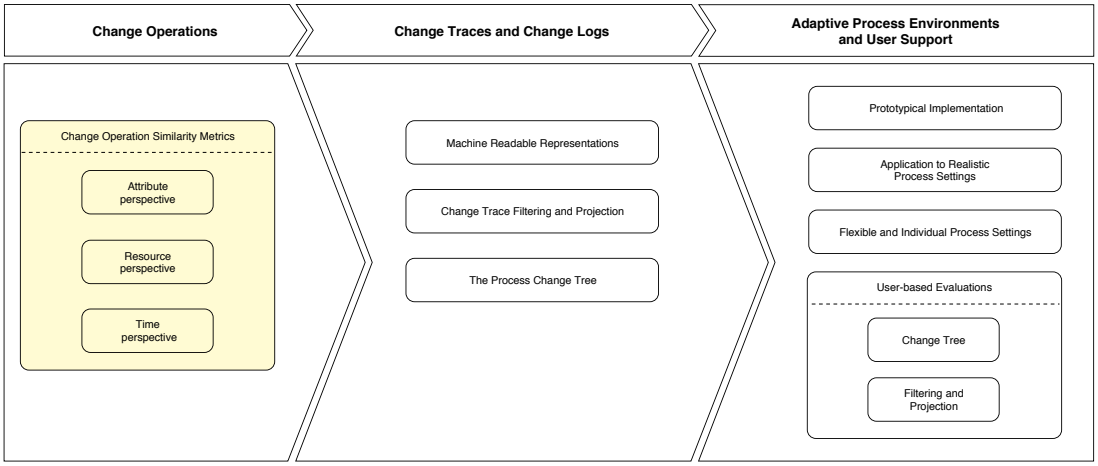


Figure 2.1: Dissertation Overview: Change Operation Similarity

be administered. A different example stems from the manufacturing domain: A manufacturing process contains all tasks required for producing a certain product. When some problems arise, e.g., a supplier does not deliver the required resources on time, the respective production process instance needs to be adapted in order to react to the problem. In this case, activities which deal with the supplier who did not deliver may have to be replaced with other activities, where a new supplier is added to the process instance. Other examples can be found in different domains. An overview over possible application scenarios is discussed in Chapter 6.

Change operations are the key concept when adapting processes. When analyzing single process change operations, it has to be clear which data is available. Based on this data, an overview over existing analysis methods for single process change operations is necessary to provide an overview over which analysis of single change operations already exists in current literature. Based on such an analysis, we have identified that comparing single process change operations has not been tackled yet in existing literature.

Hence, this chapter tackles the following research questions:

Q1: Which data is important for analyzing single process change operations? Which analysis and representation methods for single process change operations already exist? What is still missing?

Q2: How can process change operations be compared? Which perspectives can be of interest?

These research questions are tackled as follows: First, we introduce process change operations from a formal point of view, thus analyzing the data present

when inspecting single process adaptations (→ Q1, Section 2.2.1). Based on this formal definition, we discuss existing state of the art representation and analysis concepts (→ Q1, Section 2.2.2). We show that process change operations similarity metrics are an important new concept for analyzing single process change operations (→ Q2, Section 2.2.3). The remainder of this chapter presents process change operation similarity in detail: Section 2.3 presents different types of change metrics, Section 2.4 presents attribute based similarity metrics, and Section 2.5 focuses on effect based similarity metrics. Finally, in Section 2.6 we analyze the applicability of the presented metrics, and show how they perform compared to existing process model similarity metrics.

2.2 Process Change Operations

This section introduces basic concepts of process change operations, shows existing concepts for analyzing and representing them, and highlights missing concepts.

Process schemata and fragments are two of the basic components of process change operations [3]. A change operation is applied to a process schema. The fragment defines what is being inserted, removed, or in any other way affected by the change operation. Typically, a *process schema (fragment) S "comprises a set of activities with associated application components and with explicitly defined control and data flow between them"* [17], formally:

Definition 2.1 (Process Schema, Process Fragment)**.** A process schema S is defined as
$S := (N, E, D, DE, Res, Temp)$ where

- N denotes a set of nodes, i.e., tasks and gateways such as XOR and parallel splits and joins
- E denotes the set of control flow edges, $E \subseteq N \times N$
- D denotes the set of data elements.
- $DE \subseteq (N \times D) \cup (D \times N)$ is a set of data edges connecting nodes with data elements (write) and data elements with nodes (read).
- $Res : N \mapsto 2^{R \times \mathbb{N}}$ denotes a function that assigns each activity the required number of resources, i.e., $Res(n) = \{(r, x) | r \in R, x \in \mathbb{N}$, r assigned to n$\}$ where R is the set of all resources.
- $Temp : N \mapsto \mathcal{N}$ denotes the point in time associated with a node $n \in N$.

The resource assignment $Res(n)$ determines the number of resources that are required to perform task n, for example, 2 nurses and 1 doctor for a surgery. $Temp(n)$ assigns a point in time to task n. This expression of temporal requirements in processes is simple; more powerful definitions (e.g., [18]) exist. However, for the purpose of the concepts shown in this thesis, time points are assumed as being sufficient.

The top panel of Figure 2.2 shows an example of a process schema from the medical domain: After analyzing the medical history and the symptoms of a patient, a therapy plan is generated by the doctor. This therapy plan is then explained to the patient for him to conduct at home. After the therapy has ended, the patient shows up at the doctor again, and the effects of the therapy are evaluated.

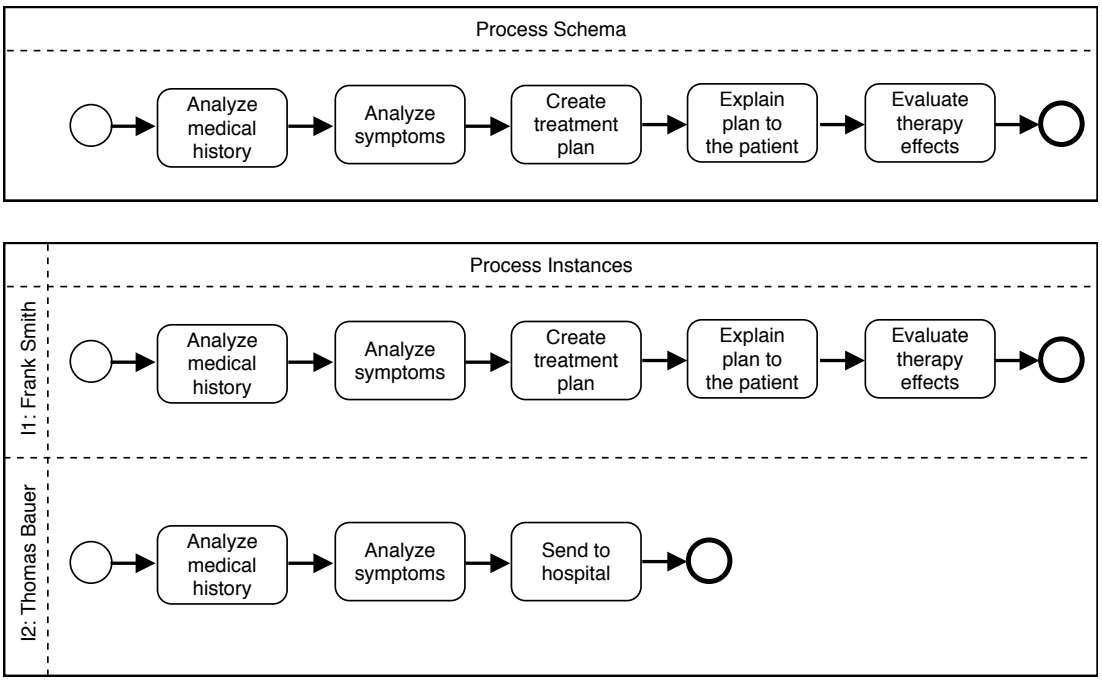


Figure 2.2: Medical example of a process schema and corresponding instances

This process schema defines a generic structure for planning and executing a therapy for patients. When this process schema is used, it first has to be *instantiated*. Creating instances of a process schema means to create executable copies of this schema. Thus, based on a process schema, process instances can be created. The bottom pane of Figure 2.2 shows this concept for the medical domain: The general therapy plan as defined by the schema is now being instantiated for two patients, i.e., one instance is created for patient Frank Smith, the other for Thomas Bauer. Definition 2.2 defines process instances formally.

Definition 2.2 (Process Instance)**.** Let S be a process schema as defined in Definition 2.1. A process instance i is defined as $i := (S)$, where S is the process schema of i.

However, for the example process with its two instances given in Figure 2.2, this therapy plan will not suffice for conducting the necessary therapy for these two patients. The reason for this is the fact that when instantiating the process instance, not all parameters are yet known. For example, when analyzing the medical history and the symptoms, the doctor has discovered a severe problem, thus sending the patient directly to the hospital, instead of creating a therapy to conduct at home.

This example shows an important reason why being able to adapt process instances is a key concept for process management: When a process instance is created, not all required information is known; thus, it has to stay flexible during process execution.

But how can such change operations be executed, i.e., how can we define the changes which have transformed the process schema of instance I2? Process change operations specify adaptations on process schemas or process instances such as insertion or deletion of an activity. Definition 2.3 is based on [9] where the pre and post set of a change are aggregated into the change position p.

Definition 2.3 (Process Change Operation)**.** A process change operation Δ is defined as a tupel $\Delta := (t, f, p, S)$ where

- t denotes the type of the change ($t \in \{\texttt{insert}, \texttt{delete}\}$).
- f denotes the process fragment $f := (N, E, D, DE, Res, Temp)$ which is used by the change operation.
- p denotes the position of the change operation.
- S denotes the process schema $S := (N, E, D, DE, Res, Temp)$ the change operation is applied to.

Information about process change operation can be analyzed to answer several questions: By taking a closer look at the different parameters of a process change operations (cf., Definition 2.3), one can analyze how the control or the data flow of the affected process schema has been affected. Existing concepts for such an analysis are discussed in Section 2.2.1. However, aside from changing the control and data flow of affected process schemata, process change operations can also have different effects. Representation and analysis forms which focus on this aspect are covered in Section 2.2.2. Based on this analysis, we discuss what is still missing.

2.2.1 Change Primitives and Change Patterns

Change operations as defined in Definition 2.3 are also called *high-level change operations* [3]. They consist of multiple *change primitives* in such a way that the correctness of the resulting process can be ensured [19]. Change primitives are four basic operations for changing a processes control flow, i.e.

- `addNode(process,node)`: Adds a node to a process
- `removeNode(process,node)`: Removes a node from a process
- `addEdge(process,node1,node2,edgetype)`: Adds an edge between `node1` and `node2` in a process
- `removeEdge(process,node1,node2,edgetype)`: Removes the edge between *node*1 and *node*2 in a process

In Figure 2.3, the high-level change operation $\Delta = (\texttt{insert}, B, \{A, B\}, S)$ is executed. Translated to change primitives, the following change operations are executed: (1) $addNode(S, B)$, (2) $removeEdge(S, A, C)$, (3) $addEdge(S, A, B, ctrl)$, (4) $addEdge(S, B, C, ctrl)$.

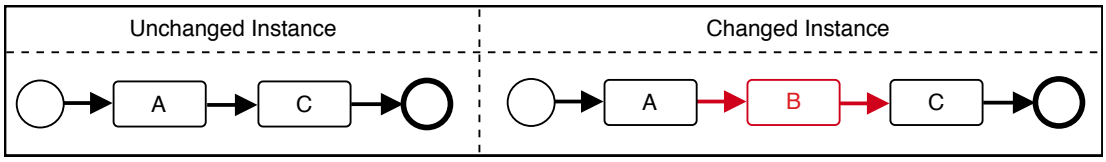


Figure 2.3: A high-level change operation adding activity B to a process instance

A process change operation transforms a *schema* S to a different schema S'. For example, the schema S on the left side of Figure 2.3 is transformed by change operation $\Delta = (\texttt{insert}, B, \{A, C\}, S)$ to S' (containing the activity sequence $\{A, B, C\}$ depicted on the right side).

The basic high-level *change operations types* are `insert` and `delete`. Additionally, other high-level change operations such as `move` or `replace` can be constructed by combining multiple `insert` or `delete` operations into one single operation. For example, `move` is constructed by first deleting a fragment from a certain position, and directly afterwards inserting it again at a different position. Encapsulating high-level change operations this way can increase the semantic expressiveness of a change log, but is not imperative. Thus, this thesis focuses on `insert` and `delete` change operations. The relation between `insert`, `delete` and other high-level change operations is shown in Figures 2.4 and 2.5: In Figure 2.4, activity D is moved from the end of the process to another position after activity A

(pane 1, top). In pane 2 and 3, the same result is achieved by applying `insert` and `delete` change operations consecutively. Similarly, Figure 2.5 shows the swapping of activities A and D: On the top, the `swap` operation is shown; pane 2 to 5 display how the same result can be achieved by using solely `insert` and `delete` change operations.

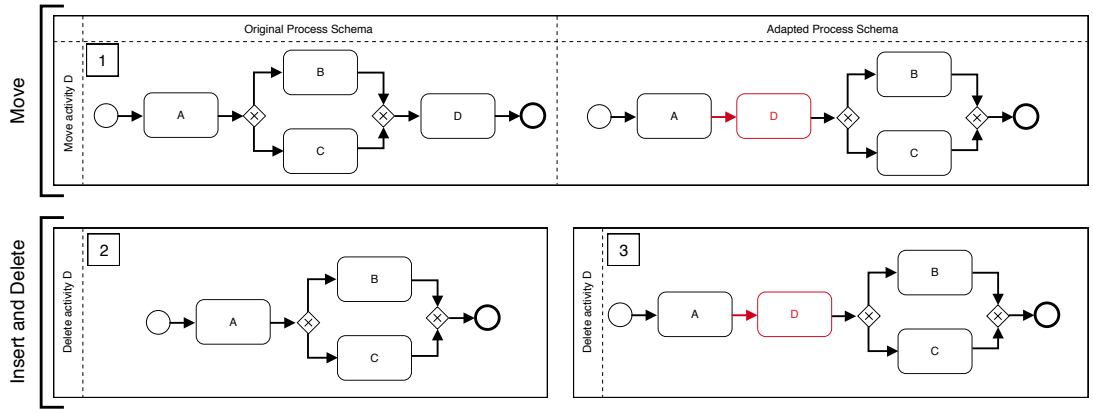


Figure 2.4: A `move` operation can be replaced by consecutive `insert` and `delete` operations

The *position* where a change occurs is defined in a parameter list p [9]. Depending on the type of the change operation, the position is defined as follows: For `insert`, IDs of two consecutive activities define the pre and post set of the change operation. This means that the change subject has to be inserted after the first and before the second activity. For `delete`, no explicit position is defined in current literature [20]. Only the subject to be removed reflects the position of the `delete` operation. However, this implies that each activity can only be found once in a process schema, which is not mandatory for realistic process schemata, as we will discuss in Chapter 6. Thus, in this thesis, both `insert` and `delete` change operations will have a position defined. Following, for `move` and `replace`, the pre and post set of both the source as the target position have to be defined. This stems from the fact that `move` and `replace` are a `delete` operation followed by an `insert`.

Change operation types such as `insert` or `delete` are only two examples of a total of 18 process change patterns [10, 21]. They are grouped in 14 adaptation patterns, including groups such as *adding or deleting fragments*, *moving or replacing fragments*, *adding or removing levels*, *adapting control dependencies* and *change transition conditions*, and four patterns to change predefined regions, i.e., to adapt parts of a process model where it is known in advance, that this part is going to be changed in the future.

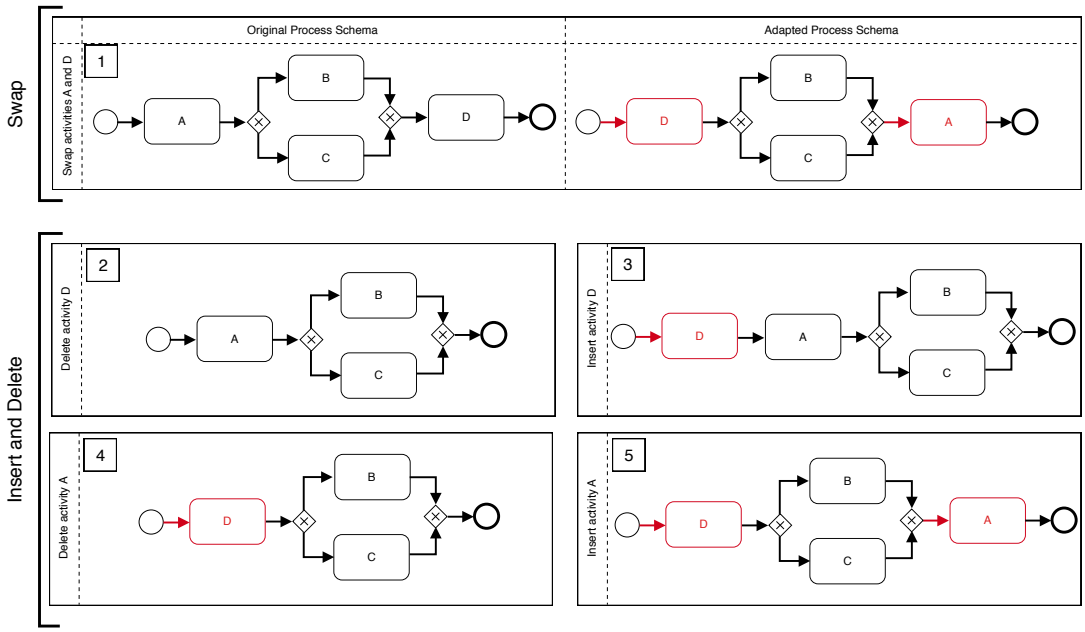


Figure 2.5: A `swap` operation can be replaced by consecutive `insert` and `delete` operations

2.2.2 Analysis of Single Process Change Operations

In the first sections of this chapter, we have analyzed the building blocks a single change operation is built upon. Additionally, we have introduced several types of change operations, and we showed that change operations can be decomposed into a set of change primitives. A single change operation always has some *effect* on a process schema: As discussed above, change operations adapt the process schema, e.g., some activities may be added, moved, or deleted from the affected process schema. Additionally, we have presented several papers which also discuss the formal semantics of single process change operations [10, 21]. However, change operations can also have an effect on process compliance in collaborative settings, which has been discussed in existing literature as well.

Especially in process choreographies, change operations often also affect the processes of partners [22]: Imagine a process by a manufacturer, which interacts with other processes; e.g., from partnering companies or suppliers in a supply chain setting. Whenever the manufacturer's process changes parts of the process which also affects the supply chain, the affected suppliers also have to change their process instances, in order to keep being compliant to the manufacturer's process. Hence, such process change operations do not only have effects on the process instance they adapt, but also on other process instances.

[23] discusses how such changes can be negotiated: Whenever a single change operation is known to not only have an effect on the process instance it adapts, but also on other instances, a fair negotiation about these process changes needs to be triggered, such that all process partners can change their processes accordingly. The basics for analyzing the behavior of such change propagation effects directly from change logs or change propagation logs in process choreographies is discussed in [24]. [25] presents an approach based on the *Refined Process Structure Tree* [26], which propagates changes in a process choreography.

For collaboration of multiple service providers which interact based on business processes, also negotiation of the required resources and services the partners have to offer has to be conducted, which is discussed in [27, 28].

Summarizing, business process change operations potentially do not only have an effect on the process schema they adapt, but can also influence other processes, especially in collaborative settings. In this section, we have presented existing work which tackles problems which arise from such a setting. In the next section, we will show how process change operation similarity can provide additional means to the analysis of single change operations.

2.2.3 Process Change Similarity

Similarity metrics provide an analysis method which analyzes whether and how similar two items are. For example, using similarity metrics for process models [29], one can see whether two models contain the same, or different activities. In the nursing domain, this can be used to compare how similar the therapy plans for two patients are: If a similarity metric returns a high value (thus indicating that the therapy plans are similar), this information can be used as a basis for further analysis; e.g., to compare similarities of the two patient's health problems. Overall, being able to compare change operations would be beneficial in the following situations [8]:

Planning future adaptations: When planning adaptations in certain situations, the person responsible for planning the change could analyze the adaptations which have been made before. Imagine a nursing home, where the therapy plan of each patient is represented as a process instance. Whenever a patient shows a new symptom, his and only his therapy process has to be changed in order to deal with the new problem. If two patients have problems with their digestion, some drugs could be applied for a certain amount of time, their diet plans could be changed, or some other therapies could be applied. Which type of therapy is applied usually depends on various individual circumstances, such as their medical history, pre-existing conditions, allergies etc.

Analyzing past change operations: When evaluating change operations, identifying similar process change operations can add relevant information to an analysis.

In a hospital, it can be analyzed in which situations similar therapies have been applied to a patient's therapy plan. Side effects of change operations can also be compared: Think of a patient who got ill, and received some kind of treatment. Additionally, all therapies which include activities which burden the patient's immune system have to be removed. By analyzing the similarity of such side effects, the evaluation of such situations can be improved.

Both situations benefit from assessing the similarity of the involved change operations based on metrics that measure how much a change operations is similar to another one in a given context. Though literature proposes several metrics for process similarity (e.g., [29]) and instance similarity [30], metrics for change similarity have mainly be neglected so far. The remainder of this chapter approaches this gap based on the following research questions:

SQ1: How can process change similarity metrics be defined in general? Considering which perspectives? Based on which information?

SQ2: In which scenarios can different change similarity metrics be used? What are their advantages and disadvantages?

SQ1 and SQ2 are tackled as follows: First, we analyze the attributes of process change operations and their effects on different process perspectives, i.e., model, data, time, and resource perspective ($\mapsto$ SQ1). The limitations of comparing the similarity of two process change operations solely based on their attributes is discussed in the sequel ($\mapsto$ SQ2). An alternative approach is to exploit the effects of applying change operations to processes. Hence, two change similarity metrics are provided that exploit the effects on the resource and time perspective of the underlying processes ($\mapsto$ SQ1). They can be useful in change scenarios where, for example, the model perspective is not available to users deciding on the change due to privacy reasons. The feasibility and applicability of these metrics is evaluated against metrics that consider the effects of change operations on the model perspective of processes ($\mapsto$ SQ2).

The remainder of this chapter is structured as follows: Section 2.2 introduces change operations as a concept, and gives an overview over the fundamentals of this topic. Based on this, change perspectives, and change effects are introduced (SQ1). This is followed by a discussion about a metric which focuses solely on the attributes of a process change operation (Section 2.4). In Section 2.5, we present effect-based metrics for comparing process change operations (SQ2), which are evaluated in Section 2.6.Section 2.7 concludes this chapter.

2.3 Types of Process Change Metrics

Basically, comparing process change operations can be based on the *attributes* of a change operation Δ, such as its position, fragment, type or schema (cf. Def. 2.3) and its *effects*. The suitability of comparing change operations based on their attributes is discussed by means of an example in Section 2.4.

For comparing change effects, at first, we have to define what change effects actually are. In literature, there are multiple definitions for effects in general, depending on the context. In *Hinge et al.* [31], effects relate to process tasks and are notated in conjunctive normal form (CNF). In *Cossentino et al.* [32], effect logs contain the result of a programs state after steps have been executed. *Rinderle et al.* [19] uses the term *effects* related to change operations to describe the difference between a model S, and an adapted model S' where S' results from applying change Δ to S.

With respect to literature, the definition by *Rinderle et al.* [19] seems most suitable as it is related to change operations. However, the approach focuses mostly on the model and data perspective of the change, even though a schema definition such as in Def. 2.1 comprises additional perspectives such as the resource and time perspectives of process schemas. In order to provide a more comprehensive picture on change effects, a short discussion on change effects in relation to process perspectives seems useful. Hereby we follow [3].

The *model perspective* captures the structural and behavioral perspective of a process. In [21], the formal semantics of change patterns for the control flow of a process are defined. These formal semantics describe the effects of six groups of process adaptation patterns, such as insertion patterns, deletion patterns, or replace patterns (cf. Section 2.2.1).

The *data perspective* describes the data, or information flow of a process model. Related to a process change operation, the data perspective covers the data flow regarding the unchanged process schema, the process fragment, and the resulting process schema. [33] describes several data patterns for the groups *Data Visibility*, which defines in which parts of a process a data element can be accessed, *Data Interaction*, which defines the interaction with data elements in the process, *Data Transfer*, which focuses on the actual data flow, and *Data-based Routing*, which defines the influence of data elements on a processes execution. Analyzing the similarity of the effects of two process change operations on the data flow can yield interesting information, e.g., if some data element which is critical to the processes execution was affected by the process change or not.

The *resource perspective* covers the organizational aspects of a process such as actors, roles, and organizational units. Effects of process change operations on this perspective can be relevant when planning future changes, specifically which and how many resources are affected by the change operation (either because they

have to do more work, less work, or something at a different point in time). In a nursing home, required resources for a long-running therapy can be planned and analyzed before adaptation. If resources are scarce, different adaptations can be compared in order to find the most efficient one. In this chapter, we will describe a metric which focuses on comparing the effects of two process change operations on the resource flow.

The *time perspective* covers all temporal aspects of the process. It is also relevant for change operations, e.g., when the change operation has been conducted (cf. logging changes in [9]) or which due dates are defined for tasks in the change fragment. When comparing the effects of two change operations on a processes time perspective, one can analyze for how long the fragment in a process change will affect the affected schema.

Conclusion: In general both, metrics that are based on attributes or change effects can be used to measure the similarity of change operations. Table 2.1 summarizes what can specifically be compared using either the attributes or the effects of a change operation along the different perspectives.

Table 2.1: Comparing Change Operations based on their Attributes or Effects for Different Perspectives

↓ *Perspective*	**Attribute Based Similarity of**	**Effect Based Similarity of**
Model / Data:	control and data flow of the original schemas and change fragments	the change of control and data flow due to process change operations
Resource:	utilizing resources in the original schemas and change fragments	resource requirement change due to process change operations
Time:	temporal constraints of tasks in the original schemas and change fragments	temporal shifts in the resulting processes due to the process change operations

2.4 Comparing the Attributes of Process Change Operations

This section illustrates that a metric which is solely based on the attributes of a process change operation only yields satisfactory results, if it is tailored to the specific situation (Q2).

When comparing two process change operations $\Delta_1 := (t_1, f_1, p_1, S_1)$ and $\Delta_2 := (t_2, f_2, p_2, S_2)$, one could use the four basic attributes which define them for comparison, i.e., the operation type, the fragment, the position and the schema. For each of these values, different metrics already exist: For the process schema ($\text{sim}(S_1, S_2)$) and fragment ($\text{sim}(f_1, f_2)$), techniques for the similarity of process models can be used [29, 34, 35]. The positions of the change operations ($\text{sim}(p_1, p_2)$), which are defined by the pre and post set of the element, can be compared by adapting the node matching similarity as defined in [29]. For comparing the change operation type ($\text{sim}(t_1, t_2)$), which is typically a string such as `insert` or `delete`, approaches which compare string similarity [36] or equivalence could be used.

Overall, for $\Delta_1 := (t_1, f_1, p_1, S_1)$ and $\Delta_2 := (t_2, f_2, p_2, S_2)$ the following attribute-based metric can be formulated:

$$sim_{attr}(\Delta_1, \Delta_2) := w_t * sim(t_1, t_2) + w_f * sim(f_1, f_2) + w_s * sim(S_1, S_2) + w_p * sim(p_1, p_2) \quad (2.1)$$

where the sum of the weights $w_t + w_f + w_s + w_p = 1$.

As the following example shows, the weights which should be used for each of these values highly depend on the situation, i.e., the schema, fragment, position, and change operation type.

In Figure 2.6, three process schemas for a surgery in a hospital are shown: The first one displays the process for an adult, the second for an elderly person, who can still care for himself, and the third shows the process for a child, where the parents still have the right to decide. With a few minor differences, the basic process works as follows: First, general information about the patient is being collected, then, he is prepared for surgery, the surgery is being conducted, and finally, a report is delivered.

In certain cases, adaptations to these basic processes are necessary: If a patient requires additional information *before* the surgery, an *Inform Patient* step is added to the process instance, where another doctor consults the patient. This is shown in the changed instance of the elderly process.

Sometimes, the surgery did not reach its desired goal, and another surgery will be necessary. Thus, the patient has to be informed separately *after* the surgery. For this, an *Inform Patient* step is added to the process instance *after* the surgery. This is shown in the changed instance of the adult process.

Following, the *position* of the *Inform Patient* step highly influences its semantic meaning: If it is added before the surgery, it usually just means that another doctor has to be consulted; if it is added after the surgery, it might hint that something went wrong. Thus, when comparing two change operations with the element *Inform Patient*, one may regard the weight of the position $w_{position}$ as very high, while the other weights are quite low.

When doing surgeries on children, their parents have to be included in every step of the process. Each time the patient is informed, his or her parents also

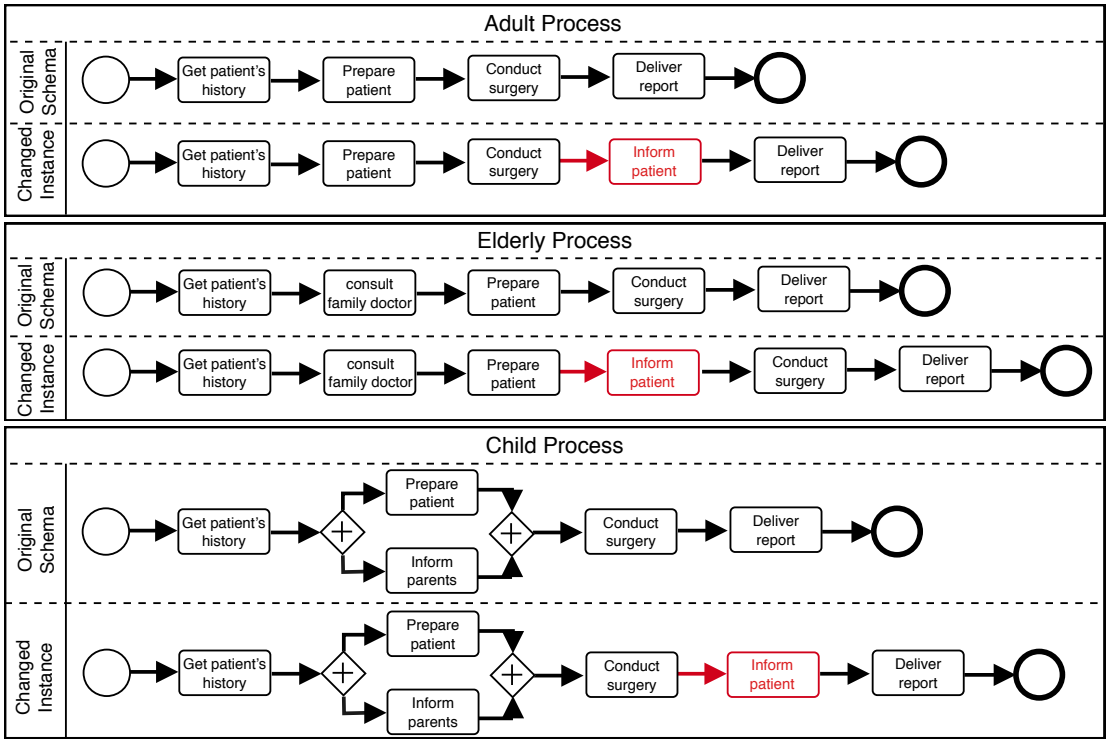


Figure 2.6: Three examples of process change operations

have to be informed. Besides this, the *Inform Patient* step has the same semantic meaning if added before or after the surgery as for the other two processes.

When comparing the change operations of the adult and the elderly patient, the *Inform Patient* step is added for the adult after the surgery, and for the elderly person before the surgery. Since the two process schemas do not have much effect on this particular change operation, but the position does have a very high effect, one may set the position's weight as very high, and the other weights quite low. However, when comparing the process of the elderly patient with the child's process, the child also receives the *Inform Patient* step after the surgery, thus indicating that another surgery might be required. However, since in a child's process the parents also have to be included into every step, the process schema has a higher influence. Thus, the corresponding weights would be different in these two cases.

This example supports the observation that the attribute-based metric for comparing process change operations requires to carefully choose the weights depending on the particular situation changes take place. In other words, the significance of the results highly depends on the choice of the weights. Hence, in the next section, metrics are proposed that are based on change effects, i.e., metrics that

abstract from the change attributes.

2.5 Effect Based Change Similarity Metrics

In the last section we have shown that a comparison metric which is solely based on a process change operation's attributes only yields satisfactory results if the relevance of each of the attributes is known. As an alternative, in this section we provide metrics which are based on the effects of a process change operation. A process change operation may have - depending on the design of the fragment which is inserted to or deleted from the schema - effects on all process perspectives presented in Section 2.3.

A common scenario in which process change operations may be compared is that of some domain expert who uses the resulting data for comparing possible adaptations, or for analyzing past adaptations, as discussed in the introduction. Depending on the question at hand, different perspectives on the process fragment may be of interest. In this section we chose to set our focus on the resource and temporal perspective for two reasons: First, these perspectives can provide answers to interesting questions for a domain expert. For a practitioner such as a nurse in a nursing home the resource and temporal perspectives provide answers to questions such as *How much staff will be required* or *How long will this therapy take?*. Second, metrics which are solely based on the temporal and resource perspective can be used to compare process change operations without any knowledge of the schema after the change, or about the other perspectives of the change operation's fragment. This can be relevant when the whole process cannot be accessed (a) due to privacy issues or (b) because the whole information is not relevant to the question. Think of a nursing home where a nurse has to plan the resources of the upcoming weeks: She may not have to know which steps the therapy processes contain exactly - all she needs is data about the required resources. By removing critical information about the patient's therapy processes (and reducing the information only to the resource and time perspective) this data can also be used with less data privacy issues.

2.5.1 Resource Perspective Metrics

In this section, we present a similarity metric for process change effects from the resource perspective, which captures all required organizational and other resources. When planning a process adaptation, it may be interesting to compare available adaptations based on the resources they require. Thus, one can choose the adaptation which requires the least resources if two or more adaptations are similar otherwise.

For this we present the aggregated resource view for process schemas and process fragments that are used for changes. This view builds the basis for creating a metric to compare the resource perspective of two change operations. For each task, a definition of resources which are connected to this task are required. The effects of a change operation $\Delta = (t, f, p, S)$ on the resources perspective of the underlying process schema S can be determined based on the resource assignments of S and the process fragment f. In order to determine the effects, the aggregated resource view of a process schema/fragment is defined as follows:

Definition 2.4 (Aggregated Resource View)**.** Let $S = (N, E, D, DE, Res, Temp)$ be a process schema. Then the aggregated resource assignment ρ_S for S is defined as

$$\rho_S := \{(r, s) | \exists (r, x) \in \bigcup_{n \in N} Res(n) \wedge s = \sum_{n \in N, (r,x) \in Res(n)} x\}$$

Consider Change Fragment 1 ($CF1$) and Change Fragment 2 ($CF2$) to be inserted or deleted by change operations as depicted in Fig. 2.7 with $CF1 = (N1, E1, D1, DE1, Res, Temp)$ and $CF2 = (N2, E2, D2, DE2, Res, Temp)$. The required resources for each task are depicted in italic as attributes of the task (so for executing task A resource *nurse* is required two times, while for executing task B resource *doctor* is required once.) This leads to the following sets of tasks and aggregated resource views:

- $N1 = \{A, A, B, C\}$ and $N2 = \{A, B, C, D\}$[1]
- For $CF1$: $Res(A)$ = (nurse,2), $Res(B)$ = (doctor, 1), $Res(A)$ = (nurse, 2), $Res(C)$ = (nurse,1)
- For $CF2$: $Res(A)$ = (nurse,2), $Res(C)$ = (nurse, 1), $Res(B)$ = (doctor, 1), $Res(D)$ = (assistant, 1)
- $\rho_{CF1} := \{(nurse, 5), (doctor, 1)\}$
- $\rho_{CF2} := \{(nurse, 3), (doctor, 1), (assistant, 1)\}$

Definition 2.5 (Similarity Metrics for Resource Assignment)**.** Let ρ_1, ρ_2 be two aggregated resource assignments. Then the similarity between ρ_1 and ρ_2 is defined as follows:

$$sim(\rho_1, \rho_2) := \begin{cases} \frac{\sum_{\rho_1, \rho_2} \{|y_1 - y_2| | \exists (r, y_1) \in \rho_1, \exists (r, y_2) \in \rho_2\}}{\sum_{\rho_1, \rho_2} \{max(z) | \exists (r, z) \in \rho_1 \cup \rho_2\}} & if \rho_1 \cup \rho_2 \neq \emptyset \\ 0 & otherwise \end{cases}$$

[1]The sets are seen as bags due to multiple occurrence of activities.

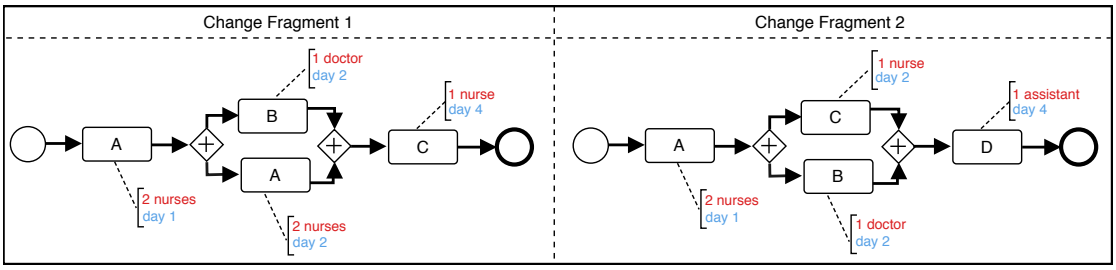


Figure 2.7: Example for Aggregated Resource Perspective in Process Fragments

$sim(\rho_1, \rho_2)$ relates the number of resources that are required for ρ_1 and ρ_2 to the maximum number of each required resource. For the example shown in Fig. 2.7 we have $\rho_1 := \{(nurse, 5), (doctor, 1)\}$ and $\rho_2 := \{(nurse, 3), (doctor, 1), (assistant, 1)\}$. Thus 4 resources are required for ρ_1 and ρ_2, i.e., 3 nurses and 1 doctor. The maximum number for each particular resource are 5 (nurse), 1 (doctor), and 1 assistant, summing up to 7. Overall, $sim(\rho_1, \rho_2) = \frac{4}{7} \approx 0.57$ for this example.

Metric $sim(\rho_1, \rho_2)$ can be calculated for the resource assignments of two fragments $f1$, $f2$ that are used by change operations $\Delta_1 = (t1, f1, p1, S1)$ and $\Delta_2 = (t2, f2, p2, S2)$. Doing so it becomes possible to measure the effects on the resource assignments of the underlying process schemas $S1$ and $S2$. It has only be further distinguished whether Δ_1 and Δ_2 insert or delete the fragments. For insertion, typically, resource assignments will be added, for deletion, resource assignments will be removed. Note that intentionally the resource assignments of the underlying schemata $S1$ and $S2$ are not considered as the metric is designed in an independent manner. The reason behind is to enable similarity calculation also for different schemas $S1$ and $S2$.

Definition 2.6 (Change Resource Similarity (CRS))**.** Let $\Delta_1 = (t1, f1, p1, S1)$, $\Delta_2 = (t2, f2, p2, S2)$ be two change operations and ρ_1 (ρ_2) the aggregated resource assignment for $f1$ ($f2$). The Change Resource Similarity CRS between changes Δ_1 and Δ_2 is defined as follows:

$$CRS(\Delta_1, \Delta_2) := \begin{cases} sim(\rho_1, \rho_2) if t1 = t2 \\ -sim(\rho_1, \rho_2) otherwise \end{cases}$$

2.5.2 Towards a Similarity Metric for the Timed Resource Perspective

The CRS metric compares the required resources of a process fragment. Sometimes, especially in long-running process settings, the time perspective is also relevant. Think about a nursing home planning the required resources for the next weeks: It may be interesting how two change operations affect the resource view

only in a certain time frame. Definition 2.7 incorporates the time information into the aggregated resource view:

Definition 2.7 (Timed Aggregated Resource View)**.** Let $S = (N, E, D, DE, Res, Temp)$ be a process schema with aggregated resource view ρ_S. The timed aggregated resource view τ_S is defined as follows:

$\tau_S := \{(r, s, t1, t2) | (r, s) \in \rho_S,$
$t1 = min\{Temp(n) | n \in N \wedge \exists (r, x) \in Res(n)\},$
$t2 = max\{Temp(n) | n \in N \wedge \exists (r, x) \in Res(n)\}\}$

Informally, the timed aggregated resource view determined for each tuple in the aggregated resource view the earliest and latest point in time the associated resource was required. For the change fragments $CF1$ and $CF2$ depicted in Fig. 2.7, we obtain

τ_{CF1} = {(nurse, 5, 1, 4), (doctor, 1, 2, 2)} and
τ_{CF2} = {(nurse, 3, 1, 2), (doctor, 1, 2, 2), (assistant, 1, 4, 4)}.

Similarity between the timed aggregated resource views of two process schemas or fragments can be defined as follows. For comparing the temporal perspective a simple comparison between the intervals for each aggregated resource is utilized, i.e., calculating the differences between upper and lower interval limit divided by the sum of the interval lengths. This similarity value is combined with the CRS by weighing both equally. Note that if more complex temporal information is assigned to the tasks, more sophisticated similarity metrics can be used.

Definition 2.8 (Similarity Metrics for Timed Resource Assignment)**.** Let τ_1, τ_2 be two timed aggregated resource assignments for resource assignments ρ_1 and ρ_2. Then the similarity between τ_1 and τ_2 is defined as follows:

$$sim(\tau_1, \tau_2) := \begin{cases} \frac{1}{2} * (sim(\rho_1, \rho_2) + \frac{\sum_{t1,t2} sim(t1,t2)}{max(|\tau_1|,|\tau_2|)}) & if \tau_1 \cup \tau_2 \neq \emptyset \\ 0 & otherwise \end{cases}$$

where $t1 = (r1, s1, l1, u1) \in \tau_1, t2 = (r2, s2, l2, u2) \in \tau_2$ and

$$sim(t1, t2) := \begin{cases} 1 & if r1 = r2 \wedge l1 = l2 \wedge u1 = u2 \\ 1 - \frac{|l1-l2|+|u1-u2|}{|u1-l2|+|u2-l1|)} & if r1 = r2 \\ 0 & otherwise \end{cases}$$

The metrics combines the similarity between the aggregated resource views with the more specific assessment of the resource requirements related to time. Each value can be also considered in a separated manner.

The corner cases for the metrics would be to have a) highly similar or equal resource assignments that are due at the same time and b) highly similar or equal

resource assignments that are due at totally different times. For case a), intuitively, the timed aggregated resource metrics yields a value close to or equal 1. For case b) assume

$\rho_1 = \{(nurse, 3, 1, 3), (doctor, 1, 4, 5)\}$ and $\rho_2 = \{(nurse, 3, 4, 5), (doctor, 1, 6, 7)\}$. In this case $sim(\rho_1, \rho_2) = 1$. Then: $sim((nurse, 3, 1, 3), (nurse, 3, 4, 5)) = 1 - \frac{3+2}{1+4} = 0$ and
$sim((doctor, 1, 4, 5), (doctor, 1, 6, 7)) = 1 - \frac{2+2}{1+3} = 0$. Hence, $sim(\tau_1, \tau_2) = 0.5$. This means that the high similarity between ρ_1 and ρ_2 is reduced by half because the same resources are required, but at totally different times.

For the example in Fig. 2.7, applying Def. 2.8 yields:

$sim((nurse, 3, 1, 2), (nurse, 5, 1, 4)) = 1 - \frac{2}{1+3} = 0.5$ and
$sim((doctor, 1, 2, 2), (doctor, 1, 2, 2)) = 1$.

Then: $sim(\tau_{CF1}, \tau_{CF2}) = \frac{0.57 + \frac{0.5+1}{3}}{2} = 0.54$.

This means that the deviations in the time assignments reduce the similarity of the aggregated resource assignment a bit.

For an example where $\rho_1 = \{(nurse, 3, 2, 4)\}$ and $\rho_2 = \{(nurse, 1, 2, 4)\}$ the resource assignment similarity would yield $sim(\rho_1, \rho_2) = 0.66$.

Looking at the time requirements, $sim((nurse, 3, 2, 4), (nurse, 2, 2, 4)) = 1$ and hence $sim(\tau_1, \tau_2) = \frac{0.66+1}{2} = 0.83$.

In this case the similarity increased when incorporating the time as a different number of the same resource is required at exactly the same time.

For comparing changes along their timed aggregated resource views a first proposal for a similarity metric is as follows:

Definition 2.9 (Timed Change Resource Similarity (TCRS))**.** Let Let $\Delta_1 = (t1, f1, p1, S1)$, $\Delta_2 = (t2, f2, p2, S2)$ be two change operations and τ_1 (τ_2) the timed aggregated resource assignment for $f1$ ($f2$). The Timed Change Resource Similarity $TCRS$ between changes Δ_1 and Δ_2 is defined as follows:

$$TCRS(\Delta_1, \Delta_2) := \begin{cases} sim(\tau_1, \tau_2) if t1 = t2 \\ -sim(\tau_1, \tau_2) otherwise \end{cases}$$

Looking at the different examples and corner cases above, TCRS seems to make sense. However, the observations from the metrics start to become blurred if the earliest and latest point in time a resource is required span a longer time frame. The interpretation can be different. The resources could be required rather at the beginning and the end of the process or during the entire execution of the process. Both cases would be treated the same. Hence, a more fine-granule comparison becomes necessary. This aspect will be addressed in future work.

2.6 Discussion and Comparison to Other Similarity Metrics

In this section we discuss the applicability of our approach, analyze the importance of weights on the attribute based similarity metric presented in Section 2.4, and compare the results of the presented similarity metrics to other existing similarity metrics for processes.

For the evaluation we have used data set 1 from the Apelands project[2]. This data set contains 136 change operations for 23 different basic models, of which 107 are based on the same model. Apelands is a game-based experimentation and evaluation service for flexible process settings, described in detail in Chapter 7. While playing the round-based game, players adapt process instances, thus generating process change logs. These change logs contain all perspectives required for the metrics presented in this thesis, i.e., a list of required resources for each activity which can be added to a process instance, and information about the game round the activity is supposed to be executed. Listing 2.6 shows a change fragment from the game. Each contains two activities which have been planned for the next two rounds (cf. <round/> tag). In the <resources/> Tag, the required resources are shown.

Listing 2.1: Apelands Change Fragment

```
<fragment>
  <call>
    <name>'Fire Elemental'</name>
    <round>1</round>
    <resource name="Alchemy Lab" count="2"/>
  </call>
  <call>
    <name>'Knight'</name>
    <round>2</round>
    <resource name="Barracks" count="2"/>
  </call>
</fragment>
```

2.6.1 Discussion of Applicability

Using (T)CRS, one can compare process change operations even when there is no information about the resulting process schemas or about other perspectives of the process change operations. This can be useful for cases where (a) other

[2]The data set can be found at http://cs.univie.ac.at/project/apes

information is not available, e.g., due to data privacy issues or (b) if the other data is not required to the current analysis. Thus, change operation similarity based on the temporal and resource perspective alone can be used as an alternative to structural or behavioral similarity metrics. We will show that the results from (T)CRS actually correlate with structural similarity of the process schemas after the change operation has been executed.

2.6.2 Analysis of the Attribute based Similarity Metric

First, we evaluate our assumption regarding the attribute similarity metric. In Section 2.4, we argued that the choice of weights depends heavily on the given situation. In this section, we want to show that the choice of weights actually has a huge effect on the results. We compare one randomly chosen change operation with all other change operations from the data set. We chose three different sets of weights for the position, the fragment and the schema, resulting in three different sets of results (shown in Figure 2.8). For comparison *weighted_1* (blue set), we chose $\frac{1}{3}$ each for the weights of position, fragment and schema. For comparison *weighted_2* (orange set), we chose 0.9 for the weight of the schema, and 0.05 each for the schema and the position. For comparison *weighted_3* (green set), we chose 0.9 for the weight of the fragment, and 0.05 each for the schema and the position. Following, Figure 2.8 shows that by adapting the weights almost any similarity between two process change operations can be created. Thus, a purely attribute-based metric is not ideal for comparing process change operations.

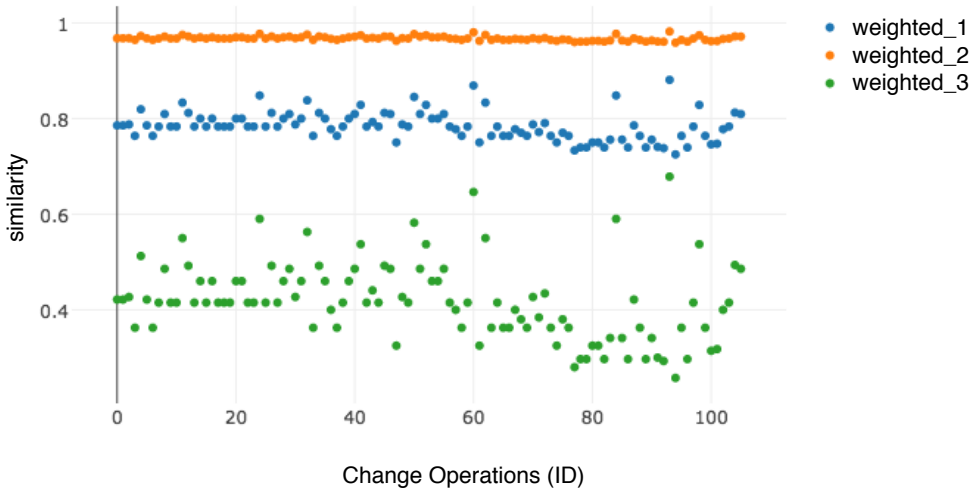


Figure 2.8: Weighted Change Operation Attribute Similarity

2.6.3 Comparing (T)CRS Effect Similarity with Structural Similarity

We evaluate the effect similarity of two change operations as measured by (T)CRS against *node matching similarity* (NMS) [29] of the resulting process schemas. NMS compares the similarity of all nodes in two process schemas and results in a value between 0 and 1.

Figure 2.9 shows the approach used in this evaluation: Since no similarity metrics exist which could be used as a reference for comparing process change operations directly, we used the node matching similarity of the resulting process schema as a reference instead. The idea works as follows: If two process instances I_1 and I_2 start from the same schema S, the node matching similarity $sim_{nms}(S, S) = 1.0$ for I_1 and I_2. If now change operation Δ_1 transforms schema S to S_1 for instance I_1, and change operation Δ_2 transforms S to S_2 for instance I_2, the similarity $sim_{nms}(S_1, S_2) <= 1.0$. However, the more similar the two applied change operations Δ_1, Δ_2 are, the bigger the similarity of the resulting schemata S_1, S_2 should be. Thus, the similarity of the applied change operation positively correlates with the node matching similarity of the resulting process schemata for insertion.

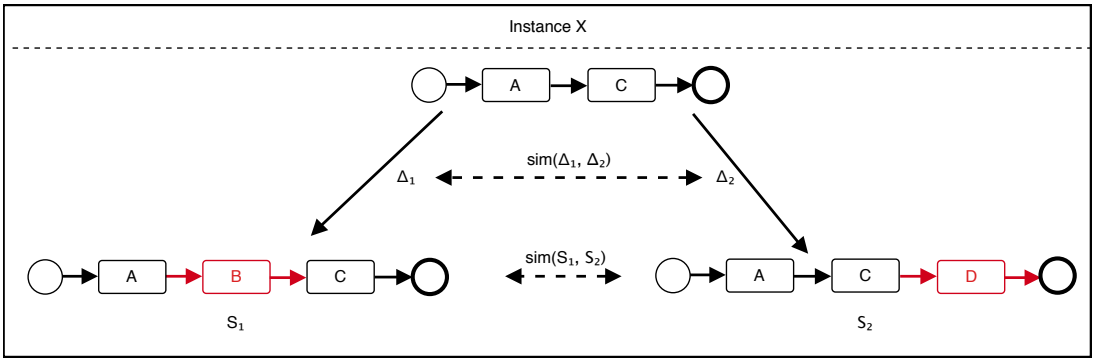


Figure 2.9: Comparing Node Matching Similarity of the resulting process instances with change similarities

We compare a randomly chosen change operation from the *Apelands* data set against all other change operations. As Figure 2.10 and 2.11 show, both CRS and TCRS correlate with NMS. In most cases, the similarity as calculated by (T)CRS is higher than NMS, since (T)CRS are based on a subset of the attributes of the process models. Attributes not considered by (T)CRS which are different do not have any effect on (T)CRS scores, but lower the NMS score. This relationship can also be seen in the fact that TCRS has a closer correlation to NMS than CRS, since it also incorporates the time-related attributes, which are not used by CRS.

Conclusion: This evaluation shows that the similarity of change operation effects as calculated by (T)CRS correlates with the similarity of the resulting process schemas. Thus, such a metric can be used to compare process change operations independent of attribute-specific weights.

Threats to validity: Our approach was evaluated with experimental data generated from a game, not with data from several different settings in which change operations occur. Also, we did not interview domain experts about the impact of the results of our metrics. These evaluations would be interesting additions to our validation which is based on a state of the art approach from process similarity.

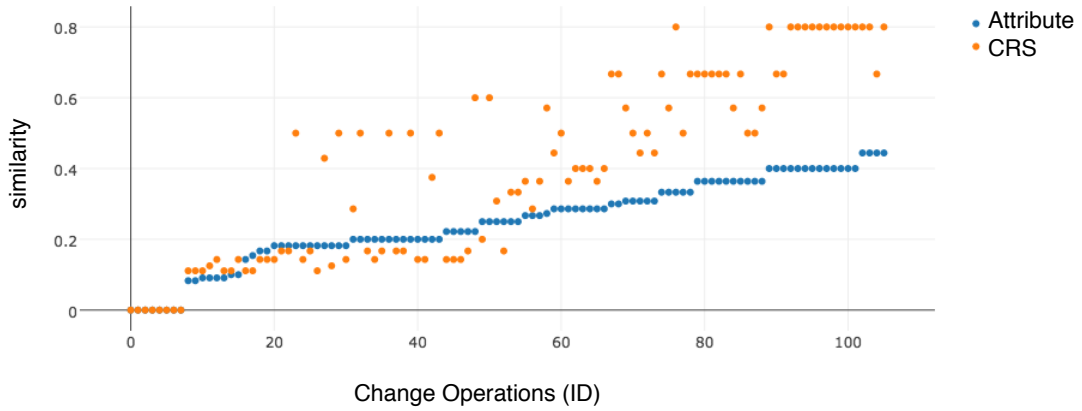


Figure 2.10: Comparing CRS with the attribute similarity

2.7 Discussion

Process change operations are usually applied when the situation requires domain experts to do so. They are used to transform one process schema to another, thus changing the set of events which may be executed in the future. Thus, change operations provide the basic for process flexibility in a procedural process management system [20]. Being able to show which change operations have similar effects on a certain perspective can be interesting for the person in charge of the adaptation: When adapting therapy processes in a nursing home, using a resource perspective metric the nurse can see which therapy process fragments require similar resources. In conjunction with information about available resources for the upcoming weeks, such a clustering can facilitate the planning of process adaptations. When evaluating the effects of former process adaptations, comparison metrics can be used to create cluster of similar process change operations, thus improving analysis.

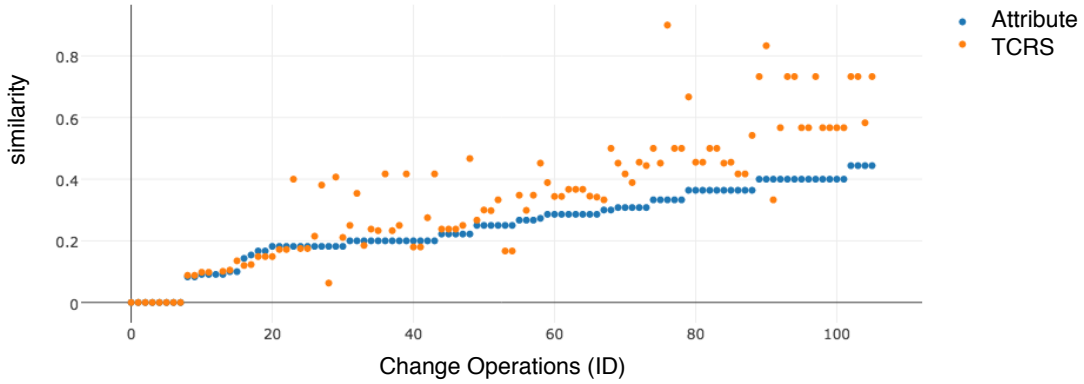


Figure 2.11: Comparing TCRS with the attribute similarity

In this chapter, we have solely focused on analyzing single process change operations. However, for realistic applications, a single change operation is not sufficient: Instead, they are stored in change traces. This concept will be discussed in the following chapter.

Chapter 3

Representation and Analysis of Process Change Traces

In this chapter, we discuss several representation and analysis methods for change traces. Figure 3.1 shows the position of this topic in the structure of this dissertation: In Chapter 2, single process change operations have been introduced, which are the basic building blocks for change traces. However, in realistic data sets, single process change operations always exist in the context of a change trace. Hence, after introducing the basics in the previous chapter, in this chapter, we focus on process change traces. After an introduction to the topic, machine readable representations and change processes, a representation and analysis method for change traces are introduced and analyzed. This is followed by techniques for filtering and projecting change traces. This chapter is based upon the paper *"Mining and Querying Process Change Information Based on Change Trees"* [37].

3.1 Overview

A change operation defines one single change to a process schema of one specific process instance. However, in a realistic setting, a single change operation is not sufficient to describe the changes which have been applied to a process instance. Examples can be found in almost any field of work: Imagine a manufacturing process, which needs to be adapted because multiple problems occur with the suppliers during the execution of an instance of this manufacturing process, e.g., because they did not deliver their products on time: In this case, some activities may have to be changed at different points in time (each time a problem is detected), which would result in multiple distinct process change operations. In the medical domain, changes can be required because of a change in the patient's health state. Imagine a patient staying for a long time in a hospital: Whenever

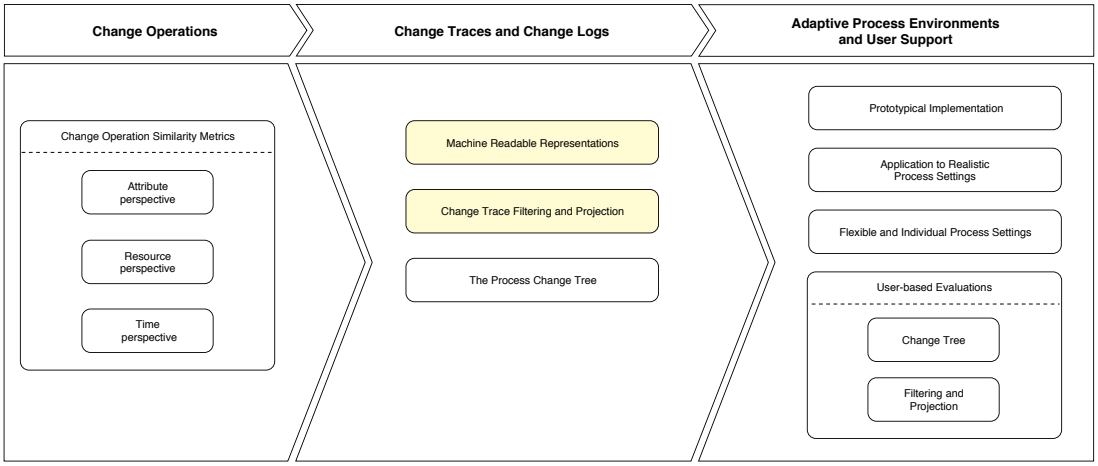


Figure 3.1: Dissertation Overview: Process Change Traces

the patient's health changes, some therapies have to be added to, while others may have to be removed from his individual therapy process instance. Each of these change operations is related to one specific process instance, and thus needs to be accessible, so that the whole adaptation history of the affected process instance can be traced. For this, the concept of *change traces* is introduced.

A change trace is a temporally ordered set of high level process change events, where each change event represents one uniquely identifiable change operation. For example, the left pane of Figure 3.2 depicts two change operations which have been applied consecutively to process instance 1: First, Δ_1 `inserts` process activity X between the AND join and activity D, then, Δ_2 `deletes` process activity A. Thus, formally $\Delta_1 = (\texttt{insert}, X, \{AND - Join1, D\}, S)$ transforms process schema S to S_1', which is followed by $\Delta_2 = (\texttt{delete}, A, \{Start, AND - Split1\}, S'_1)$, transforming S_1' to S_1". In the right pane, schema S is first transformed to S_2' by change operation $\Delta_3 = (\texttt{delete}, A, \{Start, AND - Split1\}, S)$, which is consecutively transformed to S_2" by change operation $\Delta_4 = (\texttt{insert}, X, \{D, End\}, S'_2)$. Thus, there exist two change traces for process instances I1 and I2:

- $ct_{I_1} =< (\Delta_1, \text{"c0"}), (\Delta_2, \text{"c1"})>$
- $ct_{I_2} =< (\Delta_3, \text{"c2"}), (\Delta_4, \text{"c3"})>$

Here, each change event contains the respective change operation, as well as a unique identifier (c0, c1, c2 and c3). Formally, we define the concepts of change events, change traces, and change logs as follows:

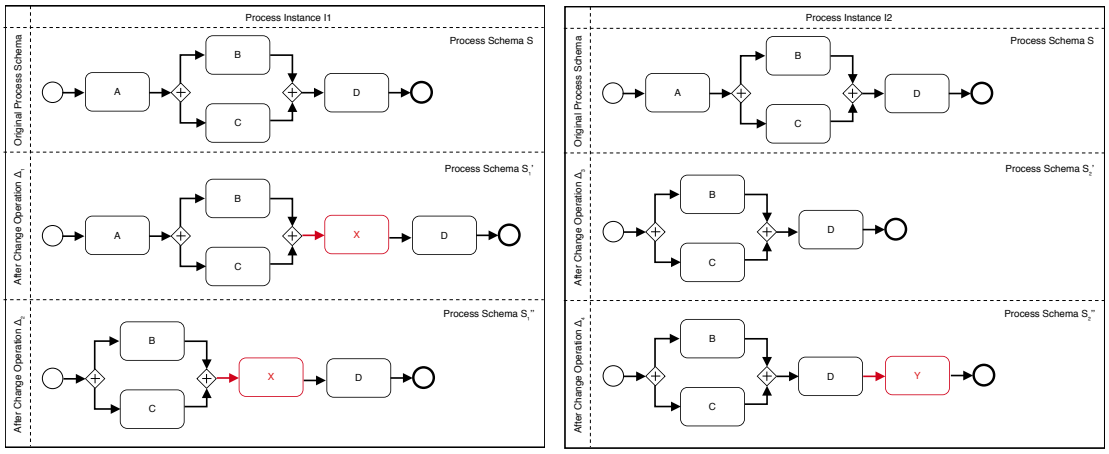


Figure 3.2: Two Change Operation Traces

Definition 3.1 (Change Event, Change Trace, Change Log)**.** Let $\mathcal{I}$ be a set of process instances based on schema S. A change log $\mathcal{C}$ is the set of change traces for all $I \in \mathcal{I}$. For each $I \in \mathcal{I}$ there exists exactly one trace $ct \in \mathcal{C}$. The change trace ct for I is the temporally ordered set of change events $< c_1, c_2, \ldots, c_n >$. A change event c in trace ct is defined as:

$c := (\Delta, id)$, where Δ is the applied change operation, and id is a globally unique identifier for the change event.

For simplicity of representation, we will omit this unique ID below whenever it is not required. Hence, for this section's running example of the process instances I1 and I2, the corresponding change traces can be represented as follows:

- I1: $< \Delta_1, \Delta_2 >$
- I2: $< \Delta_3, \Delta_4 >$

Assume the following change log, which is depicted for instances I_1 and I_2 in Figure 3.2:

$\mathcal{C} = \{I_1, I_2\}$
I_1 = < Δ_1 = (insert, X, <AND-Join1, D>, S),
Δ_2 = (delete, A, <Start, AND-Split1>, S_1')>
I_2 = < Δ_3 = (delete, A, <Start, AND-Split1>, S),
Δ_4 = (insert, Y, <D, End>, S_2')>
.

The ordering of the change operations in the change traces for I_1 and I_2 implies the execution order of the change events: Δ_1 has been executed before Δ_2, and Δ_3 has been followed by Δ_4.

In Figure 3.2, both change operations Δ_2 and Δ_3 delete activity A between the process start and the AND split; however, they are applied to different process schemas (Δ_2 is applied to S_1', while Δ_3 is applied to S). In order to increase comparability of change operations in the context of process change traces, we will always assume the unchanged process schema of the related process instance (in this case, S), instead of the adapted instance schema. Thus, $\Delta_2 = \Delta_3$, and the resulting change log for the process instances $\{I1, I2\}$ is $\mathcal{C} = \{< \Delta_1, \Delta_2 >, < \Delta_2, \Delta_4 >\}$.

Change traces contain all change operations which have been applied to a process instance. In this chapter, we analyze existing representation and analysis methods for change traces. Machine readable representations provide the data basis for representation and analysis tools such as ProM [38]. In addition to machine readable representations, dedicated analysis methods for change traces such as change processes exist, which highlight causal dependencies between applied change operations. Both for machine readable representations which can be used as a data basis for various analysis techniques, as well as for analysis methods such as change processes, it is important that users who inspect the data using visual analysis tools (e.g., using tools such as ProM) can focus on the information they need for a certain analysis question [39]. The *information overload problem* [11] describes a situation where data has not been prepared for visual analysis: While the relevant information may be available in the data set, it can be hard to see due to several other, for the current task irrelevant information. Examples for information overload in process change logs can be found in various domains: In the medical domain, a process change log may contain information about change operations which have been applied to a patient's therapy process. When a doctor has to inspect those changes, but only wants to focus on those which have been applied due to a certain problem (e.g., the patient catching the flu), he has to find this information in the change log, which also contains data of changes applied for various other reasons. Following such an analysis, he may want to focus on those change operations, which have led to the recovery of the patient in a certain amount of time, e.g., less than one week. In the manufacturing domain, a change log may contain all changes applied to production processes of some product. When a manager is interested in the changes which have been applied because a supplier did not deliver the required resources on time, he has to find the relevant adaptations in the change log. Next, he may want to find all adaptations which solved the issue (i.e., providing the required resources) in less than two days. While the information of interest may have been available in the change log, and the corresponding execution log from the beginning, in order to overcome the information overload problem, the log has to be preprocessed for the current analysis question. Hence, in Section 3.4, we discuss how the information present

in a change log can be prepared in order to overcome the information overload problem.

In this chapter, we will inspect representation and analysis methods for change logs which currently exist, and analyze whether they can represent all information contained in a change log. Additionally, we will discuss how the information overload problem can be tackled for change traces. Summarizing, this chapter tackles the following research questions:

Q1: Which analysis methods and machine readable representations for analyzing change traces currently exist? How can they be improved?

Q2: How can change logs be transformed, such that a change log contains only the data relevant for a specific analysis question, thus overcoming the information overload problem?

These research questions are tackled as follows: First, we take a look at a machine readable change log representation, i.e., MXML. Based on identified shortcomings of the MXML representation, we present an XES based representation of change logs. XES is an XML-based format for representing process logs (Section 3.2). Next, we analyze change processes (Section 3.3), an existing human readable representation for change logs, and show some of its shortcomings based on example log files. To overcome the information overload problem described above, we present the concept of *filtering* change traces, in order to constrain the set of change traces contained in a change log to those who fit a certain (set of) criteria (cf. Section 3.4.1). Additionally, we analyze how change operations in change logs can be *projected* according to the case they have been applied for (cf. Section 3.4.2), and combine filtering and case-projection techniques into a single approach (cf. Section 3.4.3).

3.2 Machine Readable Change Log Representations

Machine readable presentations are an important asset for analyzing process logs. Process mining frameworks as ProM use data encoded in such representation as a basis for various visualization and analysis tools [38]. XES (eXtensible Event Streams)[1] is such a representation format for event-driven data. In this section, we will show how XES can be extended to also include information relevant for process change logs.

XES is the successor of MXML [40], which was used for storing process logs in the past. In contrast to its predecessor, XES has many advantages, such as

[1] http://www.xes-standard.org

the possibility to dynamically enhance the language based on the requirements of the setting [41]. In contrast to MXML, in XES no attributes have a predefined meaning.

In [9], the authors define a representation of change log information based on MXML. Listing 3.2 represents the change operations applied to process instance I1 in Figure 3.2. Formally, the change trace can be defined as follows:

$\mathcal{C}_{I1} =< (\texttt{insert}, X, \{AND{-}Join1, D\}, S), (\texttt{delete}, A, \{Start, AND{-}Split1\}, S) >$

In MXML, each `AuditTrailEntry` represents a change event in MXML.

Listing 3.1: Change Log in MXML

```
<Process id="main_process">
  <Data>
    <Attribute name="description">change log for processes
       based on schema S</Attribute>
  </Data>
  <ProcessInstance id="I1">
    <AuditTrailEntry>
      <Data>
        <Attribute name="CHANGE.postset">D</Attribute>
        <Attribute name="CHANGE.type">INSERT</Attribute>
        <Attribute name="CHANGE.subject">X</Attribute>
        <Attribute name="CHANGE.rationale">Reason for
           inserting activity X.</Attribute>
        <Attribute name="CHANGE.preset">AND-Join1</Attribute>
      </Data>
      <WorkflowModelElement>INSERT.X</WorkflowModelElement>
      <EventType>complete</EventType>
      <Originator>Peron who conducted the change</Originator>
    </AuditTrailEntry>
    <AuditTrailEntry>
      <Data>
        <Attribute name="CHANGE.postset">AND-Split1</
           Attribute>
        <Attribute name="CHANGE.type">DELETE</Attribute>
        <Attribute name="CHANGE.subject">A</Attribute>
        <Attribute name="CHANGE.rationale">Reason for
           deleting activity A.</Attribute>
        <Attribute name="CHANGE.preset">Start</Attribute>
      </Data>
      <WorkflowModelElement>DELETE.A</WorkflowModelElement>
      <EventType>complete</EventType>
      <Originator>Peron who conducted the change</Originator>
    </AuditTrailEntry>
```

```
  </ProcessInstance>
</Process>
```

MXML was designed to capture the flow of event execution for a specific, predefined process. Each change event is represented by an `AuditTrailEntry`, which stores all attributes relevant to the change operation in the `Data` item. The position of the change operation is represented by `CHANGE.preset` and `CHANGE.postset`, the type by `CHANGE.type`, the activity or process fragment affected by `CHANGE.subject`, and a reason for the change can be given in `CHANGE.rationale`.

This representation comes with several shortcomings: First, due to the requirements imposed by process mining engines such as ProM [38], the `WorkflowModelElement` has to be present, although it has no specific meaning in the context of a change process [9]. Second, the `EventType` attribute captures the type of the event that has happened; e.g., if some activity has been started, scheduled, or completed. Since process change operations are logged when a change to the process instance has been conducted, this state is always set to `complete`; hence, this element is obsolete. Third, using MXML, it is not possible to attach the semantics of an event attribute. This is solved by the concept of *extensions* in XES.

Basically, XES extensions allow for the definition of arbitrary elements relevant in a certain context. These elements can be defined for up to four different levels of abstraction, i.e., *log*, *trace*, *event*, and *meta*. These abstraction levels have the following meaning:

- *Log:* Elements defined for the whole XES log.
- *Trace:* Elements defined at the trace level.
- *Event:* Elements defined at the event level.
- *Meta:* This abstraction level is reserved for nested attributes.

Listing 3.2 shows the XES extension for change logs.

Listing 3.2: XES Extension for Change Logs

```
<?xml version="1.0"?>
<xesextension name="Change" prefix="change" uri="http://
   leonardo.wst.univie.ac.at/acaplan/change.xesext">
  <log>
    <string key="subjectsrepository">
      <alias mapping="EN" name="Address of the repository
         where subjects can be found"/>
      <alias mapping="DE" name="Adresse des Repositories, in
         dem man auf die Subjekte zugreifen kann"/>
    </string>
```

```
  <string key="schemarepository">
    <alias mapping="EN" name="Address of the repository
       where schemas can be found"/>
    <alias mapping="DE" name="Adresse des Repositories, in
       dem man auf die Prozessschemata zugreifen kann"/>
  </string>
  <string key="instancerepository">
    <alias mapping="EN" name="Address of the repository
       where process instances can be found"/>
    <alias mapping="DE" name="Adresse des Repositories, in
       dem man auf die Prozessinstanzen zugreifen kann"/>
  </string>
</log>
<trace>
  <string key="schema">
    <alias mapping="EN" name="Name of the schema of the
       affected process instance"/>
    <alias mapping="DE" name="Name des Schemas der
       betroffenen Instanz"/>
  </string>
  <string key="instance">
    <alias mapping="EN" name="Name of the instance affected
        by the change operations"/>
    <alias mapping="DE" name="Name der von den Operationen
       betroffenen Instanz"/>
  </string>
</trace>
<event>
  <string key="preset">
    <alias mapping="EN" name="Element before the change
       operation"/>
    <alias mapping="DE" name="Element vor der
       Aenderungsoperation"/>
  </string>
  <string key="postset">
    <alias mapping="EN" name="Element after the change
       operation"/>
    <alias mapping="DE" name="Element nach der
       Aenderungsoperation"/>
  </string>
  <string key="type">
    <alias mapping="EN" name="Typ"/>
    <alias mapping="DE" name="Typ"/>
```

```
    </string>
    <string key="subject">
      <alias mapping="EN" name="Activity / Fragment affected
         by the change operation"/>
      <alias mapping="DE" name="Aktivitaet / Fragment, das
         durch die Operation betroffen ist"/>
    </string>
    <string key="rationale">
      <alias mapping="EN" name="Reason"/>
      <alias mapping="DE" name="Begruendung"/>
    </string>
  </event>
</xesextension>
```

The XES extension for change logs defines elements for the trace, the event and for the log level. Since each trace is related to exactly one process instance, and this process instance is always based on exactly one process schema, these two are referenced here. For each event, all relevant attributes are present as they were discussed in [9]: The `preset` and `postset` define the position of the change operation, the `type` defines the operation to be conducted (e.g.., `insert`, `delete`, `move`, ...), the `subject` references the affected process fragment, and the `rationale` allows for reasons to be added to the change operation. Since change subjects, instances, and schemas are only referenced by name, links to the repositories where these data can be accessed can be provided on the `log` level.

Based on the definition of the change log extension, the MXML code from Example 3.2 can be translated to XES as follows:

Listing 3.3: XES Change Log Example

```
<?xml version="1.0"?>
<log xes.version="2.0" xes.features="arbitrary-depth">
  <extension name="Identity" prefix="identity" uri="http://
     www.xes-standard.org/identity.xesext"/>
  <extension name="Time" prefix="time" uri="http://www.xes-
     standard.org/time.xesext"/>
  <extension name="Change" prefix="change" uri="http://
     leonardo.wst.univie.ac.at/acaplan/change.xesext"/>
  <classifier name="Change" keys="change:type change:subject
     "/>
  <classifier name="Position" keys="change:preset change:
     postset"/>
  <string key="change:subjectsrepository" value="http://
     leonardo.wst.univie.ac.at:9342/fragments/"/>
  <string key="change:schemarepository" value="http://
```

```
    leonardo.wst.univie.ac.at:9342/schemas/"/>
  <string key="change:instancerepository" value="http://
    leonardo.wst.univie.ac.at:9342/instances/"/>
  <trace>
    <string key="change:schema" value="Schema 1"/>
    <string key="change:instance" value="instance 1"/>
    <event>
      <date key="time:timestamp" value="2018-02-01 19:13:26
        +0200"/>
      <int key="change:transaction" value="0"/>
      <string key="identity:id" value="change_0"/>
      <string key="change:preset" value="D"/>
      <string key="change:postset" value="AND-Join1"/>
      <string key="change:type" value="insert"/>
      <string key="change:subject" value="X"/>
      <string key="change:rationale" value="Reason for
        inserting activity X."/>
    </event>
    <event>
      <date key="time:timestamp" value="2017-02-02 14:27:49
        +0200"/>
      <string key="identity:id" value="change_1"/>
      <string key="change:preset" value="Start"/>
      <string key="change:postset" value="AND-Split1"/>
      <string key="change:type" value="delete"/>
      <string key="change:subject" value="A"/>
      <string key="change:rationale" value="Reason for
        deleting activity A."/>
    </event>
  </trace>
</log>
```

For a change log in XES, three extensions are necessary: The *time* extension enhances XES with timestamps. In the context of a change log, each event has exactly one timestamp, which represents the exact time this change operation has been conducted at.. The *Identity* extension defines a unique identifier for each element. In the case of the change log, it provides each event with a unique identifier, which may serve as the label of the event. In the example given above, these unique identifiers are called `change_0` and `change_1`. The *change* extension then defines all other elements relevant for a change log, as defined above.

3.3 Change Processes

Change processes [9] are a state of the art representation for change traces. In this section, we will introduce the concept, and point out some of the advantages and shortcomings.

3.3.1 Introduction to Change Processes

Change processes are built upon the idea of generating a process schema based on process change logs. For event logs, such algorithms exist as well: Algorithms such as the Alpha Algorithm [42] or the multi-phase algorithm [43, 44] use the information from an execution log to build a process model.

When applying such mining algorithms to change logs, the resulting process then shows the causal relations between the process change events from this log: If, for example, change event Δ_1 is always followed by Δ_2, a causal relation is deduced: Δ_1 causes Δ_2. If two change operations reside in parallel branches of the resulting change process, no such causal connection has been deduced from the change log. However, due to the inherent nature of process change operations, the structure of change logs is very diverse: If the necessity for the process change operation would have been known right from the start, the process instance could have been planned according to the requirements; thus, no change operation would exist. Following, change operations usually deal with exceptional or unpredictable situations; hence, the resulting change logs can become very diverse.

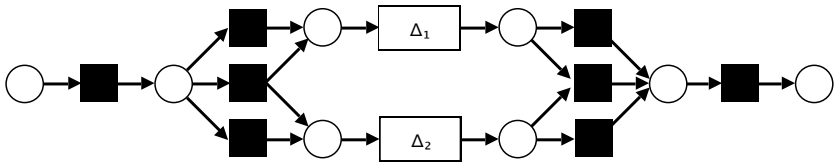
In order to apply existing process mining algorithms to change logs without resulting in spaghetti-like process models, the concept of *commutativity* is introduced [9]. Basically, commutativity of change operations can be summarized as follows:

If any two change operations Δ_1 and Δ_2 can be applied in any order (Δ_1 (Δ_2) followed by Δ_2 (Δ_1)), while still resulting in the same process schema after both change operations have been applied, then the order of these two change operations does not matter.

Process mining algorithms such as the multi-phase algorithm are based upon the causality of events: If an event `a` is always followed by an event `b`, a causality link can be established; i.e., `a` *causes* `b`. Following, these two events would be displayed as a sequence in the resulting process model.

In combination with the introduced concept of *commutativity*, change logs can significantly reduce the number of activities in the resulting change process [9].

Table 3.1: Multiple Instance Occurrences

Change Log	Change Process
I01-I09: <Δ_1> I10: <Δ_2> I11-I13: <Δ_1, Δ_2> I14: <Δ_2, Δ_1>	

3.3.2 Examples of Change Processes

In this section, we will display some exemplary change logs, show the resulting processes and discuss some of the advantages and shortcomings of this form of representation. The change processes which are depicted here have been created with the ProM implementation [38] of the approach described in [9]. Note that the resulting change process as for example shown in Table 3.1 has been transformed into a Petri Net in order to reason about the semantics of splits.

Table 3.1 depicts the change process for a change log, where for nine instances, only change operation Δ_1 has been carried out. For one change operation, Δ_2 has been executed, for three instances, Δ_1 is followed by Δ_2, and for one instance Δ_2 is followed by Δ_1. While the OR split in the resulting change process correctly describes the causality relations between the different change operations in the underlying change log, several characteristics cannot be seen in this representation: First, it is not clear that there were actually four different change traces, which require different change operations. Second, based on the change process alone, it cannot be seen anymore which change trace was used for most process instances: While change trace $< \Delta_1 >$ has been executed for nine out of 14 process instances, trace $< \Delta_2 >$ has only been executed once. This information is lost in the change process representation.

The change process in Table 3.2 cannot correctly reflect the difference between the scenario where only change Δ_1 has been applied, and the two others where Δ_1 respectively Δ_2 followed Δ_1. Additionally, the multiple occurrence of the same change cannot be reflected correctly in the change process.

Table 3.3 summarizes the problem of multiple occurrences of the same process change operations in one change log: When analyzing the change process, one cannot see any more, that for some change traces, change operation Δ_1 has been executed only once, while for other traces it has been required two times consecutively. However, the causal relations (that is, Δ_1 can be followed by another occurrence of Δ_1 can be expressed correctly.

Table 3.2: Distinction of Change Occurrences

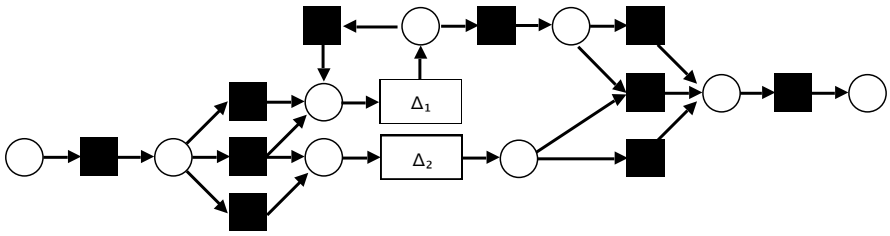
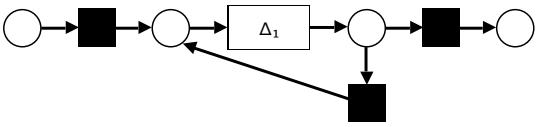
Change Log	Change Process
I1: <Δ_1> I2: <Δ_1, Δ_1> I3: <Δ_1, Δ_2>	

Table 3.3: Multiple Change Occurrences

Change Log	Change Process
I1: <Δ_1> I2: <Δ_1> I3: <Δ_1, Δ_1> --- I1: <Δ_1, Δ_1> I2: <Δ_1, Δ_1>	

Conclusion: Summarizing, the change process is a good representation for analyzing causal relationships between multiple process change operations in a change log. However, this representation proves to be ineffective when other questions arise, such as *How many change operations have evolved in a certain way?* or *What has happened after a certain change operation has been applied?*. In the next chapter, a data structure which aims at solving those issues is presented.

3.4 Filtering and Projection Techniques

In this section, we discuss filtering and projection techniques for dealing with the information overload problem ($\rightarrow$ Q3). When analyzing process change logs, e.g., by means of machine readable representations or change processes, only the information relevant to the current analysis question should be part of the change log, as discussed in the introduction. As a first step, we introduce how filtering can be applied on process change traces (Section 3.4.1), thus reducing the amount of change traces present in a change log. Next, by projecting process change traces, we shift the semantics of a change trace: While the process change trace as defined in Definition 3.1 contains all change events which describe the change operations

applied to one specific process instance, *projected change traces* contain all change events which have been applied for a specific *reason*. This is discussed in Section 3.4.2. Finally, in Section 3.4.3 we discuss problems which can arise when combining filtering and projection techniques, and show how those can be solved.

3.4.1 Filtering Change Traces

Filtering a change log results in a subset of the traces that were originally contained in the log. Filtering for an instance I is based on some filtering condition $d \odot val$ where d is a process data element and val a value written for the corresponding process instance. This information can be obtained from the execution trace of I, which is part of the corresponding execution log:

Besides change traces and change logs, execution logs also contain information important for analyzing process instances. Let $\mathcal{I}$ be a set of process instances executed based on process schema S. A corresponding *execution log* $\mathcal{X}$ consists of traces where for each $i \in \mathcal{I}$ one trace is stored in $\mathcal{X}$ (cf. Definition 3.2):

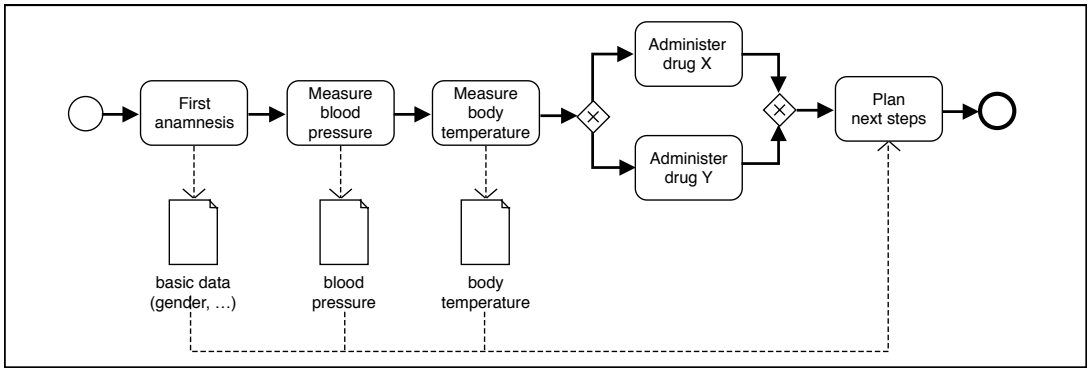
Definition 3.2 (Execution Trace, Execution Log)**.** Let $\mathcal{I}$ be a set of process instances executed based on process schema $S = (N, E, D, dEdges)$. An execution log $\mathcal{X}$ is a set of execution traces. For each $I \in \mathcal{I}$ there exists exactly one trace $t \in \mathcal{X}$. Trace t stores the temporally ordered activity executions of I, i.e. $< e_1, e_2, \ldots, e_m >$. An event e in trace t is defined as:

$e := (n, id, time, \{(d, val) | d \in D, \exists (n, d) \in dEdges$, val is the value written for d$\})$ where $n \in N$ denotes the executed activity, id a globally unique identifier for the execution event, and time denotes the point in time the event has occurred.

An example for a process execution log is given in the following:

Figure 3.3 (top pane) shows a process schema from a medical setting, which will be used as an example to explain change log filtering in the remainder of this section: The activities *First anamnesis*, *Measure blood pressure* and *Measure body temperature* generate data, which is used in the last step, where future activities will be planned. The process activities *Administer drug X* and *Administer drug Y* do not generate any data. The data is consumed by the last activity in the process, *Plan next steps*.

Figure 3.3 (bottom pane) shows how this process schema can be instantiated for multiple patients in a medical setting, and contains the corresponding execution traces: For each patient, exactly one process instance exists which contains the medical activities described above. Whenever the *Plan next steps* activity is executed, the doctor has to decide on whether and how to adapt the patient's process instance (i.e., which therapy activities to add to this patient's process instance). For each patient, there exists exactly one change trace containing the change events which have been executed for his process instance. A possible change operation



Instance	Change Trace	Execution Trace
I1:	ct1: < Δ_1 „e0", Δ_2 „e1", Δ_3 „e2", Δ_2 „e3" >	t1: < **First anamnesis** 02.04.2018 - 09:27 {**(gender, male)**, (age,65)}, **Measure blood pressure** 02.04.2018, 11:42 {(pressure, 110)}, **Measure body temperature** 03.04.2018, 10:21 {(temp, 35)}, … >
I2:	ct2: < Δ_3 „e4", Δ_2 „e5" >	t2: < **First anamnesis** 07.08.2018 - 14:32 {**(gender, female)**, {age,82)}, **Measure blood pressure** 07.08.2018, 16:12 {(pressure, 125)}, **Measure body temperature** 08.08.2018, 15:24 {(temp, 36)}, … >
I3:	ct3: < Δ_1 „e6", Δ_2 „e7", Δ_2 „e8" >	t3: < **First anamnesis** 14.08.2018 - 09:03 {**(gender, male)**, {age,91)}, **Measure blood pressure** 14.08.2018, 11:11 {(pressure, 110)}, **Measure body temperature** 15.08.2018, 11:54 {(temp, 39)}, … >
I4:	ct4: < Δ_2 „e9", Δ_2 „e10", Δ_3 „e11", Δ_3 „e12" >	t4: < **First anamnesis** 21.09.2018 - 08:43 {**(gender, female)**, {age,69)}, **Measure blood pressure** 21.09.2018, 09:12 {(pressure, 103)}, **Measure body temperature** 22.09.2018, 10:23 {(temp, 36)}, … >

Figure 3.3: Example for Change Log Filtering

of such a change event would be Δ_1 = (insert, Measure blood pressure, {Plan next steps, End}, S), which inserts the activity *Measure blood pressure* between the activity *Plan next steps* and the end of the process instance.

Hence, for each process instance, exactly one trace in an execution log exists, and exactly one trace in a change log exists. Based on this information, we define filtering for change traces as follows:

Definition 3.3 (Filtering Change Traces)**.** Let $\mathcal{C}$ be a change log, and Cond be the universe of all boolean conditions $d \odot val$ on the corresponding process execution log $\mathcal{X}$.

The function filter($\mathcal{C}$, Cond) $\mapsto \mathcal{F}$ returns the filtered set of change traces:

filter($\mathcal{C}$, cond) = $\mathcal{F}$, where $\mathcal{F} = \{ct \in C | cond$ evaluates to TRUE for the corresponding execution trace $x \in \mathcal{X}\}$

Assume that for a certain analysis question, only male patients are of inter-

est, i.e., filtering the change log returns all traces where condition `gender==male`, retrieved from the execution log, evaluates to true. We illustrate how change filtering works conceptually based on the example from Figure 3.3: When filtering, we iterate over each change trace in the change log. For each process instance a change trace belongs to, there exists exactly one execution trace which contains the history of executed activities, as displayed above for t_1, t_2, t_3 and t_4. For the chosen atomic filtering criterion `gender==male` the data for each event in the execution log is checked: For the first activity in the first trace t_1, this condition is met: the data exists, and the boolean condition evaluates to true. Since for no event in the execution trace of this instances this condition evaluates to false afterwards, the change trace is added to the set of filtered change traces. Hence, the filtered process log file turns out as $\mathcal{F} = \{c_{I_1}, c_{I_3}\}$ (gray traces in Figure 3.3). The filtering also needs to check whether the data element is present in the execution log: If not, the filtering function returns false, thus removing the corresponding change trace from the filtered change log. Thus, all patients where no gender has been set yet, would be removed from the filtered change log.

This example is based on the atomic condition `gender==male`. Beside this, more complex boolean conditions, e.g., `(gender == male)` $\wedge$ `((blood_pressure` $>$ `120)` $\vee$ `(temperature` $>$ `37))` are conceivable. For such examples, the filtering described above can be applied for each atomic condition, and the resulting sets of filtered change traces can be combined with an intersection (for $\wedge$), or a union (for $\vee$) in the correct order. In this example, the three atomic change operations could be applied as described above, resulting in the set of traces $\mathcal{F}_0 = \{c_{I_1}, c_{I_3}\}$ for `gender==male`, $\mathcal{F}_1 = \{c_{I_2}\}$ for `blood_pressure` $>$ `120` and $\mathcal{F}_2 = \{c_{I_3}\}$ for `temperature` $>$ `37`. Applying $\mathcal{F} = \mathcal{F}_0 \cap (\mathcal{F}_1 \cup \mathcal{F}_2)$, as stated by the $\wedge$ and $\vee$ operators above, results in the final set $\mathcal{F} = \{c_{I_3}\}$. In the remainder of this chapter we focus on AND connected atomic boolean conditions, such as `(gender == male)` $\wedge$ `(temperature` $<$ `37)`. While for filtering arbitrary connections of atomic conditions work as explained above, it has significant effects on the subsequent algorithms. Hence, we will present algorithms for AND connected atomic conditions, and discuss the extensions for arbitrary conditions. For other examples, values are logged several times during process execution [30]. Think about *Measure body temperature*: While it is part of the process model only once, it could be added several times when the doctor decides to do so; e.g., because the patient's body temperature was too high in the first place. This would result in several execution log events of this activity, generating different values. For such examples, query languages for analyzing process models [45, 46, 47] and execution logs [48] exist.

3.4.2 Projecting Change Logs to Cases

Filtering removes change traces from a process change log. For many situations, however, more flexibility is required. Think of process change logs in the medical setting: For each patient, there exists exactly one change trace. If however a patient has a certain problem multiple times, e.g., he runs a fever several times during his stay in the nursing home, the doctor may want to group the applied change operations not for the whole process instance (i.e., for the patient) in one trace, but for each time the patient has run a fever. Hence, a *projection* of the change operations present in a process change log to a new set of traces would be useful. As a result, a projected change trace would not contain the change events which have been applied for a certain process instance, but those applied for a certain reason, i.e., for a certain case: If one patient has run a fever three times, three projected traces would be generated, each containing all change operations which have been applied for this specific case. Further, using visualization techniques such as change trees or change processes, such projected traces could be used to analyze what has happened for one specific case.

The concept of a case has been discussed in related work before: ProCycle [8] is a system supporting users who are responsible for adapting process instances. ProCycle already defines the concept of a case, which is basically a problem description with a solution part, i.e., it couples a change operation with the reason it has been applied to a process instance. We can use this concept as a basis for the projection discussed above: For projecting change logs to such cases, we adapt the definition of a case as presented in [8] such that it contains only description of the problem:

Definition 3.4 (Case)**.** A case k is defined as a tuple $k := (id, name, desc)$ where

- id is the unique id of the case
- $name$ is the name of the case
- $desc$ is the description of the case

Example 3.5 (Case)**.** In a medical setting, e.g., a hospital, cases describe situations which require changes of the patient's therapy plans. Those situations can be anything from usual diseases such as the flu, up to very specific problems, e.g., a special type of cancer or psychological problems. An example in this context is case $k = (id, name, desc)$ with

- id = A42
- name = The flu

- desc = The patient shows symptoms of the flu. His body temperature is exceptionally high and he feels sick.

Cases can be instantiated for process instances and certain situations, e.g., a certain patient in a nursing home having the flu. For this, we introduce the *case instance* (cf. Definition 3.6). For example, a patient may have the flu several times while he stays at the hospital. Each time has to be considered individually, consequently causing a case instantiation every time.

Definition 3.6 (Case Instance)**.** Let $\mathcal{K}$ be a set of all cases and $\mathcal{I}$ be the set of all process instances. A case instance a is defined as $a := (case, instid, pinstance, t1, t2)$ where

- $case \in \mathcal{K}$
- $instid$ is the unique id of the case instance
- $pinstance \in \mathcal{I}$ is the process instance this case instance is related to
- $t1$ represents the point in time the case instance has started
- $t2$ represents the point in time the case instance has finished

$t1$ is the point in time where a case instance has been created. When a new case instance has been created, $t2$ is initially set to `null`. $t2$ is set to a value as soon as the case instance is finished, i.e., the point in time when a domain expert has declared the case instance to be finished.

Example 3.7 (Case Instance)**.** A case instance represents a case which has been instantiated for a specific process instance. If, for example, patient Timothy Smith has got the flu for the first time from April 1st until April 9th, the respective case instance could look like this:

- case = (A42, The flu, ...)
- instid = 0
- pinstance = $I47 \in \mathcal{I}$
- t1 = 01.04.2018
- t2 = 09.04.2018

In general, change operations are applied in order to react to a certain problem [8]. Thus, the application of change operations is triggered by case instances e.g., *"change Δ_n has been done because patient Smith got the flu"*. Definition 3.8 defines the corresponding mapping from change operations to their triggering case instances.

Definition 3.8 (Case Instance to Change Operation Mapping)**.** Let $\mathcal{C}$ be a change log, $\mathcal{D}$ the universe of all change events in $\mathcal{C}$, and $\mathcal{A}$ the set of all case instances. Then:

$cC : \mathcal{D} \mapsto \mathcal{A}$ maps a change event c := (Δ, id) from $\mathcal{D}$ to the triggering case instance $a \in \mathcal{A}$.

Example 3.9 (Case Instance (ctd.))**.** $\mathcal{A} = \{a_0, a_1, a_2, a_3\}$ is a case instance base with the following four case instances:

$a_0 = (A42, 0, I47, 01.04.2018, 09.04.2018)$
$a_1 = (A27, 0, I47, 13.04.2018, 21.04.2018)$
$a_2 = (A42, 0, I52, 04.12.2018, 12.12.2018)$
$a_3 = (A42, 1, I52, 07.03.2018, 13.03.2018)$

a_0 represents a case instance for case A42 (the case representing the flu), which has the instance id 0, is linked to process instance I47, has been started on 01.04.2018, and has been finished on 09.04.2018.

Further, let $\mathcal{C} = \{t_0, t_1\}$ be a change log with the following change traces:

$t_0 =< (\Delta_1, "c0"), (\Delta_2, "c1"), (\Delta_3, "c2"), (\Delta_2, "c3") >$
$t_1 =< (\Delta_1, "c4"), (\Delta_3, "c5"), (\Delta_3, "c6"), (\Delta_3, "c7") >$

The mapping of a change event in the change log to the case instance which has initiated this change event has to be provided in the data set. For example, [9] defines a `rationale` tag for the MXML format of change logs, which states the reason for a conducted change operation. Such tags can be used for storing information about the related case instance. For the given example, let the mapping from a change event $c := (\Delta, id) \in \mathcal{C}$ to a case instance $a \in \mathcal{A}$ return:

$cC((\Delta_1, "c0")) = a_0 \quad cC((\Delta_2, "c1")) = a_1$
$cC((\Delta_3, "c2")) = a_0 \quad cC((\Delta_2, "c3")) = a_1$
$cC((\Delta_1, "c4")) = a_2 \quad cC((\Delta_3, "c5")) = a_2$
$cC((\Delta_3, "c6")) = a_3 \quad cC((\Delta_3, "c7")) = a_3$

Using this information, by projecting the change traces for the given case instances, we can now see which change operations have been applied when a patient had the flu. The projected change traces can then be used as an input for visualization techniques such as change processes, or for other change mining approaches. Algorithm 1 takes as input a change log containing process change events and a set of case instances belonging to case k. Then it determines all relevant change events associated to the case instances (cf. Definition 3.8). For all distinct case

instances in this set projections onto the change events are performed which are then stored in projected traces. This is achieved by auxiliary function Π.

Algorithm 1: Generating case-based change logs

Input:

- change log $\mathcal{C}$
- set of case instances for case k: $X := \{x \in \mathcal{A} | x.case = k\}$

Output: set of change log projections $\mathcal{C}_k$ filtered for case k

```
1  Begin
2  function project(C, X)
3      Y := ∅, C_k := ∅. t_a :=<>, n := 0
4      foreach ct in C do
5          foreach c = (Δ, id) in ct do
6              a = cC(c)
7              if a ∈ X then
8                  Y:=Y ∪ {(a, c)}
9      foreach distinct y.a ∈ Y do
10         ct_a = Π(y.c, y.a) ;   // Π : 2^{D×A} ↦ 2^D where Π is a projection onto the change events
                                     assoc. to a, ordered by their original position
11         C_k := C_k ∪ {ct_a}
12     return C_k
13 project(C, X)
```

Example 3.10 (Case Instance (ctd.))**.** Generating a case-based change log according to Algorithm 1 by filtering for case A42 the auxiliary set X is built first as $X = \{(a_0, (\Delta_1, "c0")),\ (a_0, (\Delta_3, "c2")),\ (a_2, (\Delta_1, "c4")),\ (a_2, (\Delta_3, "c5")),\ (a_3, (\Delta_3, "c6")),\ (a_3, (\Delta_3, "c7"))\}$.

Projection on the distinct case instances a_0, a_2, a_3 yields the projected change traces for case A42 $\mathcal{C}_{A42} = \{t_{a_0}, t_{a_2}, t_{a_3}\}$ with

$ct_{a_0} =< (\Delta_1, "c0"), (\Delta_3, "c2") >$
$ct_{a_2} =< (\Delta_1, "c4"), (\Delta_3, "c5") >$
$ct_{a_3} =< (\Delta_3, "c6"), (\Delta_3, "c7") >$

ct_{a_0}, for example, contains all change events that have been applied for case A42, and instance I47.

In the next section, we will use information about execution events from the corresponding execution log which are related to the change events present in a projected change log, i.e., *projected execution traces* (cf. Definition 3.11):

Definition 3.11 (Projected Execution Traces)**.** Let $\mathcal{D}$ be the set of all change events $c := (\Delta, id)$ in the projected change log $\mathcal{C}_A$. Further, let $\mathcal{E}$ be the set of all execution events $x := (n, id, \ldots)$ present in the corresponding execution log. $xC : \mathcal{D} \mapsto 2^{\mathcal{E}}$ returns the set of all execution events whose corresponding activity is related to a certain process change event. Such a relation exists if the process activity causing the execution event has been affected by the change event (e.g., the activity has been moved or inserted).

For ct $\in \mathcal{C}_A, u := \cup_{c \in ct} xC(c)$

Example 3.12 (Projected Execution Trace)**.** Let $ct :=< (\Delta_1, "c0"), (\Delta_2, "c1") >$ be the change trace of process instance I, where Δ_1 = (`insert`, "check body temperature", {AND-SPLIT X, AND-JOIN Y}, S) adds an additional check of the patient's body temperature to his process instance, and Δ_2 = (`insert`, "administer drug y", {AND-SPLIT X, AND-JOIN Y}, S). Let t be the execution trace for I as follows:

$t_1 =<$ `(administer drug x, "e0", 02.04.2018 - 08:00, ∅),`
`(administer drug y, "e1", 02.04.2018 - 09:27, ∅),`
`(check body temperature, "e2", 04.04.2018 - 09:27, {(temp, 36.5)})`

Here, execution event `e2` has been affected by change event `c0`, since the corresponding activity has been inserted with this change event. If a projected change trace for a case instance only contains this change event, i.e., $ct_{a_0} =< (\Delta_1, "c0") >$, the corresponding projected execution trace would be as follows:

$u_{a_1} =<$`(check body temperature, "e2", 04.04.2018 - 09:27, {(temp, 36.5)})>`

For some situations, execution log data can also be useful for the projection of change traces: Algorithm 1 returns the projected change traces containing all change operations which can be directly linked to one of the case instances of interest. For other examples however, there may be interactions between change operations which have to be considered: Imagine a change operation (belonging to case A) adding a process fragment, and a second change operation (belonging to case B) removing or adapting it. Such a trace of change operations creates dependencies between case instances of different cases, which have to be addressed. A practical example can be found in the medical domain: If we added some sport activity to a patient's therapy process (e.g., as part of his physiotherapy (case A)), which has to be removed due to the patient catching the flu (case B), and being unable to conduct his sports activity, this deletion would have to be part of the physiotherapy case instance (case A) as well, since it affected this case instance. Another type of interaction between change operations stems from possible semantic interactions between conducted activities, which have been affected by process

change operations belonging to different case instances: If a change operation for a case instance of the case *flu* (case A) has added a certain drug to a patient's therapy process, this drug may have an effect on other case instances as well, which run in parallel (e.g., the patient having problems with his digestion system, case B). For such situations, we do not want to project only those change operations which belong to the case instance of interest, but *all change operations which have generated execution log entries while a case instance of the case of interest has been active*. Hence, for such settings, the case projection algorithm presented above has to be adapted, as shown in Algorithm 2. It works as follows: In addition to adding all change operations which can be linked to the chosen case instance, it also iterates over *all other execution events which have been conducted while the case was active* (i.e., in the time between t1 and t2 of the corresponding case instance). If an execution event has happened during that time, it may be possible that it also had an effect on the respective case instance, even if it was not a part of it. Additionally, all change operations which delete process fragments have to be added to the projected change traces, since the fragments they removed may have had an influence on the overall outcome (since these fragments may have been executed during the time of interest). Thus, the corresponding change operations (which have affected the execution events) have to be added to the projected change trace as well.

The choice of the `project` function depends on the data set and the analysis question at hand: If the data set contains change operations belonging to different case instances, which may have an impact on the parameters of interest, the project function as presented in Algorithm 2 is suited. If there are no dependencies, and the amount of data in the projected change traces should be kept to a minimum, the project function presented in Algorithm 1 may be preferred.

3.4.3 Filtering Case-based Change Traces

For some analysis questions, a combination of filtering and projection becomes necessary. Imagine a product manager analyzing production processes in a manufacturing environment: Two process instances I_0 and I_1 exist, which describe the production of two products. Due to several possible problems during production, different change operations might become necessary while the respective process instance is being executed. Hence, the following two change traces, ct_0 for I_0 and ct_1 for I_1, in change log $\mathcal{C}$ exist:

$ct_0 =< (\Delta_1, "c0"), (\Delta_2, "c1"), (\Delta_3, "c2"), (\Delta_2, "c3"), (\Delta_4, "c4"), (\Delta_4, "c5") >$
$ct_1 =< (\Delta_3, "c6"), (\Delta_2, "c7"), (\Delta_5, "c8") >$

For the sake of simplicity, each change operation in the change events inserts

Algorithm 2: Projecting change logs with dependencies

Input:

- change log $\mathcal{C}$
- set of case instances for case k: $X := \{x \in \mathcal{A} | x.case = k\}$

Output: set of change log projections $\mathcal{C}_k$ filtered for case k, including possible dependencies

```
1  Begin
2  function project(C, X)
3      Y := ∅, C_k := ∅. t_a :=<>, n := 0
4      foreach ct in C do
5          foreach c = (Δ, id) in ct do
6              a = cC(c)
7              if Δ.type == "delete" then
8                  Y:=Y ∪ {(a, c)}
9              else
10                 //check if some activities related to this change operation have been executed while a
                     case instance from Y has been active
11                 //get-execution-events returns a set of all execution events related to change
                     operation Δ at position pos
12                 execution-events = xC(c)
13                 foreach x ∈ execution-events do
14                     foreach k ∈ X do
15                         if x.time ≥ k.t1 ∧ x.time ≤ k.t2 then
16                             Y:=Y ∪ {(a, c)}
17                 if a ∈ X then
18                     Y:=Y ∪ {(a, c)}
19     foreach distinct y.a ∈ Y do
20         t_a = Π(y.c, y.a) ;     // Π : 2^(D×A) ↦ 2^D where Π is a projection onto the change events
             assoc. to a ordered by their original position
21         C_k := C_k ∪ {ct_a}
22     return C_k
23 project(C, X)
```

exactly one activity as a parallel branch into the affected process instance. However, the presented technique can be applied to larger process fragments, as well as other change operation types as well. The change operations $\Delta_1, \Delta_2, \Delta_3$ and Δ_4 from the change events are defined as follows:

$\Delta_1 = (\texttt{insert}, \text{“switch to supplier 2”}, \{\text{AND-SPLIT X, AND-JOIN Y}\}, S)$
$\Delta_2 = (\texttt{insert}, \text{“adapt production plan”}, \{\text{AND-SPLIT X, AND-JOIN Y}\}, S)$
$\Delta_3 = (\texttt{insert}, \text{“switch to supplier 3”}, \{\text{AND-SPLIT X, AND-JOIN Y}\}, S)$
$\Delta_4 = (\texttt{insert}, \text{“switch production machine”}, \{\text{AND-SPLIT X, AND-JOIN Y}\}, S)$
$\Delta_5 = (\texttt{insert}, \text{“notify customer”}, \{\text{AND-SPLIT X, AND-JOIN Y}\}, S)$

The cases for this example are defined as follows:

$c_0 =$ (0, “supplier change”, “a supplier was changed, e.g., due to production problems on his side”)
$c_1 =$ (1, “machine change”, “a production machine was changed, e.g., due to necessary maintenance”)

For these two cases, a total of five case instances in the set of case instances $\mathcal{A}$ exist (a_0, a_1, a_3 for case c_0, a_2, a_4 for case c_1):

$\mathbf{a_0} = (\mathbf{c_0}, \mathbf{0}, \mathbf{I_1}, \mathbf{02.04.2018}, \mathbf{04.04.2018})$
$\mathbf{a_1} = (\mathbf{c_0}, \mathbf{1}, \mathbf{I_1}, \mathbf{05.07.2018}, \mathbf{14.07.2018})$
$a_2 = (c_1, 0, I_1, 05.08.2018, 05.08.2018)$
$\mathbf{a_3} = (\mathbf{c_0}, \mathbf{0}, \mathbf{I_2}, \mathbf{04.02.2018}, \mathbf{12.02.2018})$
$a_4 = (c_1, 0, I_2, 23.03.2018, 23.03.2018)$

The mapping from a change event $c := (\Delta, id) \in \mathcal{C}$ to a case instance $a \in \mathcal{A}$ returns the following:

$\mathbf{cC}((\mathbf{\Delta_1}, \text{”}\mathbf{c0}\text{”})) = \mathbf{a_0}$ $\quad$ $\mathbf{cC}((\mathbf{\Delta_2}, \text{”}\mathbf{c1}\text{”})) = \mathbf{a_0}$ $\quad$ $\mathbf{cC}((\mathbf{\Delta_3}, \text{”}\mathbf{c2}\text{”})) = \mathbf{a_1}$
$\mathbf{cC}((\mathbf{\Delta_2}, \text{”}\mathbf{c3}\text{”})) = \mathbf{a_1}$ $\quad$ $cC((\Delta_4, \text{”}c4\text{”})) = a_2$ $\quad$ $cC((\Delta_4, \text{”}c5\text{”})) = a_2$
$\mathbf{cC}((\mathbf{\Delta_3}, \text{”}\mathbf{c6}\text{”})) = \mathbf{a_3}$ $\quad$ $\mathbf{cC}((\mathbf{\Delta_2}, \text{”}\mathbf{c7}\text{”})) = \mathbf{a_3}$ $\quad$ $cC((\Delta_5, \text{”}c8\text{”})) = a_4$

Let t_1 and t_2 be the execution traces for I_1 and I_2 as follows:

$t_1 =<$ `(set product category, "e0", 02.04.2018 - 08:00, {(product_category, 42)}),`
`(switch to supplier 2, "e1", 02.04.2018 - 09:27, {(delivery_time, 9)}),`
`(adapt production plan, "e2", 04.04.2018 - 09:27, ∅),`
`(switch to supplier 3, "e3", 05.07.2018 - 11:31, {(delivery_time, 2)}),`

(adapt production plan, "e4", 14.07.2018 - 13:46, ∅),
(switch production machine, "e5", 05.07.2018 - 19:11, ∅),
(switch production machine, "e6", 05.08.2018 - 17:10, ∅)>
t_2 =< (set product category, "e7", 04.02.2018 - 09:00, {(product_category, 42)}),
(switch to supplier 3, "e8", 04.02.2018 - 11:31, {(delivery_time, 1)}),
(adapt production plan, "e9", 11.02.2018 - 09:27, ∅),
(notify customer, "e10", 12.02.2018 - 11:31, ∅)>

Given this data set, the analyst may be interested in all change operations which have been applied for case instance c_0, where the filtering condition `(delivery_time <= 3 days)` $\wedge$ `(product_category = 42)` holds. Such an analysis may be relevant in order to decide which supplier should be chosen, if a short delivery time for a certain type of product is necessary. For extracting this information, a combination of the filtering and projection techniques has to be developed: While the projection creates the projected change traces for case c_0, the filtering removes all those projected traces where the corresponding execution log data does not match the filtering criteria. Relevant information can be deduced directly from the process change and execution logs. The activities `switch to supplier 2` (added by change operation Δ_1) and `switch to supplier 3` (added by change operation Δ_3) write the delivery time of the chosen supplier in the execution log, and activity `set product category` writes the product category.

After projection (using Algorithm 1, since no interactions between case instances are relevant for this example), each projected trace contains all change operations for case instances of case c_0. This leads to the following projected change traces (relevant information is depicted in bold above):

$ct_{a0} =< (\Delta_1, "c0"), (\Delta_2, "c1") >$
$ct_{a1} =< (\Delta_3, "c2"), (\Delta_2, "c3") >$
$ct_{a3} =< (\Delta_3, "c6"), (\Delta_2, "c7") >$

The projected execution traces for the three projected change traces ct_{a0}, ct_{a1}, and ct_{a3} can be derived from the execution log as described in Section 3.4.2. Here, all events which have been added by the change operations are part of the respective projected execution traces. However, the activity `set product category`, which defines the category of the produced product, has not been affected by any change operation, and thus is not part of any projected execution trace, which are derived from the given data as follows:

u_{a_0} =<
`(switch to supplier 2, 02.04.2018 - 09:27, {(delivery_time, 9)}),`
`(adapt production plan, 04.04.2018 - 09:27, ∅)>`
u_{a_1} =<
`(switch to supplier 3, 05.07.2018 - 11:31, {(delivery_time, 2)}),`
`(adapt production plan, 14.07.2018 - 13:46, ∅)>`
u_{a_3} =<
`(switch to supplier 3, 04.02.2018 - 11:31, {(delivery_time, 1)}),`
`(adapt production plan, 11.02.2018 - 09:27, ∅)>`

Next, the projected change traces have to be filtered according to the filtering condition `(delivery_time <= 3 days)` $\wedge$ `(product_category == 42)`. When filtering projected change traces for data present in the projected execution traces u_{a_0}, u_{a_1} or u_{a_3}, only the execution events present in those projected execution traces should be used for checking the filtering conditions for the projected change traces ct_{a0}, ct_{a1} and ct_{a3}. If the non-projected execution traces t_1 or t_2 were used instead, information belonging to different case instances would be contained in one of those non projected execution traces. For the example given above, trace t_1 contains information from the two projected execution traces u_{a_0} and u_{a_1}. When filtering for `(delivery_time <= 3 days)`, this condition would evaluate to true for t_1 (thus adding both ct_{a0} and ct_{a1} to the set of filtered traces), although it would evaluate to false for u_{a_0}, and to true for u_{a_1} (adding only ct_{a1} to the filtered trace set). Hence, data present in the projected execution traces has to be filtered *after* projection.

However, since the projected execution traces contain only a subset of the events present in the original execution trace, some necessary information might be missing. For the example given above, the activity `set product category`, which writes the `product_category` to the execution log is not part of any projected execution trace. Thus, when filtering for the condition `(product_category == 42)` using the projected execution traces u_{a_0}, u_{a_1} or u_{a_3}, this condition would evaluate to false, since the required data is not present. Hence, for data which is only present in the non projected execution trace, these traces t_1 and t_2 have to be used instead.

Under the assumption of AND connected atomic conditions one solution is to split them into atomic conditions. Then it is determined which atomic condition can be evaluated based on the projected execution log, and which require data only present in the non projected execution log. Algorithm 3 describes this split of conditions, and the application of the filters before and after projection.

For the given example, the overall boolean condition is `(product_category == 42)` $\wedge$ `(delivery_time <= 3 days)`. Algorithm 3 splits this in two distinct

atomic boolean conditions: The first condition `(product_category == 42)` can only be applied to the non projected execution traces, since the respective event has not been affected by any change operation. The second condition `(delivery_time <= 3 days)` has to be applied to the projected change traces.

This results in the following projected and filtered change traces:

$ct_{a1} =< (\Delta_3, "c2"), (\Delta_2, "c3") >$
$ct_{a3} =< (\Delta_3, "c6"), (\Delta_2, "c7") >$

We have presented an approach which combines filtering and projection based upon boolean conditions connected with the AND operation. However, other operations stemming from boolean algebra, including the binary $\vee$, or the unary *not* operation, can be implemented as well by splitting the condition into multiple AND connected or atomic conditions. For example, the boolean condition `((gender == male)` $\wedge$ `(blood_pressure > 120))` $\vee$ `(temp > 37)` can be split in two distinct AND connected or atomic conditions, i.e., `(gender == male)` $\wedge$ `((blood_pressure > 120)` and `(temp > 37)`. Following, the presented approach can be applied to each of these parts, thus returning a set of change traces each. As described in [49], boolean algebraic AND and OR operations, and the unary *not* operation, *"have, roughly speaking, the same properties as the set-theoretical union, intersection and complementation"* [49]. Hence, by using the equivalent set theoretical counterpart, these sets of traces then can be combined, thus creating the overall result. In the given example, the two conditions are connected with a $\vee$ operation; thus, the two resulting sets have to be unified for retrieving the correct result.

3.5 Discussion

In this chapter, we have discussed different representation and analysis methods for process change traces and change logs. Representing process change traces in a machine readable format, i.e., MXML or XES, provides the basis for interacting with process mining tools such as ProM. We have analyzed an existing representation in MXML, and presented a possible mapping of the information usually present in a process change log to its successor XES.

Next, we have introduced change processes, a representation for change logs which focuses on the causal dependencies between executed process change operations. Based on this representation, we have shown several shortcomings, e.g., when multiple change traces contain the same or similar kinds of process change operations. In order to solve these shortcomings, in the next chapter we present the change tree, a representation method which focuses on the temporal relationships

between executed process change operations in change logs.

Using information from the corresponding process execution log, we have shown an algorithm for filtering change traces which fit a certain (set of) criteria from the change log. This filtering can be used as a basis for many forms of analysis: For a hospital setting, a change log can be constrained to only contain those process change traces, which fit a certain set of patients. If we want to know what has been done to the therapy process instances to all male patients above age 85 with diabetes, such a filtering technique can be used as a basis. In a production environment, process change logs can be filtered to contain only those instances, where a certain problem has occurred, or which were started during a certain period of time. Thus, one can compare the differences between production instances which have been conducted during the Christmas holidays (where possibly less human resources were available) and those where all workers were at the factory.

By introducing the concept of *cases*, process change logs can be projected in a way that its traces contain all change operations which belong to a certain case instance. This is also relevant for practical settings which deal with process change operations on a regular basis: In the nursing domain, one can project change operations to illnesses, e.g., one can see which change operations have been applied because a patient has caught the cold. In the manufacturing environment, one can see which change operations have been applied because a certain problem occurred (e.g., a machine was out of order).

In combination with filtering, powerful analysis methods emerge: In a medical setting, doctors can analyze what has been changed for process instances when a male patient above age 85 (filtering step) had the flu (case projection step). In a manufacturing environment, one can analyze all change traces which have been applied because a certain machine was out of order (case projection) for a certain kind of product (filtering). Note that the ordering of the case projection and the filtering steps can be interchanged.

A technical evaluation for filtering and the case projection of process change logs is shown in the prototypical implementation of this thesis (Chapter 5).

Algorithm 3: Projecting and filtering change logs

Input:

- change log $\mathcal{C}$
- execution log $\mathcal{X}$
- set of case instances for case k: $X := \{x \in \mathcal{A} | x.case = k\}$
- filter = $\{(d_{1,1} \odot val_{1,1}), (d_{1,2} \odot val_{1,2}), \ldots, (d_{1,n} \odot val_{1,n})\}$, a set of AND connected boolean conditions where each d $\odot$ val operates on a data element d from the execution log

Output: filtered and projected change log $\mathcal{F}$

```
1  Begin
2  //Use projection method presented in Algorithm 1
3  C_k = project(C, X)
4  f-total = ∅
5  f-projected = ∅
6  //find all conditions where the projected execution log contains the required data
7  foreach f = (d ⊙ val) ∈ filter do
8      f-total = f-total ∪ f
9      foreach t ∈ C_k do
10         pos = 0
11         foreach Δ ∈ t do
12             //xC returns all execution log events from X affected by a Δ
13             executionlog-events = xC(Δ, pos)
14             foreach e ∈ executionlog-events do
15                 if e.d contains d then
16                     f-projected = f-projected ∪ f
17             pos++
18 //f-nonprojected contains all filter conditions for data only present in the nonprojected execution traces
19 f-nonprojected = f-total - f-projected
20 F = C
21 //Apply the filters before projection
22 foreach f ∈ f-nonprojected do
23     //let filter(condition, changelog) be a function that returns a set of all filtered traces
24     F = filter(f, F)
25 //Project the filtered change traces
26 F = project(F, X)
27 //Apply the filters after projection
28 foreach f ∈ f-projected do
29     F = filter(f, F)
30 return F
```

Chapter 4

The Change Tree

This chapter introduces the change tree, a novel representation method for process change traces. Figure 4.1 positions this chapter in the structure of the thesis: After presenting single process change operations in Chapter 2, we have discussed representation and analysis methods for change traces in Chapter 3. However, when analyzing existing representation methods, we found several shortcomings, particularly when addressing the chronological order of change operations in process change traces. Hence, in this chapter, we present the change tree as a new representation method for change logs. This chapter is based upon the papers *"Mining and Querying Process Change Information Based on Change Trees"* [37] and *"Improving the Usability of Process Change Trees Based on Change Similarity Measures"* [50].

4.1 Overview

In the last chapter, change processes [9] have been discussed as a state of the art representation for change logs. While change processes correctly capture the causality relations between the change operations applied in the change traces, several shortcomings of this representation have been found (cf. Chapter 3.3).

Summarizing, the following questions cannot be answered when consulting the change process:

Q1: Which process instances have evolved in a similar way?

Q2: Which process instances have evolved in a similar way after a certain sequence of change operations?

The first two questions are directly relevant in practical settings. Analyzing which process instances have evolved in a similar way can be used as a starting

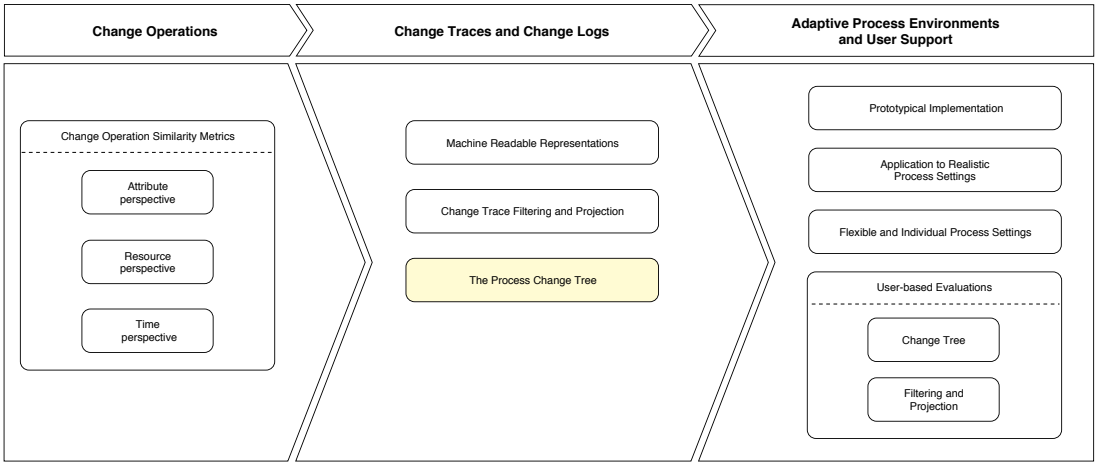


Figure 4.1: Dissertation Overview: Change Trees

point for predicting changes which may become necessary in the future. In the care domain, for example, Q1 can be used to analyze patients' treatment histories in order to predict future changes. Imagine two patients which have received the same treatments over a period of time, whereby the first patient's treatment plan is already further developed than the treatment plan of the second patient. When the question arises which changes may be necessary for the second patient's treatment plan in the future, the change information from the first patient's treatment plan can be used as a starting point.

Such examples can be found in different domains: In the manufacturing domain, data from production processes which have evolved in a similar way can be used to predict possibly required adaptations for other, similarly evolving, process instances.

Q2 provides more focus by asking for, e.g., the development of a treatment after a certain therapy was applied. This information can be used as a basis for analyzing what usually happens after a certain set of changes has been applied. Imagine a therapy in the nursing home setting which requires the nurses to conduct multiple other therapies afterwards, maybe because of possible complications. The question of interest is now, how many treatment plans have evolved in which way after this certain therapy has been conducted. This information can be used to further enhance the planning and conduction of therapies.

Similarly, for the manufacturing domain, such questions arise: When the supplier of a certain required resource needs to be replaced, e.g., due to not being able to deliver the required quantities on time, it would be interesting to see the consequences of such a decision on the whole process instance. For example, it would be possible that based on the new resources, additional tests for the resulting product

may be required. While a change log contains such information, it cannot easily be represented using state of the art approaches, such as change processes.

The question is now, how change information of process instances can be represented such that Q1 and Q2 can be sufficiently analyzed. As both Q1 and Q2 refer to the process instance level and as we assume that changes might be applied multiple times (think, for example, of steps such as physical therapy that might become necessary multiple times), a suitable representation must meet the following requirements:

R1: ability to deal with multiple occurrences of (change) instances (Q1, Q2)

R2: ability to deal with multiple occurrences of changes (Q1, Q2)

R3: ability to detect change sequences that follow a certain change pattern (Q2)

As discussed in Section 3.3, existing approaches such as change processes do not fully meet these requirements. Hence, in this chapter we first introduce two new representations for information stored in change logs. At first, the change tree is defined in Section 4.2 as a basis to meet requirements R1 and R2. In addition to the formal definition an algorithm is presented that mines change trees from change logs. Section 4.3 compares the change tree to other representations such as change processes and graphs. Next, Section 4.4 introduces the n-gram change tree as a further development of the change tree meeting requirement R3. Based on representing change sequences as n-grams, the n-gram change tree offers a projection on those parts of the change traces that evolved after the occurrence of a change sequence of interest, i.e., the respective n-gram. It is shown how n-gram change trees can be constructed from change logs.

Overall, change tree and n-gram change tree are novel change log representations that meet requirements R1, R2, and R3 and hence enable the in-depth analysis of highly dynamic process applications.

While the (n-gram) change tree as a representation for process change logs fulfills requirements R1 - R3, it would be interesting if other analysis tools for change traces and process change operations can be included as well. This leads to the following third research question for this chapter:

Q3: Can we incorporate analysis techniques, such as filtering, projection, or change operation similarity, into change trees? Which benefits would arise?

In previous chapters, we have discussed different analysis methods both for single change operations, as well as for change traces. In Section 4.6 and 4.7, we analyze if and how these techniques can be incorporated into a representation

method such as the change tree which meets requirements R1 - R3, and which benefits arise.

Section 4.8 concludes the chapter. A proof-of-concept implementation is presented in Chapter 5. Application of the presented concepts to various domains, including the financial, wellbeing and nursing domain, are presented in Chapter 6.

4.2 Change Trees

In order to be able to answer Q1 and meet requirements R1 and R2 (cf. Section 4.1), a representation for change information shall represent the chronological order of changes made to all instances in one aggregated view. Definition 4.1 presents the change tree as a representation for instance change information. Intuitively the change tree represents each of the change traces in a change log along with the number of its occurrences along paths from the root to the leafs.

Definition 4.1 (Change Tree)**.** Let $\mathcal{C}$ be a change log and $\mathcal{D}$ be the change operations contained in $\mathcal{C}$. Then change tree T is defined as a rooted multiway tree T:=(r,V,E) with

1. $r := \emptyset$ is the unique root node
2. V $\subseteq \mathcal{D} \times \mathbb{N}_0$
3. E $\subseteq V \times V$
4. $\forall$ leaf nodes v = (Δ,n) $\in$ V: n $>$ 0
5. $\forall$ paths p from root r to node v = (Δ, n) $\in$ V with n $>$ 0: p corresponds to the change traces of n changed process instances in $\mathcal{C}$

Let us illustrate the concept of the change tree along the example depicted in Figure 4.2. Figure 4.2 shows a change log (left side) and its representation as a change tree (right side)[1]. The change log for process instance I1 for example consists of two consecutive applications of change Δ_1 (*R2*: multiple occurrence of changes). Starting from the root node and going towards *Leaf 2* in the change tree, the change trace can be reproduced. The number *1x* in Leaf 2 indicates that this change trace, i.e., $< \Delta_1, \Delta_1 >$, occurred once in the change log. With this, the change tree offers the possibility to count the number of multiple instance occurrences.

Overall, the change tree representation covers all instances in the log as well as maintains the number of occurrences of change traces (*R1*) and change occurrences

[1]For illustration reasons the change tree is annotated with the number of leafs.

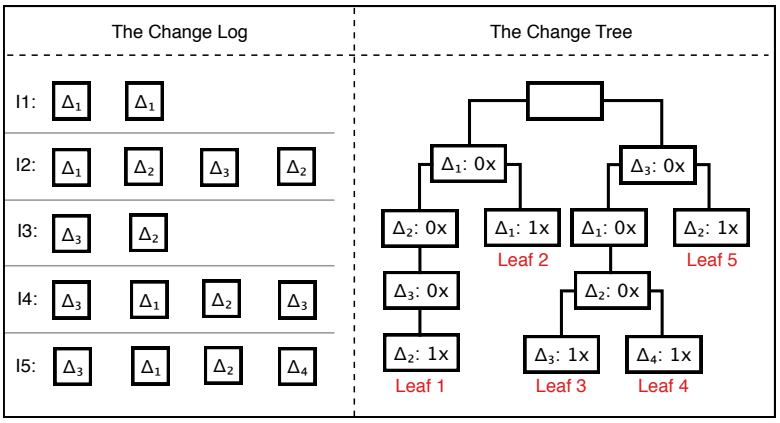


Figure 4.2: Change Log and Corresponding Change Tree

(*R2*). With respect to question *Q1* as set out in the introduction, the change tree offers the possibility to determine how many instances have evolved in which way. This information can be important for predicting and planning future changes.

Algorithm 4 sets out how a change tree can be created based on a change log. The algorithm starts with the first set of changes which has been applied to the process instances in the change log (line 30). For instances I1 and I2 from Figure 4.2 this is Δ_1, for I3, I4 and I5 this is Δ_3 (the first row of changes). For each instance in the given set of instances (for the root node this are all instances in the change log) the change at the current position is checked. If a node containing this change does not already exist a new node is generated. For the first iteration two nodes containing Δ_1 and Δ_3 are generated. Next we check if the change we just created the node for is the last change for this process instance or not. If it is, we increment the counter for this node. This means that the instance is finished, we have reached leaf level and we have found one more instance with the given set of changes. If it is not the last instance we add the current instance to the set of instances for this node which will be traversed in later iterations. The set of instances is used to generate the child nodes for our current node, and since there is a following change, there will be a child node. Now we recursively reuse this function to generate all child nodes for each node in our set of nodes. After the first iteration this means that the function is called for the nodes Δ_1 (with instances I1 and I2) and Δ_3 (with instances I3, I4 and I5). Note that the iterator is incremented, thus analyzing the second level of changes (instances I1, I4 and I5: Δ_1, instances I2 and I3: Δ_2).

The structure of the change tree emphasizes multiple features of the change log it is based on: The *breadth of the change tree* is an indicator for the diversity of the applied process change operations. The broader the change tree, the more

Algorithm 4: Create a change tree from a change log

```
Input: Change log C
//parent is the parent node for the children which are inserted
//set-of-instances is the set of instances which contains data for child nodes of the current parent node
//iterator is the number of the change in the change log instance - iterator=2 means the 2nd change
  applied to the instance
function create-children(parent,set-of-instances,iterator)
    set-of-nodes = new Array
    foreach instance in set-of-instances do
        if there is no child node with the current change for this parent then
            node = create-node(instance_iterator,parent)
            set-of-nodes.add(node)
        else
            node = the child node of the parent containing the current change
        if instance_iterator+1==empty then
            node.count++
        else
            node.nextinstances.add(instance)
    foreach node in set-of-nodes do
        create-children(node,node.nextinstances,iterator+1)
    return;
function create-node(label,parent)
    node.label = label
    node.count = 0
    node.nextinstances = new Array
    node.children = new Array
    if parent then
        parent.children.add(node)
    return node
Begin
root = create-node(null,null);
create-children(root, C, 0);
```

diverse the change operations have been. The *height of the change tree* indicates how many change operations have been applied to one or more change traces. The number of change traces in a certain branch in relation to the total number of change traces in the log shows which change operation paths have been used more frequently. The diversity of the change operations which have been applied on a certain level can be seen from the number of *sibling nodes*.

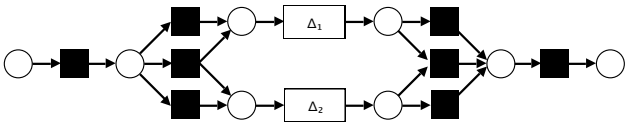
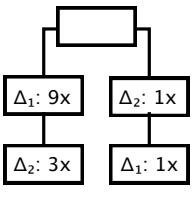
4.3 Comparison with Other Representations

This section compares the change tree to other representations such as the change process [9] and graph-based structures. In Section 3.3.2, we have discussed some examples for change logs which have been represented as change processes. We have shown that for those examples, the representation as a change process is either unclear or incomplete. In this section, we show how these problems can be solved using change trees.

First consider the scenario depicted in Table 4.1. On the left side the change log is depicted, in the middle the resulting change process after applying change mining, and on the right side the change tree.

For the scenario in Table 4.1 the change process does not convey the information that for 9 instances only change operation Δ_1 has been applied while only change Δ_2 has only been applied for one instance. Moreover, based on the OR-split close to the start of the process, it is not possible to see that for 3 instances first Δ_1 and then Δ_2 has been applied while for one scenario the reverse order of change operations occurred. The reason for this limitation is that change processes abstract from the number of instance occurrences. In contrast to this the change tree reflects multiple occurrences of change traces, thus fulfilling requirement *R1*. All possible combinations of changes as they can be found in the change logs can be easily detected and interpreted. For example, by using the change tree representation, one can immediately see that Δ_1 has been used in a lot more cases than Δ_2.

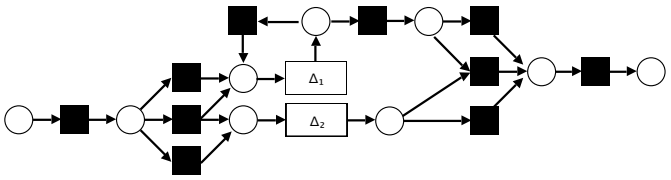
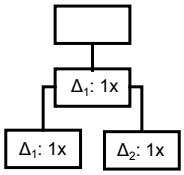
Table 4.1: Multiple Instance Occurrences

Change Log	**Change Process**	**Change Tree**
I01-I09: <Δ_1> I10: <Δ_2> I11-I13: <Δ_1, Δ_2> I14: <Δ_2, Δ_1>		

In the second scenario (cf. Table 4.2) the change tree provides a much clearer

and shorter representation than the complex change process. In this example, change operation Δ_1 has been applied two times in one change trace, and only once in two other traces. Because of the trace where it has been applied two times, it is represented as a loop in the resulting change process. This leads to the problem that it is not possible to see anymore, how many times Δ_1 has been applied. Using the change process, it is impossible to see whether it has been applied, two, three or ten times, and for how many traces this has been done. Additionally, one cannot see, for how many process instances Δ_2 has been applied. Using the change process, one cannot see whether Δ_2 is a part of most change traces, or just of one exceptional trace. These problems are solved by the change tree: Here, it can clearly be seen that (a) three process instances are part of the change log, and (b) which change operation has been applied for how many traces.

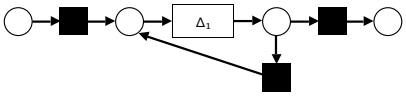
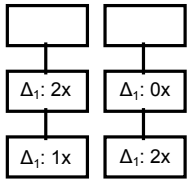
Table 4.2: Distinction of Change Occurrences

Change Log	**Change Process**	**Change Tree**
I1: <Δ_1> I2: <Δ_1, Δ_1> I3: <Δ_1, Δ_2>		

The problem that multiple occurrences of the same change cannot be reflected in the change process is aggravated in the third scenario shown in Table 4.3. Here, different process scenarios still produce the same change process: The left column shows two possible change logs, which both result in the same change process representation. Since for the first change log, in one out of three traces Δ_1 has been applied two times, a loop has been created in the resulting change process. Thus, the two other traces where Δ_1 has only been applied once cannot be distinguished in this representation any more. The corresponding change trees clearly show a distinction between the instances where Δ_1 has been applied once or multiple times, thus fulfilling requirement *R2*.

In some cases, using the change tree as a representation for process change logs leads to a great amount of repetition. Consider the change tree in Figure 4.3 (left pane) as an example: After applying change operation Δ_1 *or* Δ_2, the remaining change operations of both branches of the tree consist of the same change operations: In both branches, Δ_3 is followed by Δ_1, respectively Δ_2. Thus, these two branches contain the same information regarding the applied change

Table 4.3: Multiple Change Occurrences

Change Log	Change Process	Change Trees
I1: <Δ_1> I2: <Δ_1> I3: <Δ_1, Δ_1> --- I1: <Δ_1, Δ_1> I2: <Δ_1, Δ_1>		

operations, and could be aggregated. One way to do this can be seen in the right panel of Figure 4.3, where the graph structure represents the same change log. As can be seen immediately, the repetition of the branches starting with Δ_3 has been replaced with two edges leading into the same Δ_3 node.

Using information provided by the nodes alone, one could not see anymore, how many change traces have evolved in a certain direction. For example, it would not be clear anymore, that six change traces have started with Δ_1, while 13 traces have started with Δ_2. To compensate this information loss, this information has to be added as annotations to the edges of the graph. In the change tree, this information is available just by inspecting the nodes.

However, some information is still lost due to the representation as a change graph: As can be discovered easily in the change tree, after applying change operation Δ_1, only one instance stopped its evolution at Δ_3 while 2 instances evolved further to Δ_1 and one instance evolved to Δ_2. However, this information is lost in the graph representation since the two Δ_3 nodes are merged.

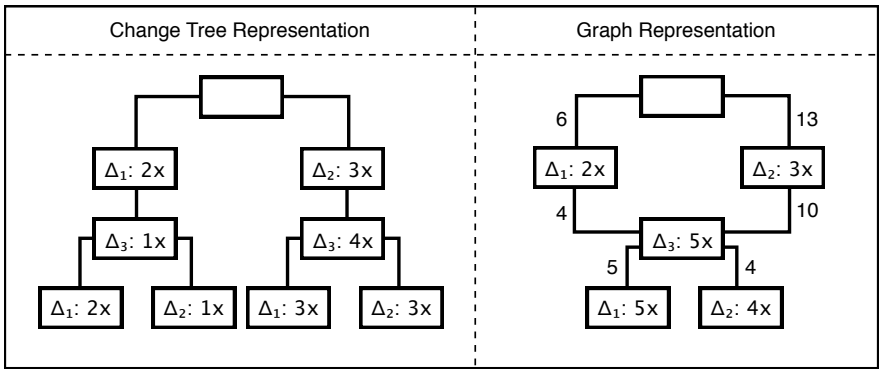


Figure 4.3: Comparing tree- and graph based models

Thus, a graph-based representation of change trees as presented in this section is a possible approach to reduce the number of nodes in change trees. However, as the example given in this section has shown, depending on the structure of the

change traces, information loss is still possible. In Section 4.7 we present a method to aggregate (and thus decrease the number of) nodes in a change tree which does not lead to such information loss of the resulting representation.

4.4 n-gram Change Trees

In order to answer Q1 (cf. Section 4.1), the change tree represents the chronological order of changes made to all instances in one aggregated view. In order to answer Q2, in addition, all occurrences of a specific change trace must be detected and "consolidated" in the resulting representation in order to analyze what happened after the change trace to the process instances of interest.

A representation which fulfills this third requirement R3 (cf. Section 4.1) can be used to answer questions such as "Which changes have occurred in the past following a change or a set of changes" or "How many instances have evolved in a certain way after a change has been applied?".

A change trace as defined in Definition 3.1 can also be understood as a string and a subset of such a trace containing change operations as a substring. Thus, the problem can be considered as a transformation of the problem of *n-gram models* in language processing where two strings are defined as equivalent *"if they end in the same n - 1 words"* [51]. The example shown in Figure 4.4 for example contains five strings (I1: $(\Delta_1\Delta_1)$, I2: $(\Delta_1\Delta_2\Delta_3\Delta_2)$ etc.). Hence, the *n-gram* in the change setting will reflect the change pattern (substring) of interest and we will determine the sequences of change operations following the change pattern for each of its occurrences in the log.

Definition 4.2 specifies a sequence of change operations which will be denoted as n-gram in the following:

Definition 4.2 (n-gram)**.** Let $\mathcal{C}$ be a change log and $\mathcal{D}$ be the set of changes contained in $\mathcal{C}$. Then a n-gram g over $\mathcal{C}$ is defined as a vector of changes contained in $\mathcal{C}$ i.e. g:=$< \Delta_1,...,\Delta_l >$ where $\Delta_i \in \mathcal{C}$, i = 1,...,l.

Assume that we are interested in the n-gram $< \Delta_1, \Delta_2 >$ and consider the example depicted in Figure 4.4. For instance I2, the n-gram can be found at the beginning whereas for instances I4 and I5 the n-gram occurs after Δ_3. This results in a change tree where the required sequence of changes can be found in two different branches of the change tree (red nodes in the middle pane).

To see all changes following n-gram $< \Delta_1, \Delta_2 >$, the two subtrees containing the n-gram have to be combined, i.e., restructuring the change tree becomes necessary. This can be achieved by using a suffix tree [52] or, more precisely, a generalized suffix tree which contains more than one string. A suffix tree represents all suffixes for a given string. Suffix trees are commonly used for pattern matching in various

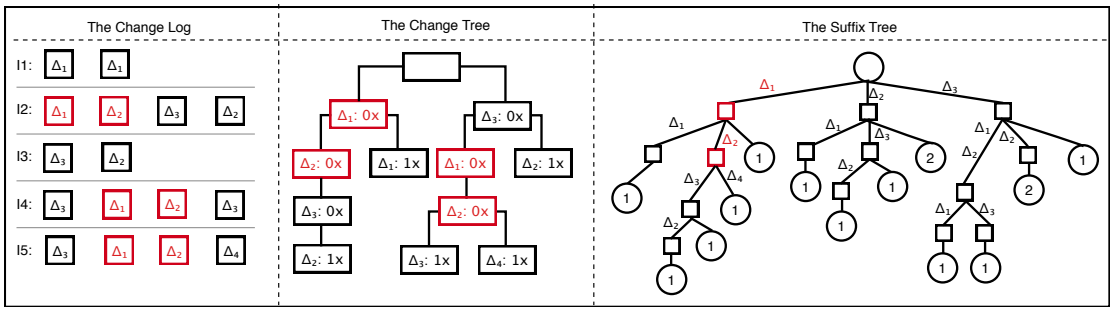


Figure 4.4: Development of the n-gram Change Tree

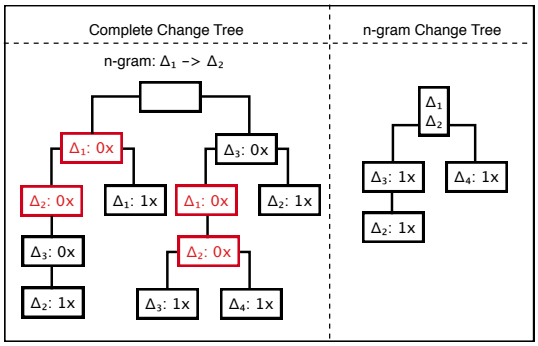


Figure 4.5: All occurrences of the n-gram in the change tree and the n-gram change tree

scenarios, from string matching to finding common motifs in DNA sequences [53]. In contrast to simple implementations of the suffix tree the algorithm proposed by Ukkonen [54] works in linear time O(n), where n represents the length of the string.

In the generalized suffix tree in Figure 4.4 the leaf node of path $< Root, \Delta_2, 2 >$ means that two instances have change $< \Delta_2 >$ as their suffix. There are also two instances which end with the changes $< \Delta_3\Delta_2 >$ (instance I2 and I3), but only one instance which ends with $< \Delta_2\Delta_3 >$ (instance I4).

As depicted in Figure 4.4 the suffix tree is constructed over the suffixes for all strings which can be found in the set of change logs for process instances I1 to I5. For n-gram $< \Delta_1, \Delta_2 >$ we can now easily see the combined subtrees since all suffixes which start with the sequence can be found directly at the root node (marked red in Figure 4.4, third pane). This information can be represented as the n-gram change tree, depicted in the second pane of Figure 4.5. The n-gram change tree contains the n-gram in its root node. The suffixes of the n-gram , i.e., $< \Delta_3, \Delta_2, \Delta_3 >$, and $< \Delta_4 >$ have produced two paths from root to leafs. Specifically, $< \Delta_3, \Delta_2 >$ and $< \Delta_3 >$ are aggregated into one such path.

The n-gram change tree can be defined similarly to the change tree with some modifications as set out in Definition 4.3.

Definition 4.3 (n-gram Change Tree)**.** Let $\mathcal{C}$ be a change log and $\mathcal{D}$ be the changes contained in $\mathcal{C}$. Then n-gram change tree nT is defined as a change tree (cf. Definition 4.1) where the following conditions are different:

1* $r :=< \Delta_1, \ldots, \Delta_l >$ where $\Delta_i \in \mathcal{D}$, i = 1, ..., l is the unique root node

4* $\forall$ paths p from root r to node v = (Δ, n) $\in$ V with n > 0: p corresponds to a projection of n change traces in $\mathcal{C}$ starting from $< \Delta_1, \ldots, \Delta_l >$ as defined in the root node.

4.5 Updating Change Trees

In this section, we discuss how change trees and n-gram change trees can be updated.

Algorithm 5 sets out how new changes are added to a change tree. Deletion on or reorganization of change trees do not become necessary since removing a change Δ from the change tree is considered as adding a "compensating" change Δ'. From an execution point of view it may seem as if the original change Δ has not been applied in the first place; nevertheless, the change log would be incomplete without Δ and Δ'.

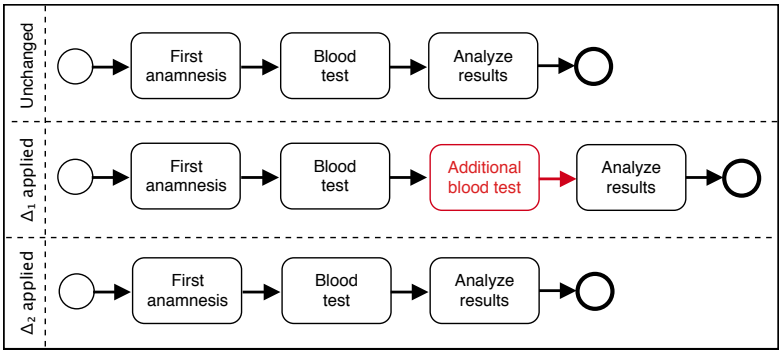


Figure 4.6: The effects of change operations need to be compensated

Take a look at the example depicted in Figure 4.6: Here, first the activity *Additional blood test* has been added to the patient's process instance by means of change operation Δ_1. For medical reasons (e.g., the first blood test proofed to be enough) the activity has been removed from the process instance again. However, as can be seen from this example, it is not enough to simply remove the

change operation Δ_1 from the change log of this process instance, since the process instance still contains the effects of Δ_1 (i.e., the activity *Additional blood test* is still part of the instance's process model). Thus, a second change operation Δ_2 has to be applied, which removes the now unnecessary activity. After applying Δ_1 followed by Δ_2, the process model is in the original state again; however, a second, compensating change operation was necessary. Following, when updating change trees, only the insertion of new change operations has to be considered.

Algorithm 5 shows how new change operations can be added to the change tree.

Algorithm 5: Adding a Change to a Change Tree

Input:

- new change operation Δ (applied to instance I)
- change trace t_I of instance I
- change tree CT (which contains t_I)

```
1  Begin
2  //Find the node containing the current position of I in CT
3  currentnode = root of CT
4  for i = 0; i < t_I.length; i++ do
5      foreach childnode ∈ childnodes of currentnode do
6          if childnode represents change operation on position i of trace t_I then
7              currentnode = childnode
8  if there is a child of currentnode containing Δ then
9      Decrement count of currentnode
10     Go to the node containing Δ
11     Increment the count of this node
12 else
13     Decrement count of currentnode
14     Create a new child node for currentnode containing Δ
15     Set the count of this node to 1
16 End
```

The algorithm works as follows: It receives a change operation which is added to a change trace, the (yet unchanged) change trace, and the change tree (which has to be updated) as input. First, the algorithm iterates over the change trace where the new change operation is to be stored. The node representing the change operation stored at the last position of this change trace is the node where (a) either a child node has to be updated, or where (b) a new child node has to be added. This depends on the change operation which is being added: If it is equal to a change operation stored in one of the child nodes of this node, the counter of this child node can simply be incremented. If there does not exist a child node yet (because no trace has contained such a change operation at this position), a new

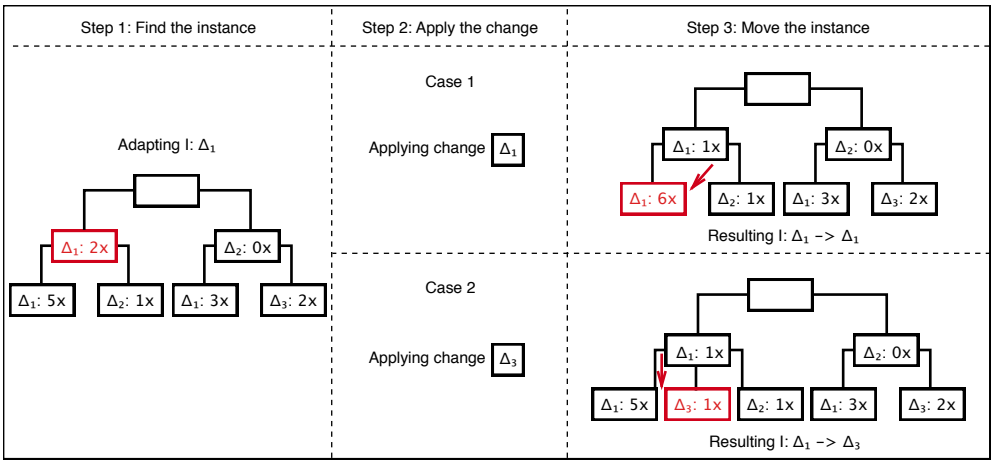


Figure 4.7: Adding a change to a change tree

change tree node has to be generated.

In order to illustrate this, consider Figure 4.7 which depicts two scenarios based on a change trace $t =< \Delta_1 >$ and a given change tree (left side). In a first scenario change Δ_1 is again applied to t such that t is updated to $t =< \Delta_1, \Delta_1 >$. As a first step t is located in the change tree (lines 2 – 7 in Algorithm 5). As a second step we look if the currently applied change has already been applied to some other change trace t' with $t = t'$ for t before applying Δ_1 the second time (lines 5–9). If such t' exists, the change trace is moved further up the change tree to the next level. This is the case for t in scenario 1. Now assume in a second scenario that not Δ_1 has been applied a second time, but Δ_3 instead. In this case, no t' exists with $t = t'$ (before the new change). Thus a new branch in the change tree is created and the change trace is moved there (change Δ_3, second and third pane).

Algorithm 5 also works for n-gram change trees with the only difference that the change is only added to the n-gram change tree if it has been applied after the respective n-gram has been applied to an instance. Consider the example introduced in Figure 4.5: It is based upon a change log containing a total of five traces, i.e.:
$t_0 =< \Delta_1, \Delta_2, \Delta_3, \Delta_2 >$
$t_1 =< \Delta_1, \Delta_1 >$
$t_2 =< \Delta_3, \Delta_1, \Delta_2, \Delta_3 >$
$t_3 =< \Delta_3, \Delta_1, \Delta_2, \Delta_4 >$
$t_4 =< \Delta_3, \Delta_2 >$

When creating the n-gram change tree for the n-gram $< \Delta_1, \Delta_2 >$, only change operations from the traces t_0, t_2 and t_3 are relevant, since only those traces contain the given n-gram. If a new change operation is applied to any of these traces, they

are also of relevance for the respective n-gram change tree.

4.6 Change Trees with Filtering and Projection

N-gram change trees can be used to create a view on the change tree, where information which is not relevant to the current analysis question has been filtered from the change tree. However, using n-gram change trees as a filtering methods, only questions such as *"what has happened after a certain (set of) change operation(s)?"* can be answered. As we have shown in Chapter 3, filtering and projection techniques for change logs provide more flexibility: They can be used to prepare the data in a change log such that the data present is relevant for a wide variety of analysis questions, thus overcoming the information overload problem.

In addition to this, change log filtering can also be applied to highlight possible effects of process change operations: Imagine a production manager in a manufacturing environment inspecting change logs containing projected change traces. These traces contain change operations which have been executed because of the same case, e.g., some supplier did not deliver required resources on time. The goal of the production manager may be to keep the time required to receive the necessary resources minimal, so he inspects historic data which match his criteria. Thus, when filtering for data from the corresponding projected execution log, he can see, which change operations have been applied such that the required resources arrived at the production facility with a delay of maximally two days. How such a projected and filtered change log can be created is described in Section 3.4.3.

However, using such a projected and filtered change log as a basis, he only sees the change traces where the desired effect has been reached. He does not see for how many traces this desired effect has *not* been reached, i.e., the delay was greater than two days. Imagine the projected and filtered change log contains a total of 10 change traces, where 7 contain change trace $s_1 =< \Delta_1, \Delta_2 >$, and 3 contain change trace $s_2 =< \Delta_3, \Delta_4 >$. Using this data set as a basis, he may be tempted to prefer change trace s_1, since it has been applied more often. However, using this projected and filtered data set as a basis, he does not see for how many executions of trace s_1 respectively s_2 the desired effect has *not* been reached. If, for example, for trace s_1 20 traces exist, where the effect has not been reached (i.e., the required delivery time was above 2 days), and for trace s_2 no such traces exist, only those traces containing s_1 would be filtered from the change log which do not match the filtering condition. However, if the production manager had this information as well, he may decide for change trace s_2, since here, the effect has been reached more reliably in the past.

Hence, this section discusses the following research questions:

QF1: How can change log filtering and projection techniques be combined with change trees?

QF2: How can we use filtered and projected change logs in combination with change trees to highlight the probability of possible effects of executed process change operations based on historic data?

The remainder of this section is structured as follows: First, we show that change trees can be created based on projected change traces using the existing change tree algorithm presented in Algorithm 4 (cf. Section 4.6.1), thus answering research question QF1. Based on this, we show in Section 4.6.2 how change traces from a projected change log can be combined with multiple filters, in order to highlight possible effects of executed process change operations ($\rightarrow$ QF2).

4.6.1 Application of Filtering and Projection Techniques

In this section, we discuss how change trees can be generated from projected and filtered change traces, and introduce a running example, which will serve as an illustration for the concepts presented in the remainder of this section.

Projected change traces contain all change events which have been applied to a set of process instances for the same reason, i.e., because of the same case. We have already introduced the example of the production manager in a manufacturing facility, who has to adapt process instances of a production process, because some supplier could not deliver the required resources on time. In order to decide what to do next, he consults historic data from the change and execution log. Change log $\mathcal{C} = (ct_0, ct_1)$ contains the following two change traces:

$ct_0 =< (\Delta_1, "c0"), (\Delta_2, "c1"), (\Delta_4, "c2"), (\Delta_5, "c3"), (\Delta_3, "c4"), (\Delta_2, "c5") >$
$ct_1 =< (\Delta_4, "c6"), (\Delta_3, "c7"), (\Delta_2, "c8"), (\Delta_3, "c9"), (\Delta_2, "c10"), (\Delta_4, "c11") >$

As for the examples given in Chapter 3, each change operation inserts exactly one activity into a process schema:

$\Delta_1 = (\texttt{insert}, \text{“switch to supplier 1”}, \{\text{AND-SPLIT X, AND-JOIN Y}\}, S)$
$\Delta_2 = (\texttt{insert}, \text{“adapt production plan”}, \{\text{AND-SPLIT X, AND-JOIN Y}\}, S)$
$\Delta_3 = (\texttt{insert}, \text{“switch to supplier 2”}, \{\text{AND-SPLIT X, AND-JOIN Y}\}, S)$
$\Delta_4 = (\texttt{insert}, \text{“switch production machine”}, \{\text{AND-SPLIT X, AND-JOIN Y}\}, S)$
$\Delta_5 = (\texttt{insert}, \text{“notify customer”}, \{\text{AND-SPLIT X, AND-JOIN Y}\}, S)$

These change operations have been applied for case instances stemming from one of two different cases, i.e.:

$c_0 = (\ 0,$ "A01 - supplier problems", "supplier change due to delivery problems")
$c_1 = (\ 1,$ "A02 - machine problems", "a production machine experienced a problem")

Five different case instances exist for these two cases, with a_0, a_2, a_4, and a_5 being case instances of case c_0, and a_1, a_3, and a_6 being instances of case c_1:

$\mathbf{a_0} = (\mathbf{c_0}, \mathbf{0}, \mathbf{I_0}, \mathbf{02.04.2018}, \mathbf{04.04.2018})$
$a_1 = (c_1, 0, I_0, 05.08.2018, 05.08.2018)$
$\mathbf{a_2} = (\mathbf{c_0}, \mathbf{1}, \mathbf{I_0}, \mathbf{05.07.2018}, \mathbf{14.07.2018})$
$a_3 = (c_1, 0, I_1, 24.08.2018, 29.08.2018)$
$\mathbf{a_4} = (\mathbf{c_0}, \mathbf{0}, \mathbf{I_1}, \mathbf{03.04.2018}, \mathbf{27.04.2018})$
$\mathbf{a_5} = (\mathbf{c_0}, \mathbf{1}, \mathbf{I_1}, \mathbf{07.07.2018}, \mathbf{09.07.2018})$
$a_6 = (c_1, 1, I_1, 13.09.2018, 15.09.2018)$

Mapping change events to a case instance allows to identify which change operations have been conducted due to which case instance (cf. Section 3.4.2). The mapping of the change events present in the change log $\mathcal{C} = (c_1, c_2)$ above returns the following case instances:

$\mathbf{cC}((\mathbf{\Delta_1}, "\mathbf{c0}")) = \mathbf{a_0}$ $\quad$ $\mathbf{cC}((\mathbf{\Delta_2}, "\mathbf{c1}")) = \mathbf{a_0}$ $\quad$ $cC((\Delta_4, "c2")) = a_1$
$cC((\Delta_5, "c3")) = a_1$ $\quad$ $\mathbf{cC}((\mathbf{\Delta_3}, "\mathbf{c4}")) = \mathbf{a_2}$ $\quad$ $\mathbf{cC}((\mathbf{\Delta_2}, "\mathbf{c5}")) = \mathbf{a_2}$
$cC((\Delta_4, "c6")) = a_3$ $\quad$ $\mathbf{cC}((\mathbf{\Delta_3}, "\mathbf{c7}")) = \mathbf{a_4}$ $\quad$ $\mathbf{cC}((\mathbf{\Delta_2}, "\mathbf{c8}")) = \mathbf{a_4}$
$\mathbf{cC}((\mathbf{\Delta_3}, "\mathbf{c9}")) = \mathbf{a_5}$ $\quad$ $\mathbf{cC}((\mathbf{\Delta_2}, "\mathbf{c10}")) = \mathbf{a_5}$ $\quad$ $cC((\Delta_4, "c11")) = a_6$

Based on the concepts defined in Section 3.4, projecting the change traces to the case instances of case c_0, i.e., a_0, a_2, a_4, and a_5 returns the projected change log $\mathcal{C}_{A01}$ containing the following change traces (relevant information is depicted bold above):

$ct_{a0} =< (\Delta_1, "c0"), (\Delta_2, "c1") >$
$ct_{a2} =< (\Delta_3, "c4"), (\Delta_2, "c5") >$
$ct_{a4} =< (\Delta_3, "c7"), (\Delta_2, "c8") >$
$ct_{a5} =< (\Delta_3, "c9"), (\Delta_2, "c10") >$

Next, the production manager wants to filter these traces, such that the resulting projected and filtered change log only contains those traces, where the resulting delivery time of the required resources was below or equal to two days. Hence, the filtering condition is `delivery_time <= 2`. This information can be deduced from the corresponding execution log $\mathcal{X} = (t_1, t_2)$, which contains the following data:

$t_1 =<$
`(switch to supplier 1, "e1", 02.04.2018 - 09:27, {(delivery_time, 1)}),`
`(adapt production plan, "e2", 04.04.2018 - 09:27, ∅),`
`(switch production machine, "e3", 05.07.2018 - 19:11, ∅),`
`(notify customer, "e4", 12.02.2018 - 11:31, ∅),`
`(switch to supplier 2, "e5", 05.07.2018 - 11:31, {(delivery_time, 8)}),`
`(adapt production plan, "e6", 14.07.2018 - 13:46, ∅)`$>$
$t_2 =<$
`(switch production machine, "e7", 05.07.2018 - 19:11, ∅),`
`(switch to supplier 2, "e8", 04.02.2018 - 11:31, {(delivery_time, 1)}),`
`(adapt production plan, "e9", 11.02.2018 - 09:27, ∅),`
`(switch to supplier 2, "e10", 04.02.2018 - 11:31, {(delivery_time, 7)}),`
`(adapt production plan, "e11", 11.02.2018 - 09:27, ∅),`
`(switch production machine, "e12", 05.07.2018 - 19:11, ∅)`$>$

The activities which contain the data relevant to the filtering condition, i.e., `delivery_time`, are *switch to supplier 1* (inserted by change operation Δ_1), and *switch to supplier 2* (inserted by change operation Δ_3). The data set also contains information about which execution event containing this data of interest has been influenced by which change operation:

$xC((\Delta_1, "c0")) = \{(\text{switch to supplier 1}, \text{“e1”}, \ldots)\}$
$xC((\Delta_3, "c4")) = \{(\text{switch to supplier 2}, \text{“e5”}, \ldots)\}$
$xC((\Delta_3, "c7")) = \{(\text{switch to supplier 2}, \text{“e8”}, \ldots)\}$
$xC((\Delta_3, "c9")) = \{(\text{switch to supplier 2}, \text{“e10”}, \ldots)\}$

Applying this data to the combined filtering and projection Algorithm 3 from Section 3.4.3 results in the following filtered and projected change traces $\mathcal{C}_{A01F} = (ct_{a0}, ct_{a4})$:

$ct_{a0} =< (\Delta_1, "c0"), (\Delta_2, "c1") >$
$ct_{a4} =< (\Delta_3, "c7"), (\Delta_2, "c8") >$

Using this data base, the generation of change trees based on projected change traces as shown in the example given above is straightforward: After projecting (and filtering) the change traces using the techniques presented in Chapter 3, a projected (and filtered) change tree can be created using Algorithm 4. Figure 4.8 shows the change tree from the original change log $\mathcal{C}$ in the left pane, and the projected change tree for the projected change log $\mathcal{C}_{A01}$ in the middle pane. The

right pane shows the change tree for the projected and filtered change log $\mathcal{C}_{A01F}$.

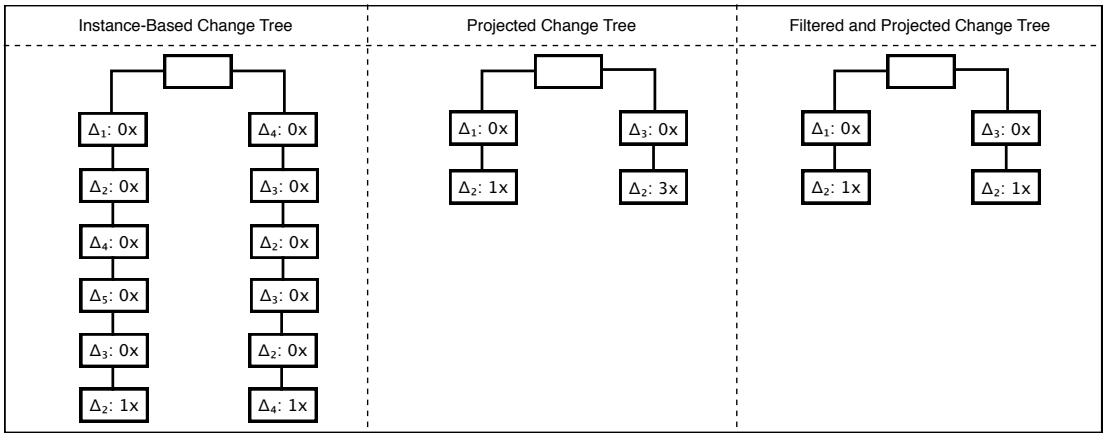


Figure 4.8: Example: A change tree and its projection onto case A42

The case-based projection of change logs and the corresponding generation of change trees can be used to project a change log along a specific case. Thus, change trees can be used to inspect the chronological order of applied process change operations for instances of a certain case. However, as stated in the introduction of this section, a change tree based on a projected an filtered change log only contains information about change events which fulfilled the given filtering condition. Using this historic data, the production manager might think that both change traces are equivalent with respect to the filtering condition, since both exist exactly once in the filtered and projected process change log. However, what he does not see here, is the fact that there exist two other change traces containing change trace $< \Delta_3, \Delta_2 >$, which did not match the filtering criterion. Thus, only for one out of three cases this criterion was met. If the production manager uses historic data as a basis for deciding which change trace to use, when a supplier has to be changed, this information might be important. In the next section, we will present the *Enhanced Change Tree*, a representation method for change logs which also contain information about all change events which did not fulfill this condition, thus providing a bigger picture on the change log. For this, the example introduced in this section will be used.

4.6.2 Enhanced Change Trees

In the last section, we have shown an example from the production domain, where a production manager uses historic data to analyze which change trace has led to a desired effect in the past, i.e., keeping the required delivery time of some resource below the threshold of two days. By projecting and filtering the change

traces using the algorithms presented in Section 3.4, and creating a change tree based on the projected and filtered change log, he sees that two traces exist, which lead to this desired effect, i.e., $ct_{a0} =< (\Delta_1, "c0"), (\Delta_2, "c1") >$ and $ct_{a4} =< (\Delta_3, "c7"), (\Delta_2, "c8") >$. However, he does not see that for two additional change traces ct_{a2} and ct_{a3}, which describe the same change trace $< \Delta_3, \Delta_2 >$, did not reach the desired effect.

In this section, we present the *Enhanced Process Change Tree* (EPCT), which combines the filtered with the non filtered projected change traces, thus highlighting for how many change traces from a set of change traces a certain boolean filtering condition has been met. For this, we define a labeled filtering condition set as follows:

Definition 4.4 (Labeled Filtering Condition Set)**.** Let $\mathcal{X}$ be a process execution log, $\mathcal{L}$ be the universe of all labels, and Cond be the set of all filtering conditions for $\mathcal{X}$, $d \odot val$ as defined in Section 3.4.1. Then, $\mathcal{F}$ is a labeled filtering condition set for $\mathcal{X}$, with $f \in \mathcal{F} : f \subseteq \mathcal{L} \times Cond$, which pairs a boolean condition with a unique label (i.e., a short, expressive description of the condition).

For the example given above, the labeled filtering condition set $\mathcal{F}$ contains the element $f_0 =$("Short delivery time", `delivery_time <= 2`).

The enhanced process change tree is defined as follows:

Definition 4.5 (Enhanced Process Change Tree)**.** Let $\mathcal{C}_A$ be the projected change log for case A. Let $\mathcal{F}$ be a labeled filtering condition set for the corresponding execution log $\mathcal{X}$, and let $\mathcal{L}$ be the universe of all labels. We define $CF_A = \{(label, cond, filter(\mathcal{C}, cond)) | cond \in Cond, label \in \mathcal{L}\}$, where $\forall f \in \mathcal{F} \exists cf := (label, cond, log) \in CF_A$ with *label* being the label of the filtering condition, *cond* being the condition, and *log* being the filtered projected change log generated with the filtering condition.

Then the enhanced process change tree $EPCT_A := (r, V, E)$ for case A is a process change tree where:

- r, E are defined as in Definition 4.1
- V $\subseteq \mathcal{C} \times \mathbb{N}_0 \times 2^{(\mathcal{L} \times \mathbb{N})}$
  $\forall v \in V : v = (\Delta, n, L_v)$ with
  - Δ, n are defined as in Definition 4.1
  - $\forall (y.label, y.n) \in L_v$: n is the number of all traces in cf.log containing this Δ, where cf.label = y.label

For the example given above, one filtering condition exists, i.e., `delivery_time <= 2`. Hence, there exists exactly one tuple in CF_A, which is ("short delivery time",

`delivery_time <= 2` , $\mathcal{C}_{A01F}$), where $\mathcal{C}_{A01F}$ contains the following two projected change traces:

$ct_{a0} =< (\Delta_1, "c0"), (\Delta_2, "c1") >$
$ct_{a4} =< (\Delta_3, "c7"), (\Delta_2, "c8") >$

The projected change log $\mathcal{C}_A$ from the example contained the following traces:

$ct_{a0} =< (\Delta_1, "c0"), (\Delta_2, "c1") >$
$ct_{a2} =< (\Delta_3, "c4"), (\Delta_2, "c5") >$
$ct_{a4} =< (\Delta_3, "c7"), (\Delta_2, "c8") >$
$ct_{a5} =< (\Delta_3, "c9"), (\Delta_2, "c10") >$

Using the EPCT as a basis allows for a deeper insight into process change logs: The left pane of Figure 4.9 shows the projected change tree without any filtering condition, and the middle pane shows the same projected change tree filtered for the filtering condition `delivery_time <= 2`: In the left pane, the production manager sees that overall three projected change traces exist which contain $< \Delta_3, \Delta_2 >$, and one change trace consists of $< \Delta_1, \Delta_2 >$. In the middle pane, which depicts the change tree based on the filtered and projected change log, the production manager only sees that for one change trace containing $< \Delta_1, \Delta_2 >$, respectively $< \Delta_3, \Delta_2 >$, the filtering condition has been met. However, he does not see that for change trace $< \Delta_3, \Delta_2 >$, two other traces exist where the condition has not been met. This information can only be found in the enhanced process change tree (right pane of Figure 4.9). However, for the given example, such information might be relevant: If a change trace has not reached this desired effect in two out of three cases in the past, it might not be the optimal solution. On the other hand, the other change trace which has reached the goal once has only been used once before. Using this information as a basis, the production manager can now decide on a change trace for his supplier problem.

4.6.3 Discussion: Filtering, Projecting, and the EPCT

In the last section filtering and projecting of process change logs as a basis for representation with change trees, and the EPCT have been described. In this section, we provide an overview over the required data sources, and discuss limitations if certain data sources are not available.

Figure 4.10 shows the interaction of the data sources the EPCT, and filtering and projecting for change trees is built upon. On the top, three phases when planning and executing process adaptations are shown: (1) preparation, where the adaptation is planned, (2) process adaptation, where the process instance is

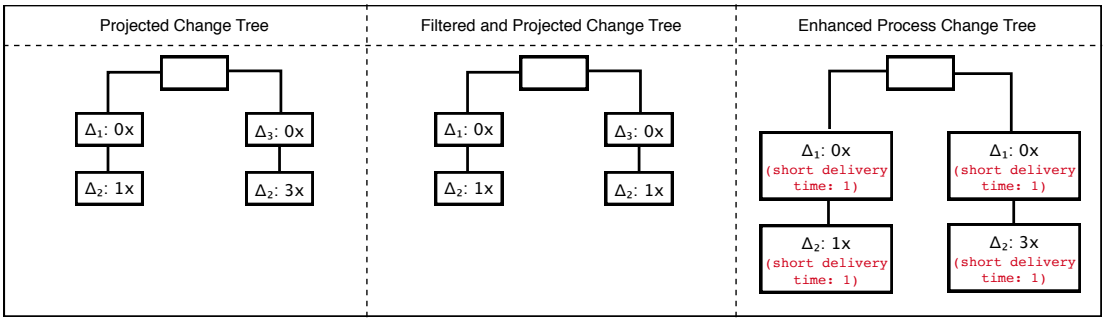


Figure 4.9: Example of an Enhanced Process Change Tree for the manufacturing domain

changed, and (3) process execution, where the effects of the executed adaptation become relevant. These three phases will be discussed in detail below. In the lower part of the figure, the related components and data sources are shown.

First, as soon as the problem at hand is known, the adaptation of the process instance has to be planned. In a medical setting, this refers to planning the therapy of a patient. The process fragment which has to be inserted into the process instance has to be specified. Based on this data, a change operation, or a trace of change operations is created.

When the change operation is defined, the process instance can be adapted. When adapting a process instance, the corresponding *change log* receives an entry containing detailed information about the change operation. This information is then used as a basis for the change tree.

After adapting the process instance, the activities of the process instance which have been affected by the change operation will be executed sooner or later. When a process activity is executed, it is logged in the *execution log*. The execution log also stores data related to the process instance, which may change while or after the activities of a certain process fragment are executed. For example, variables describing the patients condition, e.g., the patient's body temperature, can be tested in a process activity, and the results can be obtained afterwards from the execution log. Such data can then be used for filtering conditions, as explained above.

The approach presented in this section requires different data sources, i.e.,

- A *change log* containing the change operations which are applied to a process instance
- An *execution log* containing the executed process activities
- A *set of cases* containing the cases, which describe situations in which process adaptions might be necessary

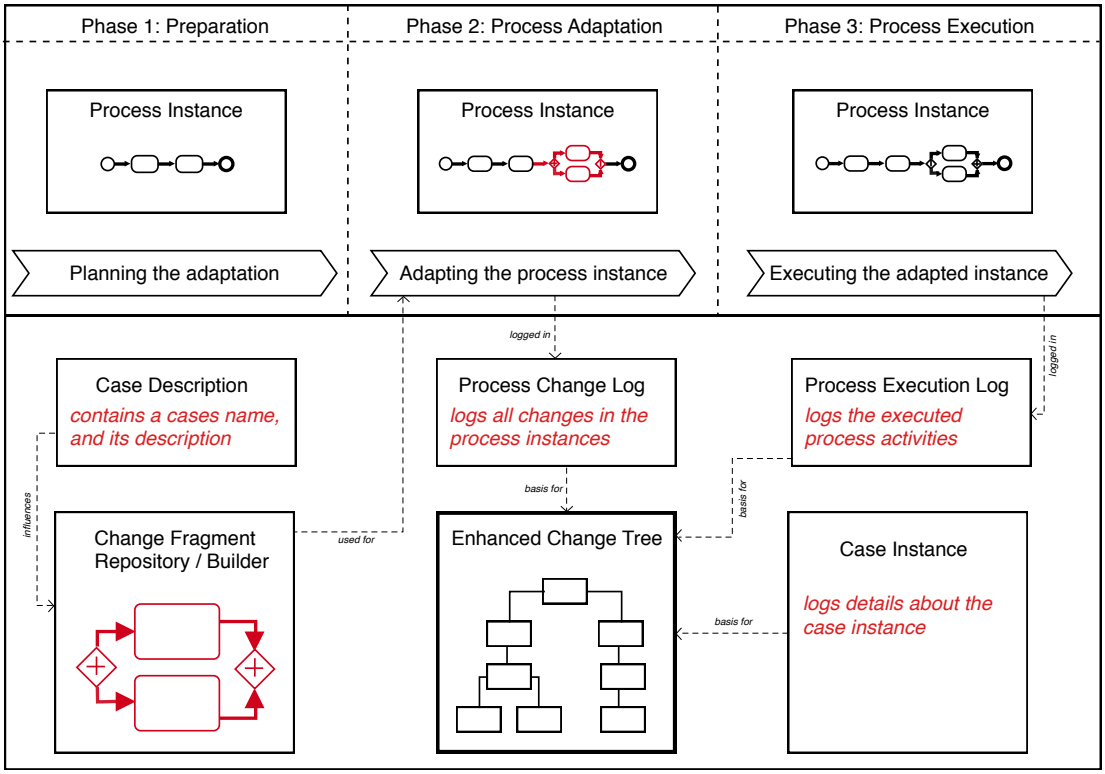


Figure 4.10: Components and Data Sources of the EPCT

A possible threat to this approach stems from the fact that not all of these data sources may be available in some settings. In this section we discuss what can be compared if some of the data sources are missing.

Information about conducted change operations is a mandatory basis for building the change tree. Even if no other data source is available, this representation can be used to find out what has been done in the past after a certain change operation has been applied. However, even if no change log is available, information about possible process adaptations can be extracted from process execution logs, which are often created by business process management systems [55]. For example, *concept drift* [56, 57]) uses changes in the data of a processes execution log to analyze which change operations might have been applied. Such changes in execution log data can also be spotted by analyzing time-related dimensions using *multidimensional process mining* [58].

Information of conducted change operations in combination with information about the execution of activities (i.e., the execution log), one can see whether activities which are added in some change operation have been executed or not. If activities have been added, but not executed, this may be a hint that these

activities were not required any more when they should have been executed. Such an analysis may lead to further investigation, e.g., by interviewing the practitioners who were responsible for executing the process activities.

If there also exists a set of cases, the change operations can be linked to those. This information alone is sufficient to generate the case-based change log as described in Section 3.4.2. Such cases exist for example in the medical or nursing domain, where each problem a patient can show is described in detail. For example, the nursing domain contains guidelines named NANDA, NIC and NOC[2], which describe diagnoses, possible therapy steps, and goals which can be reached for them. For example, for the case *Fatigue*, a goal exists which is described as *"The patient verbalizes increased energy"* [59]. Such goal descriptions can be translated into boolean conditions for filtering criteria on data from the execution log: If a patient states that his energy has in fact increased, and this statement can be found in the processes execution log (e.g., because he has been asked as part of a certain therapy process activity), a boolean condition can check for the existence of such a statement, and thus whether to filter respective change traces from the log. Aside the nursing and medical domain, some companies have implemented guidelines which define how to handle regularly occurring situations with their customers. In Chapter 7, we will describe such situations for different domains, including *hotel and event management*, *software development and support*, and *special needs schools*.

4.7 Change Tree Node Aggregation

In the last section, we have shown how filtering and projection techniques can be applied to change trees to (a) increase the information contained by in a change tree by creating an EPCT, or (b) reducing the number of nodes by filtering and projection of the change logs the tree is based on. In this chapter, we will discuss a different method for reducing the number of nodes present in a change tree. Change trees are constructed by aggregating equal change operations on the same position of change traces in a single node, and generating sibling nodes for different change operations. However, change trees compare process change operations based on their label only. If two change operations have the same label, they are considered to be equal. For many situations, such an assumption proves to be too general. Depending on the analysis question at hand, for example, it could be more interesting to compare process change operations based on the required resources, temporal constraints, or other attributes of the change operation. In other words, comparing change operations shall be shifted from *label equivalence* towards comparing their *semantics*.

[2] `http://www.nanda.org`

The idea behind is to replace label equivalence with *change similarity metrics* [16]. As discussed in Chapter 2, change similarity metrics calculate the similarity of two process change operations from different change perspectives, e.g., change attributes or change effects on the adapted process instance. This enables to compare change operations regarding, e.g., the required resources, time constraints, or affected control and data flow.

Imagine a nursing home with hundreds of patients, where each patient's therapy plan is represented by a process instance. Each task in this process instance represents an activity which has to be completed in order to improve the health status of a patient. Whenever an adaptation to a therapy instance is required (e.g., because the patient feels sick), his process instance has to be adapted, thus generating a new change in his change trace. As discussed in Chapter 2, the *Change Resource Similarity (CRS)* [16] calculates the similarity of the required resources of two change operations which insert activities into a process instance in an interval between 0 and 1: If two inserted process fragments require e.g. 3 nurses each, CRS returns 1, if they require totally different resources, it returns 0. By aggregating the change traces of several similar patients based on a similarity metric, such as CRS, responsible users can analyze change traces based on different process perspectives. If a nurse wants to know which resources are usually required for the treatment of a specific disease, she could investigate the change tree where similar change operations regarding CRS are aggregated in one node.

Replacing label equivalence by change similarity metrics poses several challenges. If changes are similar, different options for merging them in the resulting change tree become possible. This necessitates theoretical considerations on how a change tree can be built in this case. Moreover, the interpretation of a change tree based on similarity metrics is different from one based on label equivalence. Finally, it has to be investigated which benefits arise from employing change similarity instead of label equivalence for change trees. This leads to the following research questions:

QA1: How can change traces be analyzed and compared in a semantic way? Which metrics can be used?

QA2: How do these metrics have to be applied to a set of change traces?

In this section, we first discuss basic considerations which are relevant when aggregating similar process change operations in change traces ($\rightarrow$ QA1). This is followed by an analysis of how similarity metrics for process change operations can be applied to sets of change traces as they can be found in a change log, thus allowing us to answer different questions about change traces ($\rightarrow$ QA2).

4.7.1 Applying Similarity metrics to Change Logs

So far, in a change tree, two (potentially sibling) change operations have only been aggregated in one node they were equivalent regarding their labels; the attributes and effects of the change operations were not considered at all. Using label equivalence has been discussed as a potential limitation in the area of process mining (cf. e.g., [60]). For example, it would be possible that two activities have equal labels, but launch different actions based on the context they are executed in, or that two activities with different labels are actually semantically equivalent. Hence, for certain analysis questions, using an equivalence notion that refers to the semantics of activities would be useful [61]. The same holds for comparing change operations.

Consider the example depicted in Figure 4.11: Change Operation Δ_1 inserts a process fragment consisting of activity D into process schema 1 between the XOR split and activity A. Change Operation Δ_2 adds the same process fragment at the same position, but this time on the slightly different process schema 2. Comparing Δ_1 and Δ_2 by their label equivalence is not optimal, since their attributes and effects seem quite similar. However, based on the label equivalence alone, these two change operations would still be considered different, which can pose a problem for many analysis questions.

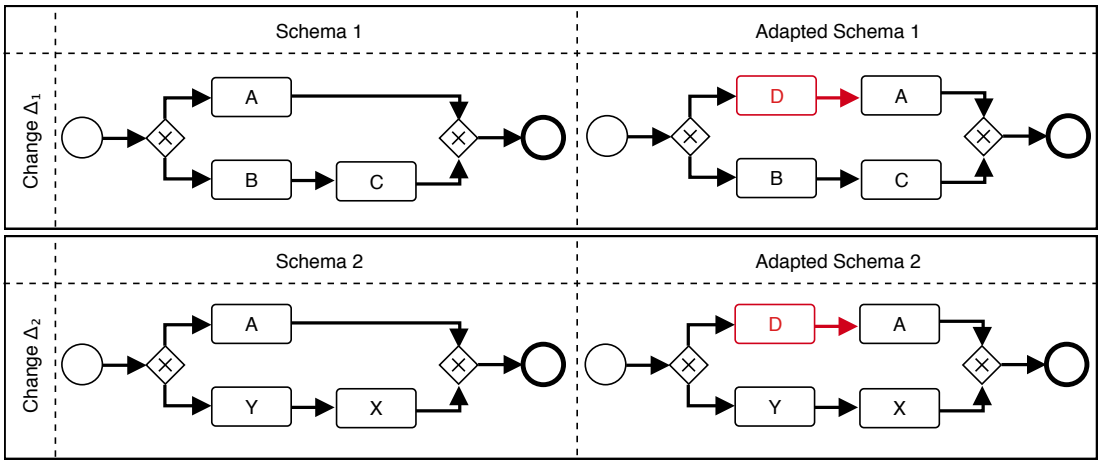


Figure 4.11: Example of similar process change operations

As a second example, consider the nursing home example given in the introduction: Since it is very improbable that multiple patients ever require the very same set of therapies, the process schemata, and the positions where process change operations are applied to are also very different. When analyzing the similarity of two process change traces, different analysis questions would be of interest: For example one could ask for the similarity of the inserted fragments, while another

question might be to consider the schema the fragments are applied to, since they represent prior therapies of the patients. Moreover, it can be beneficiary to analyze change actions that have similar effects on the process instance, e.g., on the resource view, together, instead of keeping them in different nodes. Thus, no matter which method was chosen to generate the labels of the process change operations, the analyzability of the resulting change tree would be restricted inevitably.

In summary, it would be advantageous to decide more flexibly whether two process change operations should be aggregated or not. This can be achieved by replacing label equivalence with process change operation similarity metrics [16].

Hence, this chapter proposes the aggregation of nodes in change trees based on process change similarity instead of using label equivalence. If, for example, one wants to know how different the applied changes were regarding the required resources, a similarity metric like CRS could be used. In the following, we denote the similarity of two change operations Δ_1 and Δ_2 as $sim(\Delta_1, \Delta_2) \in [l, 1]$.

For some similarity metrics, $l = 0$ holds, for others $l = -1$ is defined. Take the resource based similarity CRS as an example [16]. It compares the resources used by the fragments of two change operations Δ_1 and Δ_2, which are inserted into or deleted from a process schema. If the resources are exactly the same, and both change operations insert or delete, it returns 1. If the resources are exactly the same, but one change operation inserts, and the other deletes, it returns -1. If the required resources are different, it returns a value between (-)1 and 0. Contrary, the attribute based similarity metric $sim_{attr}(\Delta_1, \Delta_2)$ returns a value in the range of $[0, 1]$ [16]. It is calculated by defining the similarity for each attribute of the two process change operations (fragments, positions, operation types and schemas), and weighing the results. Using weights enables to prefer certain attributes when comparing the change operations. Attributes can be even ignored by assigning a weight of 0 to them.

Which process change operation similarity metric is used highly depends on the analysis question: If resources are of interest, a resource-based similarity metric such as CRS may be useful; if control-flow related features are of interest, a more attribute-based metric might be favorable. However, the approach presented in this chapter abstracts from the implementation of concrete similarity metrics. Thus, any metric which calculates a similarity $sim(\Delta_1, \Delta_2) \in [l, 1]$ can be used as a basis for creating the aggregated change tree.

To discriminate between traditional change trees (based on label equivalence) and the change trees based on similarity metrics as presented in this chapter, we will refer to the latter by the term *aggregated change tree*. The term stems from the fact that multiple similar change operations will be *aggregated* in a single node. Figure 4.12 depicts an example of a change log, a change tree and an aggregated change tree. The middle pane shows a change tree according to Definition 4.1

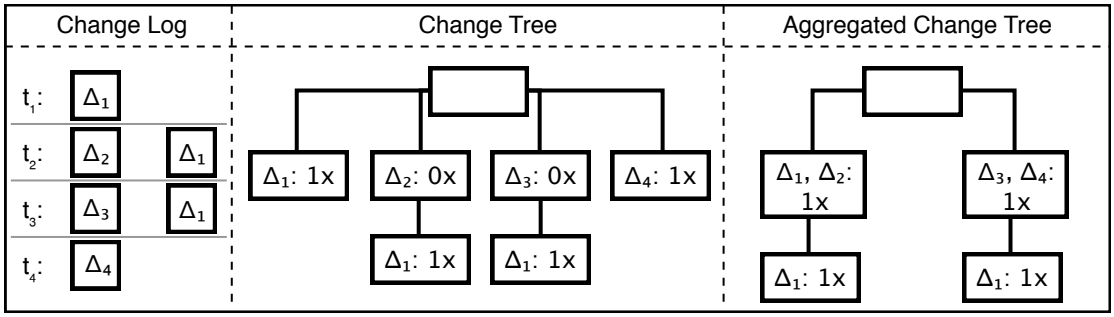


Figure 4.12: Basic Example

based on the change log in the left pane: It consists of two levels, where the first level contains the four change operations Δ_1, Δ_2, Δ_3 and Δ_4. The change traces t_2 and t_3 additionally contain a second change operation, Δ_1. The right pane depicts the *aggregated change tree*. In the following sections, we will discuss how this aggregated change tree is created.

To preserve the change traces of a change log, which are represented in a change tree, only change operations are considered for aggregation which would be on the same level of the change tree. While it would be possible to aggregate change operations in an aggregated change tree vertically (by aggregating change operations which happened consecutively), the horizontal aggregation as presented in this chapter preserves the basic properties of the change tree: As the change tree, the aggregated change tree should, when read from the root to the leafs, represent the set of change traces from a process log. If consecutive change operations would be aggregated as well, this property would be lost.

The only difference between the change tree and the aggregated change tree lies within the structure of the nodes. Instead of a node V containing a single change operation (as defined in Definition 4.1), V now contains a set of similar change operations:

Definition 4.6 (Aggregated Change Tree)**.** Let $\mathcal{D}$ be a set of change operations as defined in Definition 4.1. An aggregated change tree $AT := (r, V, E)$ is a change tree $T := (r, V, E)$ where V $\subseteq 2^{\mathcal{D}} \times \mathbb{N}_0$.

There are two border cases for the number of nodes present in the aggregated change tree (ACT) compared to the change tree: Due to the horizontal aggregation, the minimum number of nodes equals the depth of the original change tree, since the ACT contains only one node for each level of the change tree. If however no nodes can be aggregated, the ACT contains exactly as many nodes as the change tree.

In order to decide whether two change operations Δ_1 and Δ_2 should be aggregated in a single node, a similarity threshold value $\tau \in [l, 1]$ is specified where l

corresponds to the lower bound of the similarity metric of interest. If the similarity metric $sim(\Delta_1, \Delta_2) \geq \tau$, Δ_1 and Δ_2 are aggregated in a single node; else, two separate nodes are created. In the next section, we will present two approaches for generating aggregated change trees based on a similarity metric, a threshold τ, and a change log.

4.7.2 Generating the Aggregated Change Tree

Depending on the chosen similarity threshold τ, the aggregated change tree can be created in different ways. Section 4.7.2 presents the algorithms for $\tau < 1.0$ and Section 5 follows up with an illustration for $\tau = 1.0$.

Aggregation for Similar Process Change Operations

When aggregating similar change operations in one node, the similarity threshold τ has to be reached for all similarity metrics between the change operations in the aggregation node. Figure 4.13 shows an example of such an aggregation: In pane 1, a change log is depicted which contains three change traces consisting of the process change operations Δ_1, Δ_2 and Δ_3. Pane 2 shows the similarity of the three change operations: $sim(\Delta_1, \Delta_2) = 0.75$, $sim(\Delta_2, \Delta_3) = 0.8$ and $sim(\Delta_1, \Delta_3) = 0.6$. Pane 5 shows the change tree which would be generated based on label equivalence (cf. [37]). Depending on the chosen similarity threshold τ, different change operations can be aggregated in one node: If $\tau = 0.6$, all three change operations can be aggregated in one node (cf. aggregated change tree depicted in pane 6). If $\tau = 0.7$, only change operations Δ_1 and Δ_2, or change operations Δ_2 and Δ_3 can be aggregated in one node, but not Δ_1, Δ_2 and Δ_3, since the similarity between Δ_1 and Δ_3 is below τ. The similarity metric is not transitive. Thus, for $\tau = 0.7$, two aggregated change trees are possible (cf. panes 7 and 8). These two change trees are the results of two different aggregation strategies, which are explained at the end of this section.

The similarity metrics between all change operations which can be aggregated in one node need to be above the threshold τ. In order to find the subset of change operations for which this condition holds, we introduce the *change similarity graph.* (cf. Definition 4.7).

Definition 4.7 (Change Similarity Graph)**.** Let $\mathcal{D}$ be a set of change operations, $sim(\Delta_i, \Delta_j)$ be a similarity metric, and τ the similarity threshold. The Change Similarity Graph $\Gamma := (\mathcal{D}, E)$ is an undirected graph where $E \subseteq \mathcal{D} \times \mathcal{D}$ denotes the set of edges. There exists an edge between change operations $\Delta_i, \Delta_j \in \mathcal{D}$, iff $sim(\Delta_i, \Delta_j) \geq \tau$.

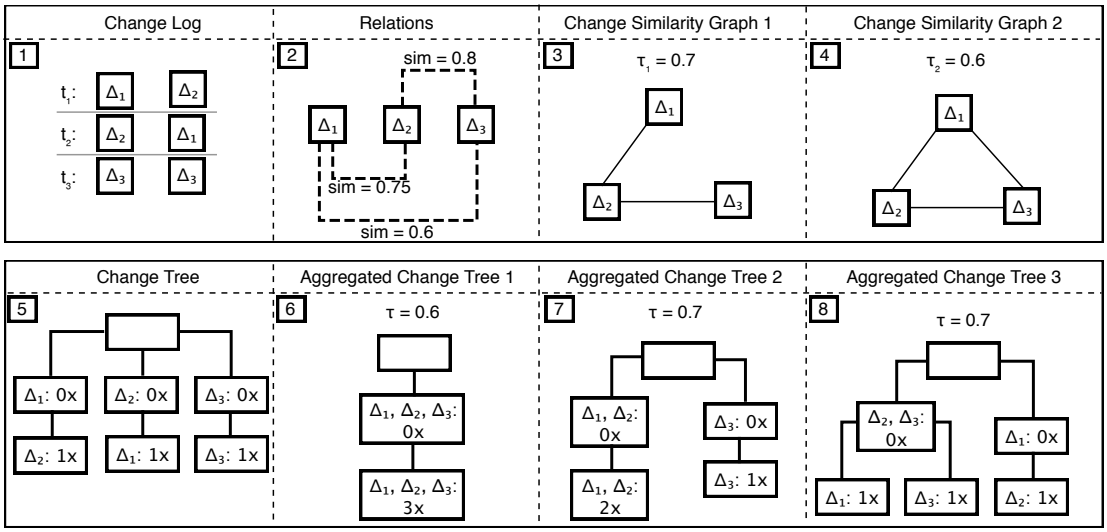


Figure 4.13: Transitivity Example

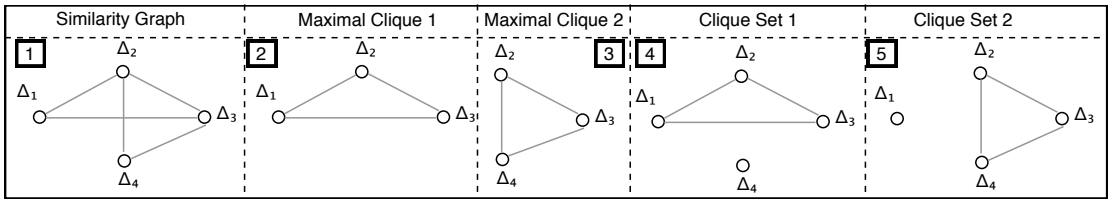


Figure 4.14: Set of maximal cliques in a change similarity graph

Pane 3 and 4 of Figure 4.13 show the similarity graphs for two thresholds $\tau_1 = 0.7$ and $\tau_2 = 0.6$ for the process change operations Δ_1, Δ_2 and Δ_3 from the example given above. In the similarity graph for $\tau_1 = 0.7$, there exist only edges between change operations Δ_1 and Δ_2, and between Δ_2 and Δ_3. There is no edge between Δ_1 and Δ_3, since its value lies below the threshold at 0.6.

Using the similarity graph as a basis, the problem of finding the subset of change operations where the similarity of all change operations exceeds the threshold τ can be seen as the problem of finding all maximal cliques in the graph. A maximal clique in an undirected graph is a *maximal complete subgraph* [62], that is a set of nodes and edges from that graph, where there exists an edge between each node. Using a set of cliques as a basis, one could aggregate all change operations which are in the same clique, since the similarity of those change operations exceeds τ.

The graph depicted in the third pane of Figure 4.13 ($\tau = 0.7$), contains two cliques: One contains change operations Δ_1 and Δ_2, the other contains change operations Δ_2 and Δ_3. In the graph in the fourth pane ($\tau = 0.6$), there exists one

clique which contains all three change operations. The *Bron-Kerbosch algorithm* [62] takes an undirected graph such as the change similarity graph as input, and returns all maximal cliques of this graph. Figure 4.14 shows an example for a set of maximal cliques in a change similarity graph. Pane 1 shows the similarity graph which contains four different process change operations. By applying the Bron-Kerbosch algorithm [62], two maximal cliques (depicted in panes 2 and 3) are generated. Obviously, Δ_2 and Δ_3 appear in more than one clique. If both cliques were used directly to generate the nodes of the aggregated change tree, Δ_2 and Δ_3 would appear in more than one node. The question is now how the set of cliques (as depicted in panes 2 and 3) should be adapted, so that each node in the original graph only appears in exactly one clique. This procedure is explained in Algorithm 6.

Algorithm 6: Generating set of clique sets

Input: change similarity graph $\Gamma := (\mathcal{D}, E)$
Output: set of clique sets $\mathcal{U}$ where each node only appears once per clique sets

```
1  U = ∅
2  generate-cliqueset(D, ∅, U)
3  return U
4  function generate-cliqueset(nodes, currentCliqueset, finalCliqueset)
5      if |currentCliqueset| == 0 then
6          newset = true
7      else
8          newset = false
9      //the Bron-Kerbosch algorithm is defined in [62]
10     C = bron-kerbosch(nodes)
11     //now clean up: each node may only appear once per clique set
12     //start with all cliques with the highest number of nodes
13     foreach {c ∈ C||c| = max(|c| ∈ C)} do
14         currentCliqueset = {c}
15         //remove all nodes from this clique from the graph for the next iteration
16         cleanedNodes = nodes \ {c}
17         returnCliqueset = generate-cliqueset(cleanedNodes, currentCliqueset, finalCliqueset)
18         currentCliqueset = currentCliqueset ∪ returnCliqueset
19         if newset == true then
20             finalCliqueset = finalCliqueset ∪ currentCliqueset
21         else
22             return currentCliqueset
```

Algorithm 6 uses the Bron-Kerbosch algorithm [62] and a similarity graph to generate a set of *unique clique sets* (cf. Definition 4.8), such that each node appears only in one clique of the set. Depending on the structure of the graph, different clique sets can be created. The algorithm applied to the example given in Figure

4.14 creates the two clique sets depicted in pane 4 and 5 of Figure 4.14.

Definition 4.8 (Unique Clique Set)**.** Let $\mathcal{N}$ be the set of nodes from a change similarity graph. A unique clique set is a set of cliques, such that each node appears in exactly one clique.

For this example, it works as follows: It starts by calling the recursive function *generate-cliqueset()* with the change similarity graph Γ, an empty set, and the output set $\mathcal{U}$, which is at the beginning empty as well. Next, for the current graph, the set of maximal cliques is determined and stored in $\mathcal{C}$ using the Bron-Kerbosch algorithm [62]. In the example, $\mathcal{C} = \{\{\Delta_1, \Delta_2, \Delta_3\}, \{\Delta_2, \Delta_3, \Delta_4\}\}$. Next, the algorithm iterates over all cliques in the set of cliques with the highest amount of nodes. In the example, both cliques have the same amount of nodes, so both are used. Next, it creates a new graph *cleanedNodes* which contains all nodes from the original graph, except the nodes from the clique. Thus, for the first clique, it contains only Δ_4, and for the second clique, it contains only Δ_1. Next, the function *generate-cliqueset()* is called recursively with the graph *cleanedNodes* (containing Δ_4 respectively Δ_1), the current clique set as a second parameter, and the - still empty - final clique set as the third parameter. The nested recursive call returns Δ_4 respectively Δ_1, since they are the only possible cliques which are left in the graph, to the first call of the function *generate-cliqueset()*. There, these nodes, together with the current clique, are added to the set *finalCliqueset*, which - after both iterations of the foreach loop - contains the two unique clique sets as depicted in Figure 4.14 (panes 4 and 5).

These two unique clique sets can now be used to generate the change tree nodes. Depending on the choice, either Δ_1, Δ_2 and Δ_3 are aggregated in one node, and Δ_4 is in the other node, or Δ_2, Δ_3 and Δ_4 are aggregated in a node, and Δ_1 is in its own node. Two distinct features can be used as a basis for deciding which process change operation should be aggregated in a node: (a) the most similar change operations should be aggregated or (b) the number of nodes in the tree should be minimized.

Figure 4.15 shows a change log with four change operations Δ_1 to Δ_4. Pane 2 depicts the similarity relations between them. In the remainder of this section, this example will be used to explain the different aggregation strategies.

Aggregating the most similar change operations Algorithm 7 shows the strategy which can be used to aggregate the most similar change operations on one level into an aggregated change tree node. It is based on the creation for change trees as defined in [37] and starts with the first level of change operations in the log, containing Δ_1 to Δ_4 (cf. example from Figure 4.15). First, the unique clique sets are created. For a $\tau = 0.3$, two unique clique sets are created:

$\{\{\Delta_1\}, \{\Delta_2, \Delta_3\}, \{\Delta_4\}\}$ and $\{\{\Delta_1, \Delta_2\}, \{\Delta_3, \Delta_4\}\}$. The optimal clique set is defined as the unique clique set where the overall similarity of the contained nodes is maximal. This is calculated and compared to the similarity of other unique clique sets in the function *get-optimal-cliqueset*. Initially, it iterates over a set of unique clique sets. For each unique clique set, if a clique contains more than one Δ, the similarity of all change operations in this clique is calculated, and its sum is divided by the number of total nodes in the clique. For the first clique set containing 3 cliques, the similarity of Δ_2 and Δ_3 is 1.0, which results in an overall similarity of $\frac{1}{3}$. For the second unique clique set containing 2 cliques, the similarity is 0.3 for both cliques, thus, the resulting similarity is also 0.3. Since $\frac{1}{3} > 0.3$, the first clique set $\{\{\Delta_1\}, \{\Delta_2, \Delta_3\}, \{\Delta_4\}\}$ is chosen. Based on this clique set, for each clique, a node is created and the traces are added for creating the child nodes with the change operations in the next recursive iterations. For each change operation in the clique, the corresponding change trace is checked: If the change operation is the last change operation of this trace, the node's counter is incremented; else, the trace is added to the traces, which will be considered in the next recursive call. Thus, all relevant traces and change operations will be considered for the child level. The resulting aggregated change tree is presented in pane 3 of Figure 4.15.

Algorithm 7: Aggregating the most similar change operations

Input:

- Change log $\mathcal{C}$
- Similarity metric $sim(\Delta_1, \Delta_2)$
- change similarity graph $\Gamma := (\mathcal{D}, E)$

Output: Aggregated Change Tree ACT

```
1  Begin root = create-node(null,null);
2  create-children(root, C, 0); return ACT
3  function create-children(parent,traces,position)
4      nodes = ∅, deltas = ∅, cliquesets = ∅
5      foreach trace ∈ traces do
6          deltas = deltas ∪ {trace.get-changeoperation(position)}
7      generate-cliqueset(D, ∅, cliquesets)
8      //variable cliquesets now holds the set of clique sets
9      optimal-cliqueset = get-optimal-cliqueset(cliquesets)
10     //variable optimal-clique set now holds the optimal clique set
11     foreach clique ∈ optimal-cliqueset do
12         //creates a new node containing the change operations and links it to the parent node
13         node = create-node(clique.changeoperations, parent)
14         nodes = nodes ∪ {node}
15         foreach delta ∈ clique do
16             //get-trace-of(delta) finds the trace this change operation is contained in
17             trace = get-trace-of(delta)
18             if trace.get-changeoperation(position+1)==null then
19                 node.count++
20             else
21                 node.nexttraces = node.nextraces ∪{trace}
22     foreach node in nodes do
23         create-children(node,node.nexttraces,position+1)
24 function get-optimal-cliqueset(cliquesets)
25     highest-sim = 0, optimal-set = ∅
26     foreach cliqueset ∈ cliquesets do
27         sim = 0
28         foreach clique ∈ cliqueset do
29             if |clique| > 1 then
30                 //calculate similarity in clique
31                 sim = (Σ_{i,j∈clique,i≠j} sim(i,j)) / |clique|
32         if optimal-set == ∅∨ sim > highest-sim then
33             highest-sim = sim, optimal-set = cliqueset
```

Minimizing the number of nodes The second strategy minimizes the number of nodes in each level of the aggregated change tree. Algorithm 8 shows this strategy. It replaces the function *get-optimal-cliqueset()* with a different algorithm which prefers the unique clique set with the minimum number of cliques. Since each clique is represented by a node in the aggregated change tree, this results in the

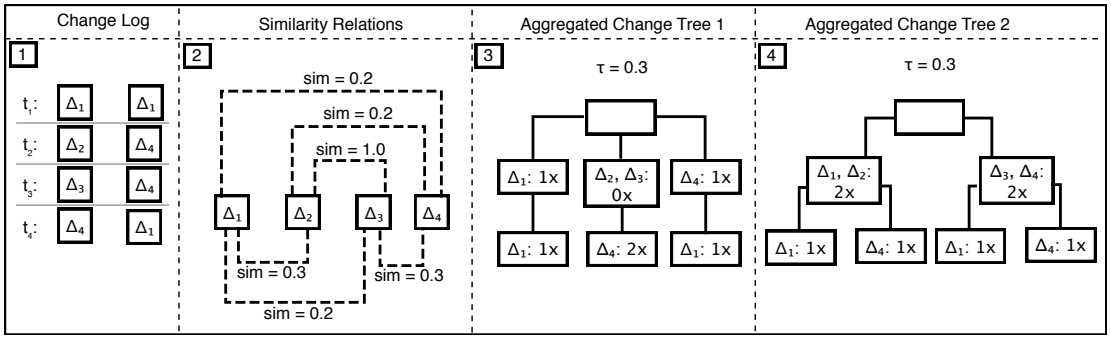


Figure 4.15: Example Change Tree for different strategies

lowest number of nodes *per level*. For the first call of the function in the example given in Figure 4.15, this would return the clique set $\{\{\Delta_1, \Delta_2\}, \{\Delta_3, \Delta_4\}\}$, thus resulting in the aggregated change tree depicted in pane 4.

Algorithm 8: Generating the minimum amount of nodes per level

```
1 function get-optimal-cliqueset(cliquesets)
2     min-count = 0, optimal-set = ∅
3     foreach cliqueset ∈ cliquesets do
4         if optimal-set == ∅∨ min-count < |cliqueset| then
5             min-count = |cliqueset|, optimal-set = cliqueset
```

Discussion: We have presented two strategies for aggregating similar change operations in change tree nodes. Which one should be used highly depends on the structure of the change log: For most situations, aggregating the most similar change operations in one node seems logical. In the nursing home, for example, one could aggregate the most similar therapy adaptations in a single change tree node. If however the resulting number of nodes would become very large due to the structure of the change log, minimizing the number of change tree nodes instead can enhance the tree's readability, thus simplifying further analysis of the change log.

Aggregation for Equal Process Change Operations

For the aggregation of equal change operations ($\tau = 1.0$), no change similarity graph is required, since the equality relation is transitive ($\text{sim}(\Delta_1, \Delta_2) = 1.0 \wedge \text{sim}(\Delta_2, \Delta_3) = 1.0 \implies \text{sim}(\Delta_1, \Delta_3) = 1.0$). The following example explaining Algorithm 9 is based on the change log as presented in Figure 4.12. While the middle pane shows the change tree based on label equivalence as defined in [37], the

right pane shows the resulting aggregated change tree. Assume that $sim(\Delta_1, \Delta_2) = 1.0$, $sim(\Delta_3, \Delta_4) = 1.0$, and all other similarities < 1.0. Algorithm 9 works as follows: First, an empty root node with no label and no parent is created. Using this node the whole change log $\mathcal{C}$, and the position integer which is initialized with 0, the first call of the function `create-children()` is called. The function iterates over all applicable traces (in the first iteration, this is the whole change log). For each change operation which is found at the current position in the trace (Δ), it checks whether a childnode already exists which contains a Δ_2 such that $sim(\Delta, \Delta_2) = 1.0$. Since the equality relation is transitive ($\Delta_1 = \Delta_2 \wedge \Delta_2 = \Delta_3 \implies \Delta_1 = \Delta_3$), only the first Δ of the childnode has to be checked. If the similarity equals 1.0, the change operation is added to the node. If not, a new node is created. For the first iteration, no child nodes exist for the root node, so a new node is created for Δ_1. Since Δ_1 is the last change operation of trace t_1, this node's counter is incremented. Next, Δ_2 on position 0 in trace t_2 is compared to Δ_1: the similarity equals 1.0, so Δ_2 is added to this node. Since trace t_2 contains a second change operation, this trace is added to the set of next traces, which should be considered for recursive calls of the function *create-children()* for the node. When comparing Δ_3 from t_3 to Δ_1 (in the first node), the similarity threshold does not hold, so a new node is created. Since Δ_3 is not the last node of its trace, the trace is added to the set of relevant traces for this node. In the last iteration, Δ_4 is also added to the node containing Δ_3, and the node's counter is incremented. For all traces which are in the set of traces of some node, create-children() is called recursively again with the current node as a parent, the relevant traces, and an incremented position. The function iterates over the children of this node recursively, ultimately creating the tree depicted in pane 3 of Figure 4.12.

Algorithm 9 aggregates equal change operations in a change tree node, i.e. $\tau = 1.0$. It is based on the algorithm that creates change trees based on label equivalence [37].

4.7.3 Discussion: Aggregating Change Tree Nodes

In this section, we discuss alternative metrics for comparing change traces, and describe some corner cases and special properties of the aggregated change tree.

We have presented an approach to aggregate changes over change traces while preserving the inherent structure of the traces with respect to the chosen similarity metric. Other approaches which apply similarity metrics to change traces would be conceivable as well, e.g., by calculating the average of the change operation similarities of all change operations in the given traces. On the one hand, such approaches would be much simpler, on the other hand, the analyzability of their results would be limited. For example, their results could not be used to answer questions which require the structure of the trace to be present, e.g. *What has*

Algorithm 9: Aggregating equal change operations in a single change tree node ($\tau = 1.0$), adapted from [37]

Input:

- Change log $\mathcal{C}$
- Similarity metric $sim(\Delta_1, \Delta_2)$

```
1  Begin
2  root = create-node(null,null,null);
3  create-children(root, C, 0);
4  function create-children(parent,traces,position)
5      nodes = ∅
6      foreach trace ∈ traces do
7          Δ = trace.get-changeoperation(position)
8          node = null
9          //check all aggregation nodes: if one exists where the similarity matches, use it
10         foreach child ∈ parent.children do
11             if sim(child.changes[0], Δ) == 1.0 then
12                 node = child
13                 child.changes = child.changes ∪{Δ}
14         if node == null then
15             node = create-node({Δ},parent)
16             nodes = nodes ∪ {node}
17         if trace.get − changeoperation(position + 1)==null then
18             node.count++
19         else
20             node.nexttraces = node.nexttraces ∪{trace}
21     foreach node in nodes do
22         create-children(node,node.nexttraces,position+1)
24     return;
25 function create-node({Δ},parent)
26     node.changes = {Δ}
27     node.count = 0
28     node.nexttraces = ∅
29     node.children = ∅
30     if parent != null then
31         parent.children = parent.children ∪{node}
32     return node
```

happened after a certain point in the change traces?.

The aggregated change tree preserves the information stored in the change traces, i.e., the general structure of the change traces is kept, although multiple change operations are aggregated in one node. Since any similarity metric which returns a score between -1 (0) and 1 can be used as a basis for aggregation, the presented approach is very flexible and applicable to different questions. The structure, and thus the gist of the aggregated change tree highly depends on the chosen similarity metric, thus, it has to be chosen with care depending on the question at hand.

This information preservation of the aggregated change tree can also be interpreted as a *height preserving property* with respect to the change tree based on label equivalence. Since only change operations are aggregated in one node which reside on the same level of the tree, this is not surprising. Following, two corner cases can be determined: The upper bound for the size of the aggregated change tree is the number of change operations present in the change log (i.e., for each change operation a single node is created). The lower bound is a single path going from the root to exactly one leaf (i.e., all change operations per level are aggregated in exactly one node).

An analysis of the similarity of change traces can provide interesting insights for many situations: In the nursing home, for example, nurses can compare sets of therapies which have been applied to different patients over a long time. Using different similarity metrics, different features of the change traces can be analyzed, e.g., the diversity of required personnel, resources or time. The presented approach can be used with any similarity metric which returns a numeric value within a certain range; thus, it is very flexible and extensible. The presented aggregation strategies allow to either aggregate the most similar change operations, or to minimize the number of nodes, which can be particularly helpful if a compact overview over the change traces is required.

Additionally, aggregated change trees can also be combined with filtering and projection techniques for change trees presented above: After filtering and projecting a change log, the generated change traces can also be used as a basis for change trace aggregation. Hence, powerful application scenarios emerge, which will be discussed in Chapter 6.

4.8 Discussion

This chapter first introduced change trees and n-gram changes together with the associated mining algorithms in order to discover analysis models from change logs that support users in deciding on future change application based on previously applied changes. The benefit of change trees – specifically when compared to existing

change mining approaches, such as change processes – is that they reflect multiple change traces and multiple change occurrences, thus allowing for an analysis of the evolution of process instances from their initial schema up to their current state. In addition, n-gram change trees enable answers to questions such as *'which changes happened after the occurrence of a certain change pattern?'*. This can be very interesting for users, as they do not have to search possibly complex change tree structures containing all the information in a change log, but a "projection" of the trees to the information of interest. Change trees and n-gram change trees have been evaluated in several ways: we compared them to existing change mining techniques and graph based methods, provided a technical implementation and an application to a real-world log.

Next, we presented two extensions of change trees, i.e., combining change trees with filtering and projection techniques, and aggregating change tree nodes based on similarity metrics. For both extensions, we have provided detailed discussions in the corresponding sections.

With these methods for representing and analyzing the chronological order of conducted change operations in process change logs, various analysis questions from realistic environments can be answered: In a nursing home, a nurse may want to know which change operations have been conducted for the therapy instances of patients who have diabetes, are overweight, male, and have sudden problems with their blood pressure. In such a situation, the filtering and projection methods described above can be used: First, all change traces would be filtered, such that the remaining set only contains those change traces corresponding to the process instances of male overweight patients with diabetes. Next, the projection could be used to create a view on the change log, where each trace contains all change operations which have been applied whenever some patient had problems with their blood pressure. Finally, the nurse may want to only focus on those change traces, which have led to the desired effects, e.g., where the patient overcame his problems in a few days. Hence, a second filter could be applied which removes all change traces where the desired effect could not be found in the corresponding execution traces. Based on this information, the nurse can analyze which change operations in which execution order might lead to the desired effects. If the resulting tree is still too large to analyze, she can apply change tree node aggregation, such that the nodes are aggregated based on a chosen similarity metric. For example, if the nurse knows that some resources, e.g., a certain drug or some therapeutic machine, will be available only sparsely in the near future, she can aggregate all change operations where similar resources may be required. Based on this information, she can deduce the best course of action for her current situation.

Similar examples stem from the manufacturing domain: A production manager may want to analyze what has been done for a certain type of product (fil-

tering), whenever a certain problem has occurred, e.g., a supplier did not deliver the required resources on time (projection). Finally, he may want to restrict the projected change traces to those where a solution has only cost a certain amount of money (filtering), and he then may want to aggregate the remaining change traces based on their similarity of the required time (node aggregation).

Overall, the presented representation and analysis methods for process change operations allow for finding answers for a variety of different analysis questions on historic data contained in change logs. In Chapter 6, we will analyze the applicability of these methods in detail. In the next chapter, we will show how the concepts developed in this thesis have been implemented.

Chapter 5

Prototypical Implementation

In this chapter, we evaluate the technical feasibility of this thesis based on a prototypical implementation. Figure 5.1 shows the position of this topic in the overall structure of this thesis. After discussing several representation and analysis techniques for single process change operations, as well as change traces, we introduced the change tree as a data structure and visualization technique for process change logs, which highlights the temporal aspects of process change traces. As a first part of our evaluation, we want to show how these techniques have been implemented and integrated into a framework which can be used to analyze process log files. Hence, in this chapter, we present an implementation of these techniques based on a RESTful microservice architecture.

5.1 Overview

In the previous chapters, we have introduced various representation and analysis techniques for both single change operations, as well as change traces and change logs. For single change operations, we have designed several similarity metrics which can be used to compare multiple process change operations. For change traces, we have developed methods for filtering change traces regarding a boolean condition on the corresponding execution log, and for projecting change operations in traces to cases. Additionally, we have developed the change tree as a basis for representing change logs.

This change tree has then been extended in several ways: First, the *n-gram change tree* can be used to see which change operations have been applied *after* a certain subset of change operations has been applied. Second, the *aggregated change tree* uses similarity metrics to reduce the number of nodes contained in such a tree. Third, the *case-based change tree* uses case-based filtered change logs (as discussed in Chapter 3) as a basis for the data structure. Fourth, the *enhanced*

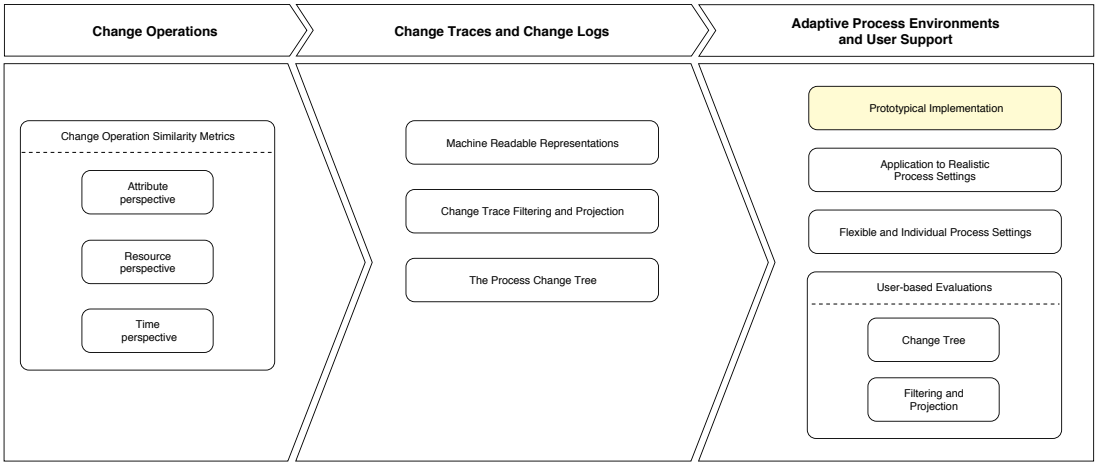


Figure 5.1: Dissertation Overview: Prototypical Implementation

process change tree can be used to enhance the resulting change tree data structure with effects of the conducted change operations which can be deduced from the execution log. In combination, these techniques can be used to provide powerful analysis tools relevant for different domains, as discussed in Chapter 6.

To show the technical feasibility of the approaches presented in this thesis, we have created a prototypical implementation, called *Adaptive Process Environment Support (APES)*. The name stems from the fact that all approaches can be used to support users who interact with adaptive processes. User evaluations will be provided in Chapter 8. In order to make change trees also available to a broader part of the process mining community, we have also created a ProM [38] plugin for representing change trees.

The rest of this chapter is structured as follows: In Section 5.2, we discuss the general design of the various components which are relevant for the APES prototype. Based on the design, we show the implementation of the two main components: The logging server (Section 5.2.1) and the APES server (Section 5.2.2). In this section, we also present the various features of the APES prototype, including similarity metrics for single process change operations and change traces, and features dedicated to change trees. Section 5.3 shows the ProM plugin of our change tree, and Section 5.4 discusses the implementation and concludes the chapter.

5.2 The APES Prototype

The APES prototype[1] consists of several microservices deployed in three different servers, which can be accessed via a HTML based user interface. In this section, we first outline the general architecture of the three servers based on a component diagram, and highlight important parts of the communication based on a sequence diagram. After this introduction, each server and the user interface are described in detail.

The communication between the servers has been implemented based on RESTful web services [63], using standard HTTP methods such as `get`, `post`, `put` and `delete`. Thus, the services can be accessed with any browser supporting those methods, such as Google Chrome or Firefox, or Chrome plugins such as the Advanced Rest Client or Postman. The messages exchanged between the client and the server are represented as XML or JSON, thus providing readability for human users as well as the required structure.

Figure 5.5 summarizes the architecture of the APES prototype as a UML component diagram. The following servers are relevant:

- The *CPEE*[2] [64] is the process engine used throughout this work. It provides REST-based interfaces to create, change, read and interact with process instances, and has been developed in the research group *Workflow Systems and Technology*[3] at the University of Vienna.

- The *Logging Server* communicates with the CPEE over this REST protocol: Whenever a process instance is changed in the CPEE, or an activity is executed by the CPEE, the logging server is notified about this event, and creates an entry in the corresponding process change or execution log. Details about the logging server can be found in Section 5.2.1.

- The *APES Server* represents the core component of this prototypical implementation: All approaches developed in this thesis have been implemented, and can be accessed using a RESTful web service interface. The APES server uses data provided by the logging server as a data base. Section 5.2.2 provides a more detailed explanation on the different services of the APES server.

- The *User Interface* has been developed as a web based interface, using HTML, CSS and JavaScript. Thus, it can be accessed from any modern browser from a plethora of different devices, including PCs, tablets and

[1] available on our project homepage `http://cs.univie.ac.at/project/apes`
[2] `http://www.cpee.org`
[3] `http://wst.cs.univie.ac.at/`

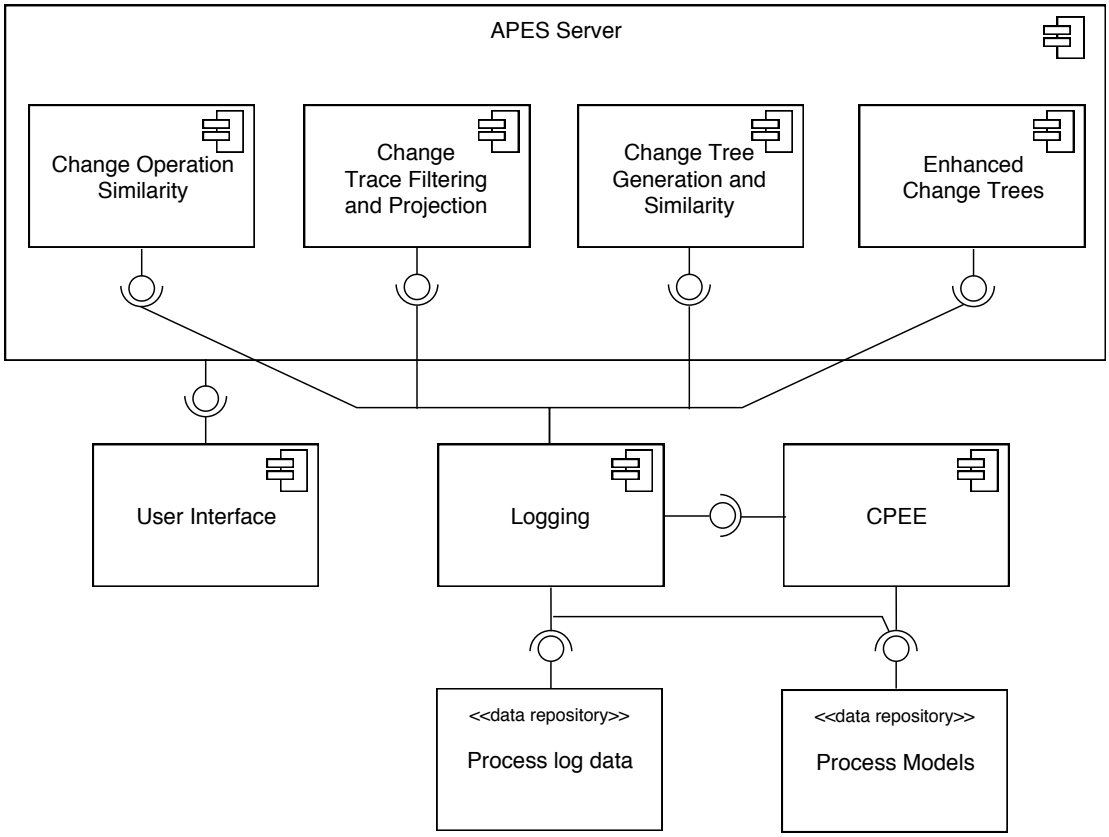


Figure 5.2: Architecture of the APES Prototype

smartphones. It solely communicates with the APES server, which prepares the data obtained from the logging service depending on the query of the user interface. Section 5.2.3 shows basic concepts implemented in the user interface.

Each server has one dedicated job: Thus, by decoupling the microservices into different servers, exchanging and scaling of services when required is very simple.

Figure 5.3 shows a sequence diagram of the communication between the different components:

First, whenever any kind of event happens in the process engine (executing an activity, or applying a change operation to a process instance), the logging server is informed, and creates a log entry in the respective log file. This log file is now exposed to other services via its own API (see Section 5.2.1 for more details). Whenever a user interacts with the APES server (which provides interfaces for all approaches developed in this thesis), he does so using the web based user interface. Depending on the required information, the data is then retrieved from the logging

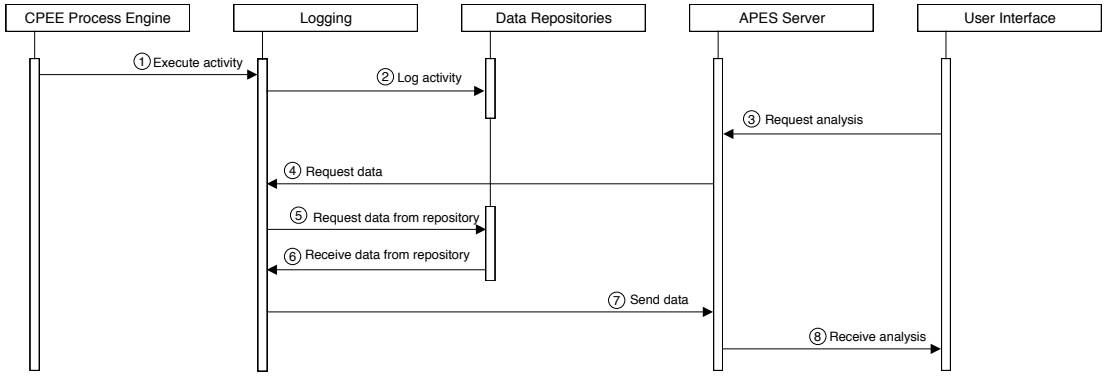


Figure 5.3: Sequence Diagram of the Components Interacting in the APES Prototype

server. For the change tree, for example, the APES server would require the change log only; for the EPCT, the execution log would be required as well.) The logging server now retrieves the respective data from the data repository, and sends it back to the APES server. Based on the implemented algorithms and approaches, the APES server now generates a reply, which is sent back to and displayed by the user interface.

In the remainder of this section, we describe three main components in more detail: The Logging Server (Section 5.2.1), the APES server (Section 5.2.2) and the user interface (Section 5.2.3). A detailed explanation of the CPEE can be found in [64].

5.2.1 Architecture of the Logging Server

The logging server provides the data required for the representation and analysis algorithms implemented in the APES prototype: It generates change and execution logs by listening to events provided by a workflow engine. In this prototypical implementation, the workflow engine chosen is the CPEE [64], since it is very flexible and easily extensible. However, by decoupling the implementation of the logging service from the actual workflow engine (i.e., the CPEE), the APES prototype gains additional flexibility: The process engine can easily be exchanged for any different engine, as long as the interfaces to the logging server are kept consistent.

Another advantage of decoupling the logging server from the implementation of the approaches developed in this thesis is the fact that it can easily be reused in other services and projects: For example, projects like our process-based nursing technology project ACaPlan [65], (cf. Section 6.3.1) can also use the logging

server as a basis to show which change operations have been conducted, and which therapy steps have already been executed, without being forced to use the representation and analysis forms present in the APES prototype.

Figure 5.5 shows the basic architecture of the logging server. Each process model and log file present in the logging server belongs to exactly one domain. Examples for domains are the nursing domain, the financial domain (cf. Section 6.2.1), or the wellbeing domain (cf. Section 6.3.2). By encapsulating information by the domain they belong to, information can be separated, thus increasing understandability of the data: If one wants to access all data belonging to the nursing domain, he can do so very easily - using the API provided, there will be no overlap with data stemming from any other domain.

For each domain, three main services exist:

- **Fragments** contains all process fragments which have been added, removed, or in any other way affected by process change operations *in any process model* of the chosen domain. By separating fragments by the domain they have been applied in, one can make sure to only see the fragments relevant for the specific domain (e.g., the financial domain), without being distracted by fragments from any other domain. While the service `fragments` returns a list of all fragments available in the chosen domain, `{fragment-id}` returns the details of one specific fragment in this list.

- **Models** returns a list of all *unchanged* process models present in the chosen domain. As for fragments, the top level service `models` returns a list of all available models, while `{model-id}` returns the details of one specific model. For each model, the instances which have been created based on this model can be inspected: By calling the `instance` service, a list of all created instances is shown. `{instance-id}` shows the details for one specific instance. Additionally, for each instance, the change and execution traces can be inspected. These can also be seen for all instances of the model in one log file, by calling the `{model-id}` service with the respective parameters.

- **Environments** returns a list of all environments present in the current domain. An environment is an additional data container for domains: For each domain (e.g., nursing or manufacturing), multiple different environments can be defined, whereas data belonging to one specific environment can be viewed separately from data belonging to other environments. For the nursing domain, an environment would be one specific nursing home. If the logging server receives events from two different nursing homes, those events would be connected to two different environments in the same domain. Using environments in this domain, one can see all change operations which have

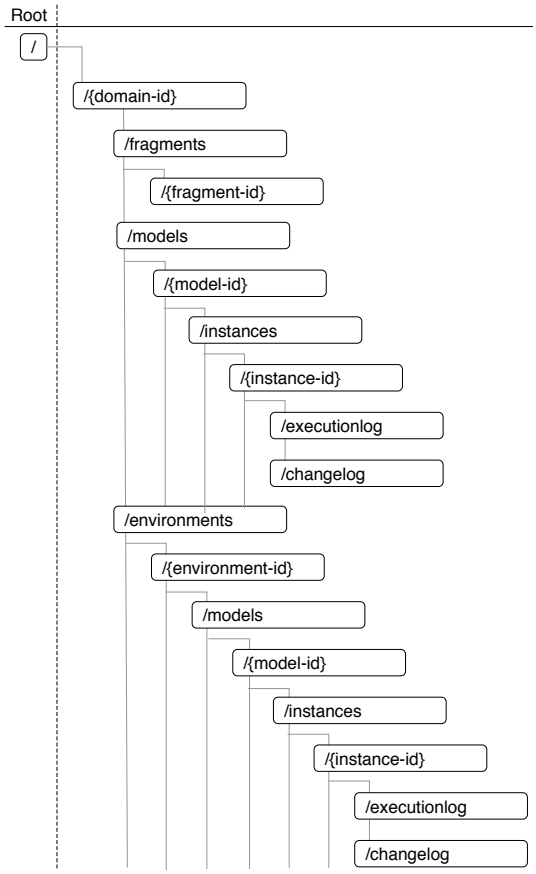


Figure 5.4: Architecture of the Logging Server

been applied in one specific nursing home, thus analyzing possible differences between nursing homes (i.e., between different environments). Just as for the domain, for each environment all models with their respective process instances, execution and change logs can be queried. Thus, one can see exactly which models have been relevant in a specific environment, which changes have been applied, and which events have been executed.

The architecture of the logging server is generic, thus it can be applied to a variety of domains. In this prototype, it is used as a data basis for the representations and analysis forms implemented in the APES server. The next section discusses, how this server has been implemented.

5.2.2 Architecture of the APES Server

The APES server contains the prototypical implementation of the various approaches for representing and analyzing process change operations and change logs which have been presented in this thesis.

The RESTful architecture of the APES server is depicted in Figure 5.5. The basic structure is similar to the structure of the logging server: For each domain, the models, the environments and the change operations can be accessed and analyzed. Domains work the same way they do for the logging server; thus, a separation between different domains such as the nursing or the financial domain on the same server is easily achievable. For each domain, the following services exist:

- **Models:** As for the logging server, the `models` service returns a list of all process models available in the current environment. For each model, the change tree can be inspected, and a list of instances can be shown. These services will be explained in more detail below.

- **Cases:** For each domain, it is possible to define a set of cases and filtering conditions which can become relevant while process activities are being executed. Since information about these cases has to be defined once, it is part of the APES server, instead of the logging server, which only focuses on retrieving execution and change log information from the respective process instances.

- **Environments** work the same way they do for the logging server: Basically, all services available for the `models` can also be restricted to only include those models who have been enacted in a certain environment (i.e., in a certain nursing home, special needs schools, or financial institution).

- **Changes:** We have developed different change operation similarity metrics for comparing two different change operations regarding different features, such as their required resources, time and resources, or regarding their attributes. This service provides this functionality.

- **Statistics** return a set of statistics for the chosen environment, such as how many change operations exist, which models exist, how many instances exist per model.

Throughout this thesis, several representation and analysis approaches for both single change operations, as well as change traces and change logs have been discussed. In the remainder of this section, their technical feasibility is presented.

Chapter 2: Change Operation Similarity: In Chapter 2, we have presented different metrics for comparing multiple process change operations. The *change resource similarity (CRS)* can be used to compare two change operations regarding their required resources, the *timed change resource similarity (TCRS)* also adds the time perspective. Using *attribute similarity* metrics, one can compare two change operations regarding their attributes, such as their position, fragment, schema or change operation type. These change operation similarity metrics have been implemented in the service `{domain-id}/changes`.

Chapter 3: Change Traces: In Chapter 3, an approach for filtering change traces has been presented. Using information from the execution log, change operations from change logs can be filtered: For example, only those change operations which belong to patients who are male and above a certain age should be part of a change log. This can be done by calling the `{domain-id}/models/{model-id}` (thus filtering all change logs from the domain), or the `{domain-id}/environments/{environment-id}models/{model-id}` service (filtering only change logs which belong to a certain environment) with the appropriate set of parameters. Additionally, in this chapter we have presented an approach for projecting process change operations in a change log to cases. By calling the `{domain-id}/models/{model-id}/cases/{case-id}`, or the respective service in the `{environment}` subtree of the service architecture, such a mapping can be requested from the server. Additionally, we have shown how change traces which have already been projected to cases can be filtered: This service can be found under the resource `{domain-id}/models/{model-id}/cases/{case-id}/{filtering}`.

Chapter 4: Change Trees: In Chapter 4, change trees were introduced as a data structure for representing process change logs. In addition to the basic data structure, we have also presented *n-gram change trees* (Section 4.4), case-based change trees (Section 4.6.1), and enhanced process change trees (Section 4.6.2). These data structures can all be accessed by calling either `{domain-id}/models/{model-id}/changetree`, or the respective service constrained to a certain environment with the respective set of parameters.

Additionally, we presented an approach to aggregate change operations in change trees not based on their label, but by applying process change operation similarity metrics instead. Just like all other services which are related to the change tree, such a tree can be created by either calling the service available at `{domain-id}/models/{model-id}/changetree`, or the respective environment service from the APES server.

The APES server is the main building block of the prototypical implementation: Based on data obtained from the logging server, the APES server provides the proof

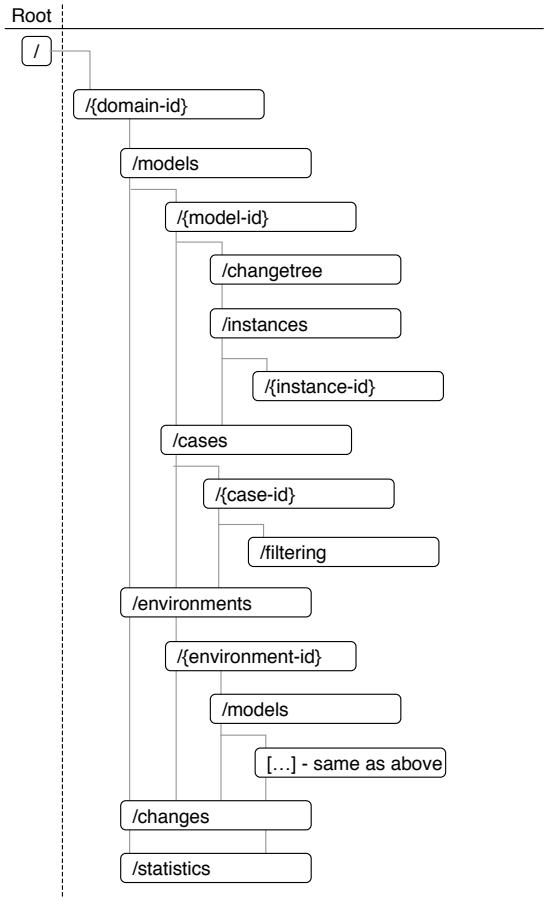


Figure 5.5: Architecture of the APES Server

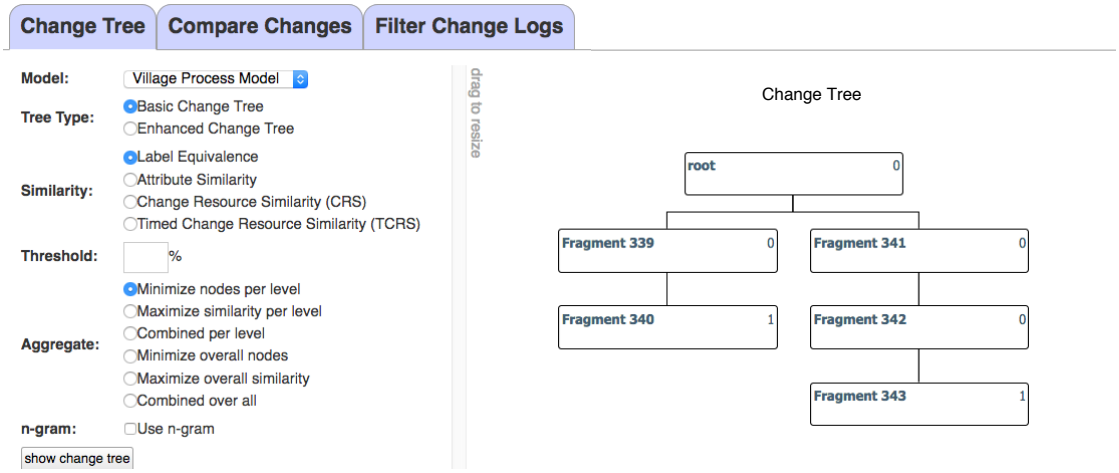


Figure 5.6: APES User Interface: Change Tree

of technical feasibility for the different representation and analysis tools presented in this thesis. While the data provided by the APES server can be used directly for further analysis (e.g., by other services which build upon this data), it has to be visualized for being directly accessible to users. Thus, we have developed a user interface using HTML, JavaScript, and CSS, which is shown in the next section.

5.2.3 User Interface

By providing RESTful interfaces, the logging and the APES server both can be called from various user interfaces, including smartphone or tablet apps, and web based interfaces, as well as other applications or projects, such as ACaPlan. In this section, we present such a user interface, namely a HTML5 based web implementation. Such a web interface calls the services discussed in previous sections, and presents them to the user using widespread libraries such as `jquery`[4].

Figure 5.6 shows the user interface for visualizing the change tree, the aggregated change tree, the enhanced process change tree, and the n-gram change tree. Users can choose between different similarity measures (presented in [16]), as well as the change tree presented in [37] by choosing *label equivalence* as a similarity measure. Additionally, they can choose to include filtering conditions (by selecting the *Enhanced Process Change Tree*). In this example, the labels of the nodes in the change tree reflect the names of the fragments which are used by the adaptations, but any other label would be possible as well.

Figure 5.7 shows the interface for calculating process change operation similar-

[4] `https://jquery.com://jquery.com/`

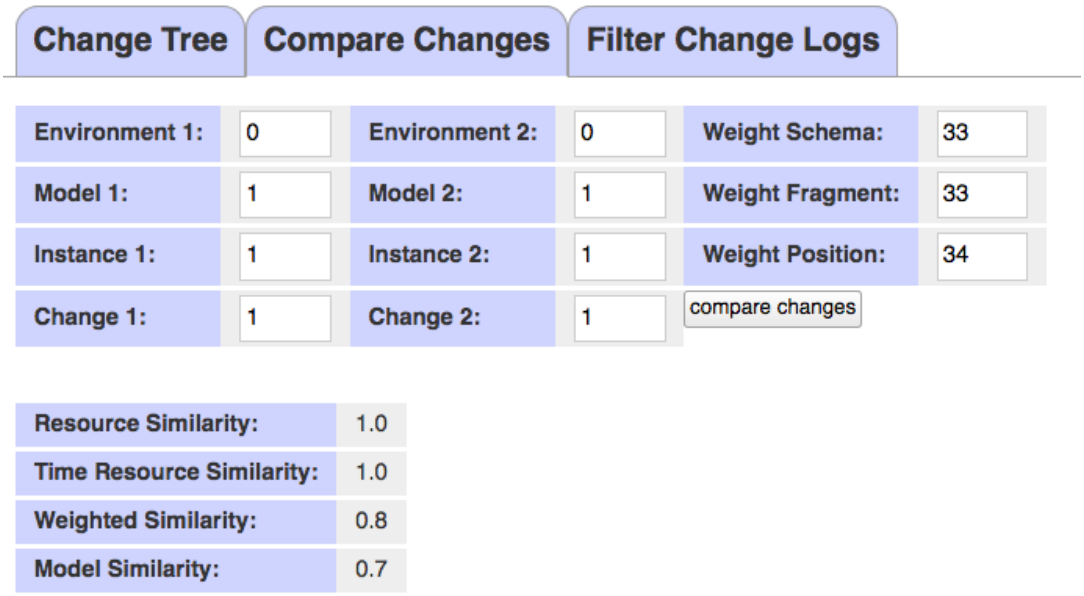


Figure 5.7: APES User Interface: Change Similarity

ity measures. Based on the position of two change operations in the change log, the different similarity metrics presented in this thesis are calculated and presented to the user. For calculating the *attribute similarity* (cf. Section 2.4), weights for the different attributes are necessary. These can be entered in the respective input fields. Once the user presses the *compare changes* button, the table on the bottom of the screenshot is presented, which shows the score for each similarity metric.

Figure 5.8 shows the interface for filtering change logs based on values from the corresponding execution log: On the left side, the user can enter a set of rules by using key value pairs, which are compared by standard boolean operators, such as

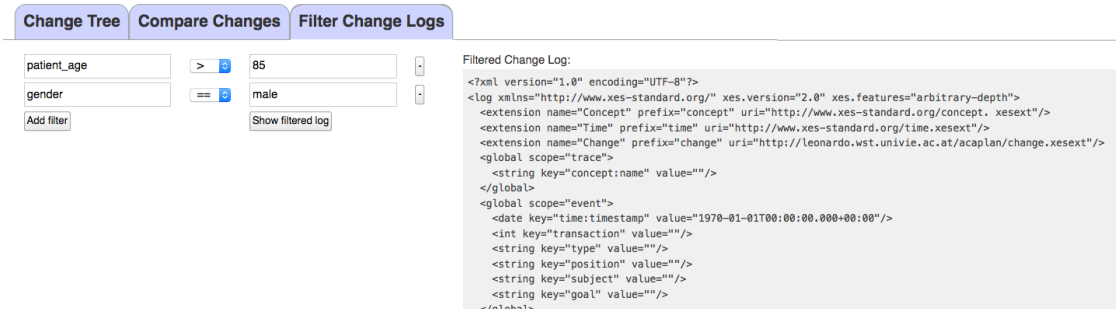


Figure 5.8: APES User Interface: Filter Change Logs

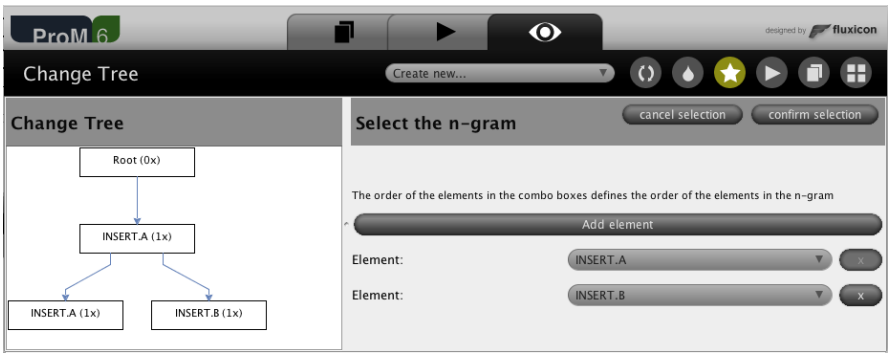


Figure 5.9: Implementation of the change tree as a ProM plugin

`greater than`, `equals` or `lower than`. For example, in the nursing domain, the user can say that only those change logs should be presented where the patient's gender is male, and he is above 85 years old. New rules can be added by pressing the *Add filter* button, and removed by pressing the - button next to each rule. By pressing the *Show filtered log* button, the filtered change log in XES is shown to the right.

The user interface implements the most important features of the APES prototype, thus showing the technical feasibility of the concepts presented in this dissertation. While some features are shown in individual tabs, such as filtering change logs or comparing change operations, most features which have been used in combination with change trees, including case-based change trees and change logs, can be accessed from the *Change Tree* tab.

In the next section, we present the ProM implementation of the change tree.

5.3 ProM implementation

In addition to our web based implementation, the basic change tree was also implemented as a plugin for the process mining framework ProM [38]. The goal of this implementation was to provide researchers who are familiar with this framework and process change mining in particular with the possibility to explore the advantages of using change trees for analyzing process change logs, while using a familiar user interface[5].

Change logs can be provided both in MXML [9], as well as in XES, as discussed in Section 3.2. Based on these logs, both change trees as well as n-gram change trees can be created (cf. Figure 5.9).

[5]The current version of the plugin is available as a nightly build at http://www.promtools.org/prom6/nightly/

However, most of the representation and analysis methods presented in this thesis have been implemented in the APES prototype only. The reasons for this decision are manifold: First, by means of the HTML based user interface, the prototype can be accessed from any device with a browser (including smartphones, tablets and PCs), thus providing additional flexibility to the users. Second, since the APES prototype works in the browser, users do not need to download or update their implementation, when a new version is available. Third, by means of the RESTful API provided by APES, it is possible to include the prototypical implementation directly in any project which can access such APIs. For example, the game-based experimentation setting *Apelands* (presented in Section 7) directly accesses some of the features provided by APES. For these reasons, we decided to only provide the change tree to the nightly builds of the ProM framework, and implement the other concepts presented in this thesis in an autonomous, dedicated prototype, which provides more flexibility.

5.4 Discussion

In this chapter, we have presented the proof of technical feasibility for the various concepts developed throughout this thesis. By implementing (a) a REST based server application, which can be accessed from various devices without the requirement of installing client side software, the user can access and test various representation and analysis forms for change operations and logs. In addition, we have presented a ProM plugin, which can be used by researchers familiar with the ProM framework, in order to get a first view on change trees. In the next chapter, we will discuss different application settings for process flexibility in general, and for the discussed approaches in particular.

Chapter 6

Application

Throughout this thesis, we have presented several representation and analysis approaches both for single process change operations, as well as change traces and change logs. In the last chapter, we have focused on the proof of technical feasibility by means of a prototypical implementation. While this implementation shows that all concepts developed in this thesis can be used to analyze process change logs, it does not show which information can be gained when doing so. Hence, in this chapter, we present the proof of applicability: The concepts developed in this thesis are applied both to real world data sets, as well as to possible tools and user interfaces for realistic scenarios. Thus, we demonstrate what the presented approaches can be used for in realistic environments. This chapter is based upon the papers *"ACaPlan - Adaptive Care Planning"* [65], *"The NNN Formalization: Review and Development of Guideline Specification in the Care Domain"* [59], and *"Multidimensional Process Mining: Questions, Requirements, and Limitations"* [58]. Figure 6.1 shows the position of this topic in the structure of this thesis.

6.1 Overview

Process adaptations are a key component for the flexibility of realistic process settings. In this thesis, we presented several representation and analysis techniques both for single process change operations, as well as for change traces and change logs. In this chapter, we analyze how the concepts developed in this thesis can be applied to realistic process settings. Here, two perspectives are relevant: First, it would be interesting to see how the techniques developed in this thesis can be applied to existing log files from realistic process settings. Thus, we gain additional insights into possible application scenarios. Second, an overview over the possible features of tools and user interfaces tailored to a specific domain which integrate the

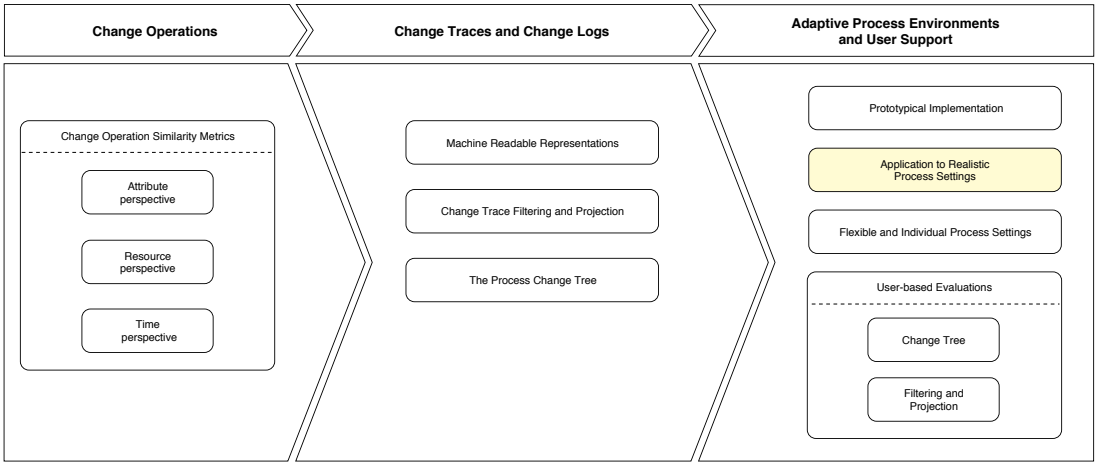


Figure 6.1: Dissertation Overview: Application

concepts developed in this thesis would provide a greater picture: For the nursing domain, for example, it would be interesting to see what nurses and doctors who have to plan and adapt therapy process instances of patients can actually gain from the various representation and analysis techniques. Overall, this chapter tackles the following research questions:

Q1: How can the concepts developed in this thesis be applied to existing process log files from realistic domains?

Q2: How can we integrate the representation and analysis approaches into tools and user interfaces, which can be used by practitioners in a certain domain?

These research questions are tackled as follows: First, we address the analysis of existing process log files (Section 6.2, $\rightarrow$ Q1). For this, we take a closer look at the financial domain (Section 6.2.1), the medical domain (Section 6.2.2), as well as the educational domain (Section 6.2.3). Next, we analyze how the representation and analysis techniques as discussed in this thesis can be integrated into tools and user interfaces, thus supporting users who work in a certain domain (Section 6.3, $\rightarrow$ Q2). For these application scenarios, we take a closer look at the nursing domain (Section 6.3.1) and the wellbeing domain (Section 6.3.2).

By providing both perspectives, we show that the concepts developed in this thesis can both be used for inspecting existing process data, as well as a basis for creating new innovative tools and user interfaces. Finally, Section 6.4 summarizes and concludes the chapter.

6.2 Application for Log Analysis

In this section, we analyze how the concepts developed in this thesis can be applied to existing process log files from realistic domains. The application of concepts like the change tree is not necessarily limited to change logs - execution logs can be interpreted using such instruments as well. Since both log file formats are represented as traces containing a temporally ordered set of events, the standard techniques for creating and updating a change tree applies to execution logs as well.

In this section, we first discuss how change trees can be applied to execution log data, stemming from the financial domain. Next, we discuss the application of change trees in combination with multidimensional process mining [58], a technique which allows to create multiple views on a change log.

6.2.1 Application to the Financial Domain

During the Business Process Intelligence (BPI) Challenge, real world process logs are analyzed from various points of view. The BPI Challenge 2014[1] was based on data from different processes of Rabobank Group ICT, i.e., from the financial domain. When inspecting the log files we found that one of these processes is suited to be analyzed by the change tree since its log provides a set of activities which describe what has happened to a specific item in order to solve some problem: The change records of Rabobank include a total of 30275 activities which describe changes to Rabobank's change management process. These activities are the basic building blocks for our change tree: Each time a new activity is planned for a specific item, the process of this item changes. Thus, the change tree can be used to mine change information from these log files.

Since the execution log contains more than 30.000 events, an analysis of the whole log with the change tree would return a tree too big to be visually conceivable. Thus, we decided to constrain the log to data which might be of interest for certain analysis questions. However, since the change tree is built upon execution logs, the filtering and projection techniques presented in Chapter 3 cannot be applied here. Thus, by mining process logs with the n-gram change tree, we want to analyze what has usually happened after a certain change or a set of changes. For example we found multiple repeating process steps after including "`Standard Change Type 88`" (SCT 88) into the process. This is reflected by the change tree depicted in Figure 6.2, which has been generated with the ProM [38] plugin described in Section 5.3 as part of our prototypical implementation. Specifically,

[1] http://www.win.tue.nl/bpi/2014/challenge, doi: `10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35`

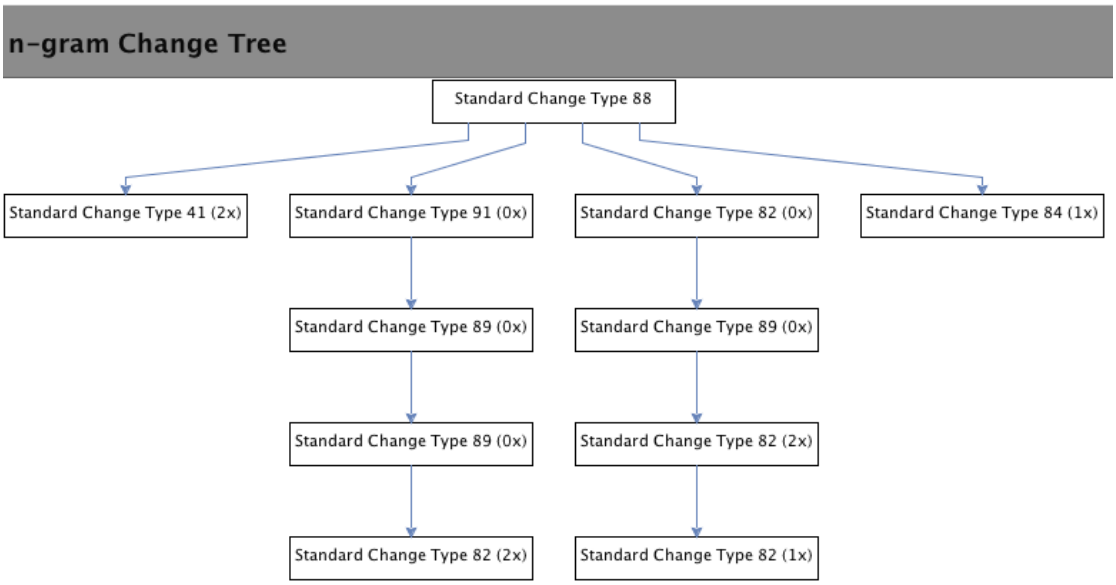


Figure 6.2: Process steps following Standard Change Type 88

the tree is a 1-gram change tree with 1-gram "`Standard Change Type 88`". Using such a change tree, one can see which operations have been applied after a certain event. When inspecting the data, the 1-gram change tree for `Standard Change Type 88` revealed interesting features for further analysis:

From the 1-gram change tree it can be concluded that change "`Standard Change Type 88`" was followed in most cases by a change sequence containing "Standard Change Type 82" or "`Standard Change Type 41`". Since the provided data lacks context information it is not possible to deduce any semantic information from the generated change tree. If more information was available one could predict certain requirements for future tasks based on the change tree. Think for example of knowledge about resources required for a specific task, such as requiring a technical specialist for executing "`Standard Change Type 82`". Using the n-gram change tree as a resource planning instrument it can be predicted that after executing change "`Standard Change Type 88`", with a certain probability "`Standard Change Type 82`" will become necessary as well and a technical specialist will be required (even if not required for "`Standard Change Type 88`" in the first place). Based on the analysis of the 1-gram change tree, additional analysis using, for example, 2- or 3-grams would be conceivable: If the 1-gram change tree contained too much information to be visually conceivable, one could subdivide this tree for example into multiple 2-gram change trees. For the example given here, the 2-gram `(''Standard Change Type 88'', ''Standard Change Type 82'')` would reduce the n-gram change tree to the one branch starting with

the event ‘‘`Standard Change Type 82`’’. Hence, users who analyze the change log can focus on those parts of the change tree which are of interest for the current analysis question.

In this example, we have used the n-gram change tree as a basis for analyzing an event log, which does not reflect process change operations. However, conclusions regarding the data set this representation is based on can be drawn using such an approach: We saw, how many events of a certain type have been applied after the event `Standard Change Type 88` has occurred. Hence, the change tree is not only limited to the analysis of process change logs, but can also be applied to the analysis of different event logs.

6.2.2 Application to the Medical Domain

In the last section, we have analyzed an event log from the financial domain using the n-gram change tree. Using n-grams, we have constrained the events present in a change tree to those of interest for a certain analysis question. We have shown that change trees can not only be applied to process change logs, but to other event logs stemming from flexible processes as well. We have reduced the data represented by the change tree by means of n-grams, since the filtering and projection techniques presented in Chapter 3 focus on data present in both process change and execution logs. In this section, we combine change trees with *Multidimensional Process Mining* (MPM) [58], another concept for reducing the information present in an process log. The general idea of MPM is to combine classic process mining techniques with OLAP operations (Online Analytical Processing [66]) for multiple dimensions. Thus, a process log can be viewed from multiple perspectives, which allows for additional ways to analyze process logs.

In the given example, we use event log data from the medical setting in German hospitals as a data basis, which has been constrained by MPM techniques to event data stemming from a surgical department. In Figure 6.3, an excerpt of the generated change tree from this MPM based change log is shown. It depicts the activities which have been executed before a surgery, including analysis from the radiology department, and pre-surgical lab tests. Using this change tree as a basis, one can analyze historic execution data, e.g., with respect to outlier data: For example, the change tree contains the activity describing the radiology report on multiple positions in the execution log. A domain expert could now use this insight to analyze whether such positions should be possible, or if they hint at wrong or missing data, or, in the worst case, procedures being executed wrong in the surgery department. Thus, if a change tree shows that certain activities have been executed in a process, before it should be possible to add them (e.g., the end of some procedure before it has even started), such things can be easily detected using the change tree representation. Additionally, multiple occurrences, like in

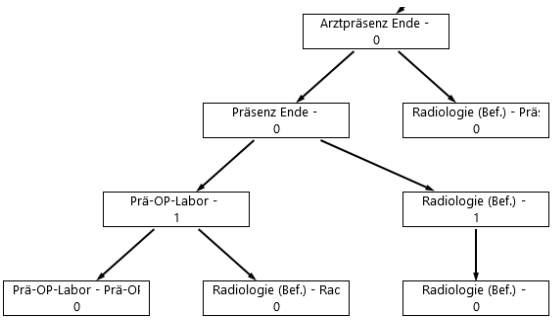


Figure 6.3: Application of MPM Change Trees to a Medical Setting

the change tree depicted in Figure 6.3, can be easily spotted and analyzed.

In the last section, we have shown that n-gram change trees provide a basis for constraining the data contained in a process change tree to answer certain analysis questions in realistic environments. In this section we have shown a different method for constraining such change trees: By means of MPM techniques, analysts can create different views on an event log, which can then be analyzed using process change trees. Again, we have applied process change trees to event logs. Although change trees were developed for process change traces, the application to event logs provided additional insights to the data.

6.2.3 Application to the Educational Domain

In the last two sections, we have shown that change trees can be also used as a representation form of process execution logs. Using n-gram change trees or multidimensional process mining techniques, we have constrained the execution logs the change trees were based upon, such that the resulting change trees can be used to gain additional insights into the domain data. In this section, we show that these two approaches can also be combined, such that the resulting data can be inspected for further analysis.

In this application analysis, we have applied change trees, in combination with n-grams *and* MPM techniques to a data set from the educational domain, i.e., the *Higher Education Processes*[2] (HEP) data set, which has been used as a data set for many papers [67, 68, 69], and contains 28.129 events in a total of 354 process execution traces. These traces describe activities students have done during multiple courses, such as taking a tutorial, uploading submissions, or evaluating milestones.

Figure 6.4 shows an excerpt of the application of n-gram change trees to the higher education processes data set. The execution log this tree is based on has

[2]HEP data set, available at `www.wst.univie.ac.at/communities/hep/`

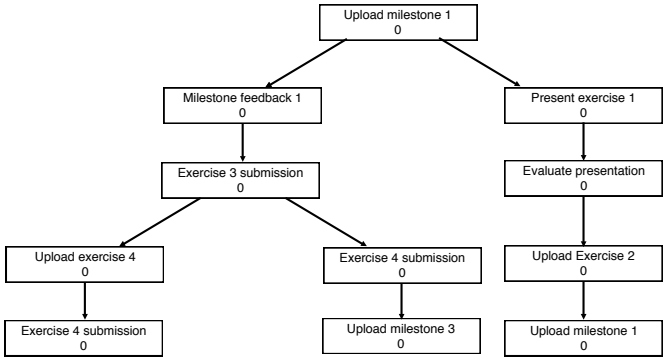


Figure 6.4: Application of MPM Change Trees to Higher Educational Processes

been constrained by MPM techniques such that it only contains those activities which belong to one specific course. Additionally, using n-gram change trees, we have analyzed which execution events have happened after the activity `Upload milestone 1`. Using such a change tree as a basis, analysts can reconstruct which activities most students have executed after they had completed the respective milestone. Such an analysis of student behavior can then be used to refine the structure of the course, or highlight possible problems the students encounter. Hence, when combining multidimensional process mining, and n-gram change trees, even more detailed questions can be answered in realistic data sets.

Conclusion: Overall, in this section we have shown that change trees can also be applied to execution logs. However, filtering and projection techniques, as introduced in Chapter 3, cannot be applied here, since these techniques require both change and execution logs to be present. As an alternative, we have shown that existing methods such as MPM can be used to constrain the data in a change log to those parts which are relevant for a certain analysis question. While this section has focused on application of the change tree to real world data sets, the next section, we will show how all representation and analysis techniques presented in this thesis can be applied in tools and user interfaces.

6.3 Applications as Tools and User Interfaces

The application of the concepts developed in this thesis as a basis for tools and user interfaces for users, who are not necessarily experienced with process adaptations from a technical point of view, provides an interesting setting: Information which can be gathered by analyzing process change- and execution logs has to be preprocessed in order to be of use for this group of users. In this section, we analyze how

the concepts developed in this thesis can be integrated into user interfaces to be of use for users from such application domains. For this, we analyze the nursing (Section 6.3.1) as well as the wellbeing domain (Section 6.3.2).

6.3.1 Application to the Nursing Domain

Applying and managing processes, especially flexible processes where change operations are necessary on a regular basis in the context of the nursing domain has been tackled by the Adaptive Care Planning (ACaPlan) [65] research project, conducted at the University of Vienna. Its goal is to ease the burden of care personnel during their daily routine with the patients in a nursing home. Usually, patients are seen in a data centric way: For each patient, there exists a folder (which can either be a real folder, or a digital one), containing all data related to him. This data contains among others his anamnesis, medical history, and all therapy activities which have to be done by the nurses during his stay in the nursing home. The execution of each of these activities has to be documented thoroughly by the nurses. Since nurses have to take care of their patients during their whole shift in the nursing home, this documentation can only be completed during the night; after the end of the shift. Thus, the nurse has to remember each therapy activity she has conducted during the whole day in order to comply to national and European laws.

The goal of ACaPlan is to ease the burden for care personnel by (a) providing means for planning treatments, (b) remembering care personnel about upcoming activities, and (c) conduct the documentation semi-automatically; thus allowing for documenting conducted activities on the fly while the patients are being treated by the nurse [70]. To provide such user support, all data related to the patients and the resulting therapy plans is handled by process instances. This section first shortly introduces the basic concept of ACaPlan, gives a short overview over existing user interfaces, and shows how the concepts discussed in this thesis can easily be integrated into the user interface in order to provide additional user support.

A Process-based Representation of Patient Data

In ACaPlan, the therapy plan of each patient is seen as a process instance which gets adapted whenever a patient requires some change to his personal therapy plan. Thus, for each patient, one individual therapy process instance exists (cf. Chapter 7). Figure 6.5 summarizes the therapy process of a patient: In the top pane, the general process schema which is instantiated for each patient is depicted: Each patient's individual therapy process instance evolves from this schema. It contains only one task, *First anamnesis*, during which the doctors and nurses

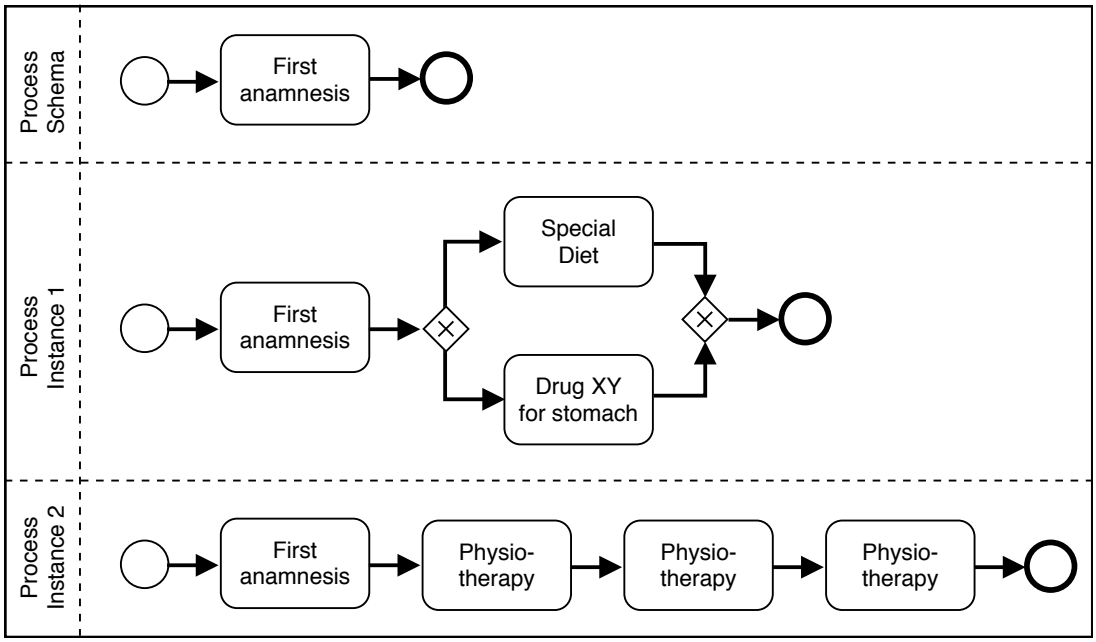


Figure 6.5: Process Concept of ACaPlan

should analyze which kinds of treatment the patient requires. Then, based on his individual requirements, adaptations to the process instance are conducted; i.e., newly predefined process fragments are added or removed from the process instance's schema. This is depicted in pane two and three: Here, for two patients, two process instances have been created and adapted in different ways: While the patient of process instance 1 requires a special diet, and some drugs for his stomach, patient 2 requires some kind of physiotherapy.

This process-based concept stands in contrast to the way many hospitals and nursing homes view patients [65]: Usually, they are seen as a heap of data, which contain the information about the patient, about prior and upcoming therapies, and about the patient's environment. By shifting this view to a process-based approach, concepts and approaches existing in state of the art process research, which are also partly discussed in this thesis, can be applied to this domain; thus improving the way nurses interact with their patients and plan corresponding therapies.

Application of Change Mining

In this section, we will give an overview on how *ACaPlan* supports nurses using flexible process instances, and change mining based on change and execution logs.

Whenever a change in the patient's therapy is required, the nurse has to de-

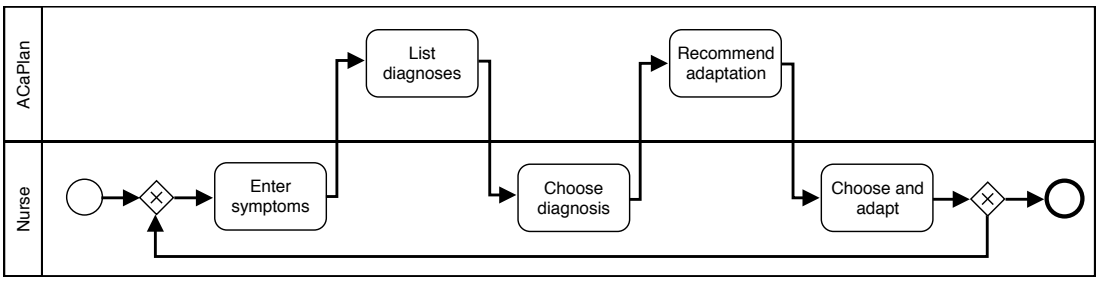


Figure 6.6: Creating new therapies based on a standardized process loop

cide on a change operation which should be conducted. While it is possible to create new change operations, a repository of prior change operations exists as well. In order to decide which change operation to choose, the nurse has to follow a predefined process, which is depicted in Figure 6.6. These steps are based upon the taxonomy developed by the *North American Nursing Diagnosis Association* (NANDA), the *Nursing Interventions Classification* (NIC) and the *Nursing Outcomes Classification* (NOC) [59]. These three taxonomies, which are called NNN for short, define several possible diagnoses, including possible outcomes for the patient, and interventions which can be performed to reach more desirable outcomes (i.e., therapies conducted with the patient).

As a first step, the nurse initiates a new iteration of the nursing cycle by entering information about the patient's symptoms. For each diagnosis in NANDA, a set of symptoms exist which describe this condition. Thus, based on a list of given symptoms, and the NANDA data set, ACaPlan can now show the nurse a list of probable diagnoses. This is visualized in Figures 6.7 and 6.8: Here, the nurse enters the information *The patient feels tired*, and in result receives three possible diagnoses, ranked from the most probable one to the least probable. In ACaPlan, this system is implemented as a simple string-based pattern matching: If the string received as input from the nurses matches any string in the NNN data set, the respective diagnosis is added to the list of possible diagnoses. The more matches are found, the higher the resulting ranking is.

After a diagnosis has been selected, ACaPlan suggests possible change operations to a patient's process instance. Figures 6.9 shows a basic interface for selecting a therapy, which is then inserted into the patient's therapy plan. As soon as the inserted activities become active, they are displayed for the nurse in her individual worklist for this one patient, as visualized in Figure 6.10.

Based on the choices of the nurse regarding the symptoms and the diagnosis, ACaPlan recommends possible adaptations to the nurses. Here, the concepts developed in this thesis come into play: Based on information present in NANDA, NIC and NOC, and information about the patients, ACaPlan can be enhanced to

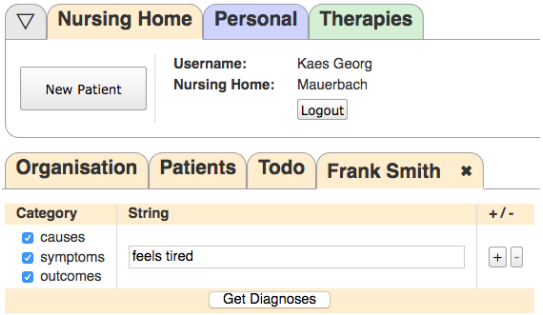


Figure 6.7: Selection of Symptoms in ACaPlan

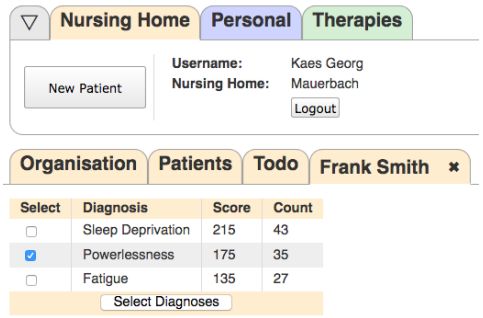


Figure 6.8: Selection of Diagnoses based on the Symptoms in ACaPlan

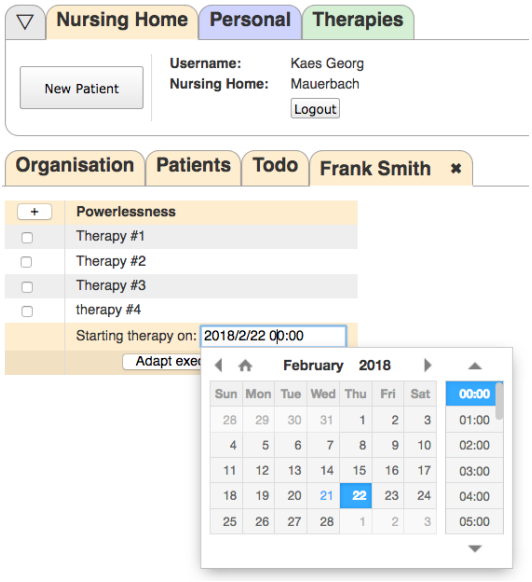


Figure 6.9: Selection of the Therapy in ACaPlan

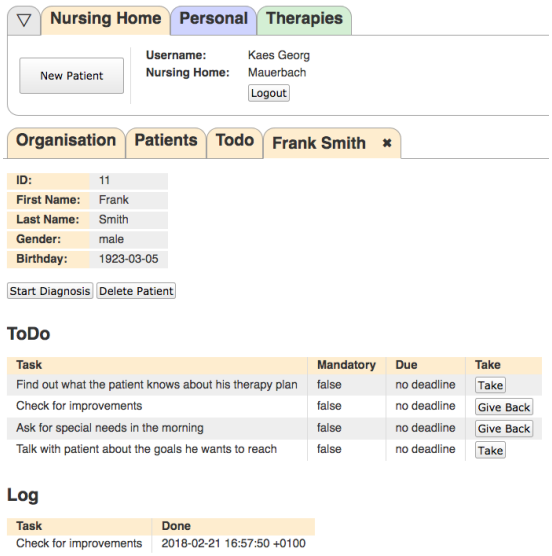


Figure 6.10: A Patient Profile in ACaPlan

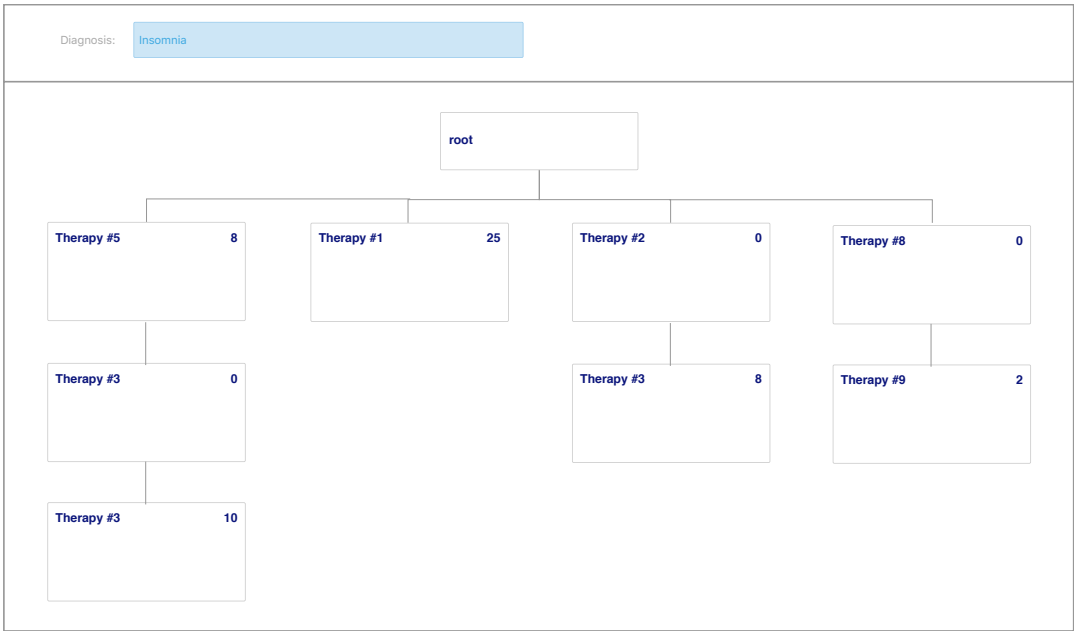


Figure 6.11: Application of the Change Tree in ACaPlan

support the nurse in analyzing which change operation may work, and which may not reach the desired goal.

Using the change tree, nurses could see for a certain process change log projected to the change operations for a certain diagnosis, which change operations have been conducted more or less frequently in such a scenario, as depicted in Figure 6.11. Especially when filtering only for relevant parts of a change log, as discussed in Section 3.4.1, or in combination with techniques such as multidimensional process mining [58], such a view on a change log can provide additional information to a nurse. For example, using filtering and projection techniques, the nurse could set the focus of the change log to those change traces, which belong to patients with a certain age, gender, and type of problem, e.g., diabetes. Thus, she can see immediately which change operations have been applied for such patients.

The Nursing Outcomes Classification defines a set of possible outcomes for each diagnosis, which are the effects of conducted change operations which can be seen in the corresponding process execution log. For example, for the diagnosis of *insomnia*, the outcomes *"The patient reports better sleep patterns"* and *"The patient reports to feel more rested"* are defined. These statements of the patient can be easily checked with a respective process activity, where the nurse asks the patient respective questions. Hence, the evaluation of those outcomes (whether they have been reached or not) can be found in the process execution log. In

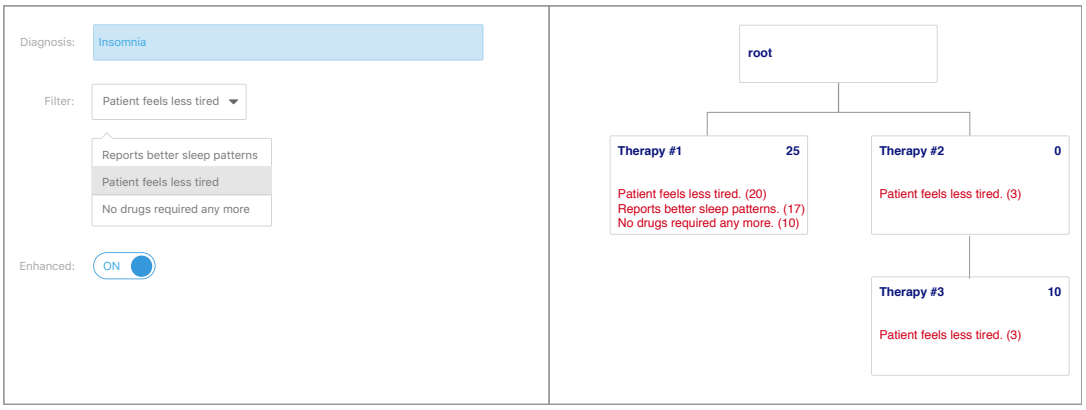


Figure 6.12: Application of Process Change Effects in ACaPlan

combination with the filtering and projection techniques developed in Section 3.4.3, an Enhanced Process Change Tree (EPCT) can be created, where the nurses can see which outcomes defined in NOC have been reached using certain adaptations. When a nurse now selects a certain outcome which is especially important for her patient, she can immediately see which change operations can reach this outcome with which probability. An exemplary visualization is shown in Figure 6.12: Here, the second outcome, *"The patient feels less tired"* is chosen. By projecting change traces to the corresponding case, and filtering based upon the desired effect, the nurse immediately sees which change operations can reach the desired outcome.

When multiple change operations are possible, e.g., because multiple change operations have a similar chance of reaching a desired outcome when consulting historic data, other dimensions may become of interest, such as the resource or time perspective. For such an analysis, the similarity concepts developed in Chapter 2 can be of interest. Here, we have created three process change operation similarity metrics, which can be used to compare process change operations based on the attribute, resource and time perspective. Figure 6.13 shows an exemplary interface which implements this functionality: After choosing a change operation and a change operation similarity measure, all other change operations present in the (filtered) change log can be compared and listed in descending order regarding their similarity. Thus, a nurse can compare multiple single change operations, and decide for a change operation based on required resources or time.

As discussed in Chapter 4, change operation similarity metrics cannot only be used to compare single process change operations, but also to aggregate change traces in a change tree. This is visualized in Figure 6.14. In this example, the nurse is interested in inspecting which change operations require similar resources, up to a similarity threshold of 80 percent. In combination with process change

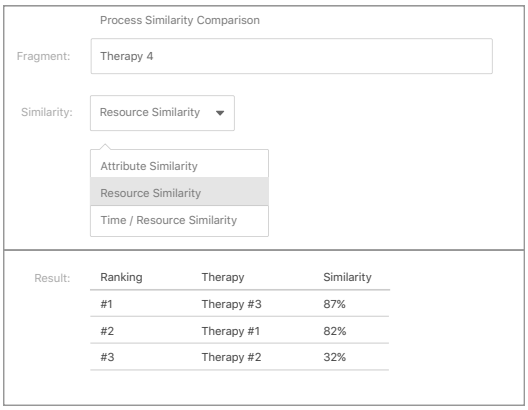


Figure 6.13: Application of Process Change Similarities in ACaPlan

trace filtering and projection techniques, such a visualization helps the nurse to see which resources may be required to reach a certain outcome. For example, if a certain resource (e.g., a special kind of machine necessary for a certain therapy activity) is not available for several weeks due to necessary maintenance work on the machine, the nurse can see the success chance of therapies which do not utilize this resource, and compare them to the success rate of therapies which utilize it.

Summarizing, ACaPlan is a process based tool which aims at supporting nurses while they take care of their patients. By applying the concepts discussed in this thesis, it can be extended and support for the nurses can be increased.

6.3.2 Application to the Wellbeing Domain

Another application lies in the application to the wellbeing domain, specifically to the treatment of chronic pain.

Chronic pain is a common problem for many people. It has been proven that chronic pain is learned by the brain [71, 72], which leads to the conclusion that pain can be "unlearned" by the brain by applying certain ways of treatment [73].

As several studies have shown, patients who experience chronic pain can be supported by regularly executing certain exercises, such as breathing or meditation techniques [74, 75, 76, 77]. These exercises can easily be conducted at home, thus making it easier for patients to conduct them on a regular basis.

However, patients need to be supported in conducting these therapy activities on a regular basis. Usually, this is done in a group therapy conducted once a week over the course of five weeks in total: The patients get educated about the positive effect such activities can have on their chronic pain, do some of these activities as a group, and get instructed to repeat them at home until the next session on a regular basis. However, many patients tend to only execute the

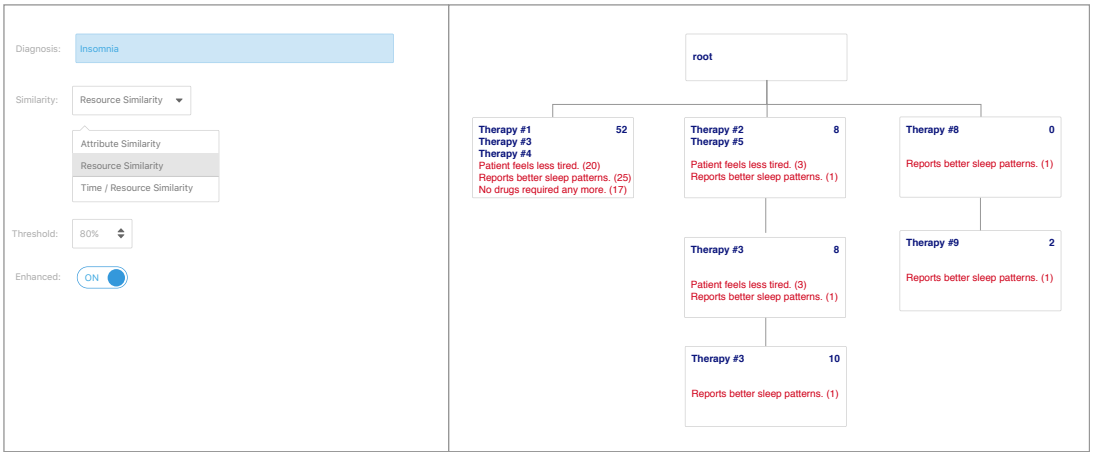


Figure 6.14: Application of Aggregating Change Trees based on Process Change Similarities and Outcomes in ACaPlan

recommended activities when they participate in the group therapy sessions. As soon as they leave group therapy however, they do not execute them any more, thus diminishing the positive effect those activities can have to their chronic pain. This challenge could be tackled by implementing a mobile application for smartphones and tablets, which reminds them on a regular basis (e.g., once or twice a day, depending on the chosen interval) that the exercises should be done.

In this section we present how flexible processes and change log mining can be used as a basis of such an app, thus showing the applicability of the approaches developed in this thesis to the medical domain. The concepts the app is based on have been developed in corporation with a medical specialist for neurology and doctor from the *SMZ Baumgartner Höhe Otto-Wagner-Spital*, a hospital located in Vienna, Austria.

Process-based Representation and Features of the Implementation

As for *ACaPlan*, for each patient there exists one individual process instance containing all tasks which have to be executed by the patient. While the basic concept remains similar, in contrast to *ACaPlan*, where the nurses carry out the therapy steps and create the documentation, here, the patient completes these activities all by himself. Since it cannot be expected that patients know the exact consequences of each therapy activities they can conduct, this induces a new level of necessary user support: First, the generation of the therapy process fragments has (a) to be more automatized, or (b) still supported from the responsible doctors. Additionally, instructions which therapy activity has to be conducted in which way have to be provided. Thus, for each activity a detailed explanation has to be

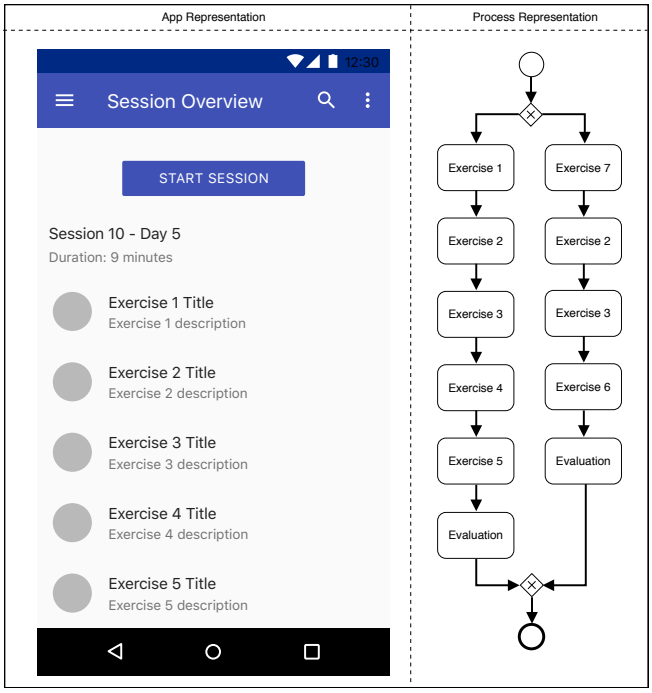


Figure 6.15: The app has to hide the complexity of the underlying process technology

available at all times, since it cannot be expected that the users of the app know how to conduct the therapy activities. Aside from this, the process concept of the pain therapy app is similar to the one from ACaPlan depicted in Figure 6.5: For each patient, an individual therapy process instance exists, which gets adapted depending on the requirements of this individual patient.

Based on this concept, we will now discuss which features such a process based app should have in order to support patients who want to treat their chronic pain using an app-based therapy:

An interface for such a therapy plan needs to be simple, understandable, and should not require any technical knowledge of the patient. Thus, the complexity of the underlying process technology should be hidden from the patient, as depicted in Figure 6.15. Here, all the patient sees is a list of activities he has to execute. When he swipes left or right however, he can switch between different sets of exercises. In the underlying process model, these two choices are depicted as two exclusive branches, which will be activated based on the patient's decision. Thus, the patient can guide the control flow of his process instance by well known gestures and user interface interactions, such as swiping, tapping or entering values, while the complexity of the process instance itself is hidden.

The user interface also has to handle different types of process activities, e.g., activities which should trigger exercises the patient executes, and activities which allow the patient to enter data, e.g., to evaluate how he feels after he has completed the exercises. Thus, depending on the process activity being active, a different interface has to be shown. This is visualized in Figures 6.16 and 6.17: Figure 6.16 depicts the interface being shown when an exercise activity is active: A video of what the patient should be doing illustrates the activity. Additionally, a textual description and a title help the patient to understand what has to be done. When the *Evaluation* activity (cf. Figure 6.15) is activated, a different interface (depicted in Figure 6.17) is shown: Now, the patient has to evaluate his level of pain after the training has been finished.

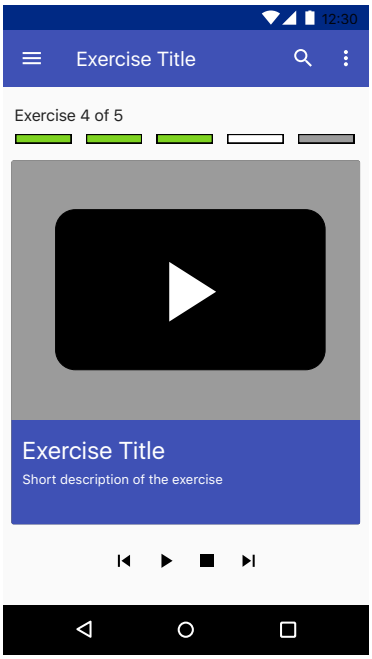


Figure 6.16: Exercises in a therapy session

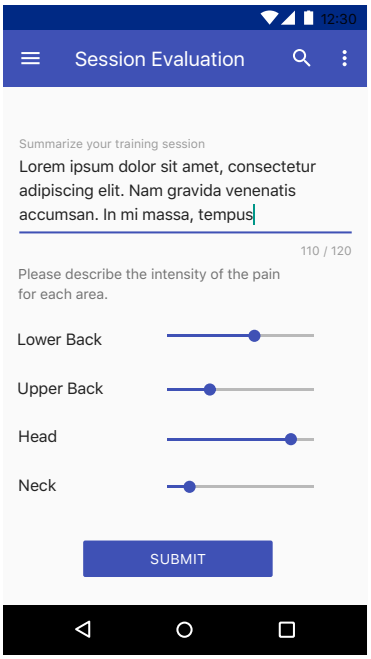


Figure 6.17: Evaluating the results of the therapies

Application of Change Mining

Change mining techniques as presented in this thesis can be used for multiple purposes, e.g., to analyze which steps executed by the patient may work, and which will not reach a desired effect, or to compare several possible sets of exercises:

Each time the patient has to conduct a new set of exercises, this set is added to the patient's process instance. When the patient executes these activities (marking

an activity as *completed* when he has done an exercise, or entering evaluation information in an evaluation activity), these executed steps are then added to his execution log. This information can be used in multiple ways:

Using the projection and filtering techniques discussed in Chapter 3.4.3, it can be analyzed which therapies have led to a certain effect for how many patients. Each time the patient enters how strong the pain is he experiences, he creates a new event in the execution log. If it is defined that the pain the patient experiences should be below a certain threshold (this information can directly be deduced from the execution log), the *enhanced process change tree* can now be used as a basis to see which therapies have led to which effects. If one wants to focus this analysis on a certain type of patient (e.g., one wants to find out which therapies work for male patients elder than 70 years, which have chronic back pain), the filtering technique presented in Chapter 3 can be used. The resulting change log can be fed into the EPCT, and the resulting change tree shows the probability of reaching a certain effect for one specific type of patient only. Such an analysis can be useful both for patients who want to see which therapies might be successful, as well as for doctors who want to analyze and improve the effectiveness of existing therapies.

Change operation similarity metrics can be integrated into the analysis of such a process based application as well: If a patient wants to compare possible change operations (i.e., therapies), e.g., regarding the time it will take him, he can do this by using change similarity metrics (cf. Chapter 2). Using resource- or time based metrics, he can, for example, see all exercises where he needs no equipment (for example, if he is on holidays), or where he needs only a certain amount of time.

The discussion presented in this section suggests that representations and analysis techniques discussed in this thesis can also be applied to the wellbeing domain, thus supporting individual patients who want to find a way to improve their life (e.g., by experiencing less pain). All data which is required by the algorithms and representation methods discussed in this thesis are present: In process change logs, we can see how the training plans of different patients have evolved. Using data from the corresponding execution logs, we can see how regularly these activities have been executed, and if the overall pain has improved. Thus, information from the processes change and execution log, in combination with information about the patient, and the techniques presented should form a solid basis for showing the users a good set of change operations for his current situation. Next, we analyze this assumption in an interview with a domain expert.

Evaluation with a Domain Expert

As stated above, the general concept for the app which supports pain therapy patients has been developed in cooperation with a neurology specialist from the Otto-Wagner hospital in Vienna. In order to analyze whether the concepts de-

veloped in this thesis can be applied to such a domain as well, we conducted a qualitative interview with a neurology specialist who interacts with pain therapy patients on a regular basis. Before the interview, we presented the applications discussed in Section 6.3.2 above to the specialist. Starting from these suggestions for user support, the interview was based upon the following set of questions:

- Is the application of change mining techniques useful to the domain? If yes, why is this the case? If no, why not?
- Can doctors and nurses who work with pain patients on a regular basis use such an app as a support tool?
- What would be the benefits for doctors and nurses?
- Can change mining techniques support the patients when doing their exercises in your opinion?
  - Would the patients be motivated by such a design?
  - Is the interface usable and motivating? Are all necessary features available, or is something missing?

Each question led to a short discussion with the specialist, in which we examined the question from various vantage points. A summary of the answers and discussions based on those questions are listed in the remainder of this section:

Is the application of change mining techniques useful to the domain? If yes, why is this the case? If no, why not? During the interview, it became clear that the change mining techniques as presented in this thesis can in fact be applied to a setting as described above. The most important aspect of such an app would be to motivate patients to do the recommended exercises on a regular basis. Such a behavior can be encouraged by providing good user support: Using change similarity measures, patients can find alternative process adaptations or therapy fragments which fit their current requirements. This is especially useful if patients are not fit, or have a negative history with a certain type of exercise: If the app can show them alternatives, it can increase the motivation to actually participate and do the exercises on a regular basis. Also, some exercises require additional resources to be present, such as a special type of elastic band. Such additional requirements can be seen as resources which are required by the process fragment; following, the *change resource similarity* (CRS) as presented in Section 2.5.1 can be used as a basis to compare process adaptations: By comparing all change operations in the repository with an adaptation which requires this elastic band, one can see which change operations require it as well.

Can doctors and nurses who work with pain patients on a regular basis use such an app as a support tool? An app which motivates the patient to conduct the necessary therapy activities at home on a regular basis would definitely be a good support tool. As stated by the neurology specialist, the most important thing is that the activities are carried out by the patients on a regular basis, in order to create a habit, and *"unlearn"* the pain. Thus, the app also facilitates work for the doctors and nurses, since patients are intrinsically motivated to do the activities, if the app is designed in a motivating way.

What would be the benefits for doctors and nurses? Aside from motivating patients to execute the recommended therapy activities, change mining techniques as presented in this thesis can be of use for doctors and nurses as well: Change trees, including n-gram change trees, enhanced change trees (with effects), and aggregated change trees provide a good evaluation tool, especially for doctors and nurses who want to take a closer look at what has helped a certain set of patients: By means of filtering and case projection techniques (cf., Chapter 3), as well as including process change effected, one can take a close look at historic data and evaluate which change operations have caused some desired effects. Thus, as stated by the neurology specialist, the change tree can be used as a support tool for doctors and nurses who work with patients, since they can use it to inspect historic process data from different perspectives.

Can change mining techniques support the patients when doing their exercises in your opinion? The most important thing for patients is that they are motivated to execute the recommended activities. This works best when the patient feels comfortable with the exercises provided for him. Thus, change mining techniques, which recommend activities based on historic data, are well suited to suggest activities which fit this requirement: If a certain set of activities has been evaluated as being a positive influence on the patient's pain, the patient may be more motivated to do them again.

Would patients be motivated by such a design? Motivation of the patients highly depends on their knowledge and experience with smartphone apps: While good and useful suggestions are definitely an important part, additional techniques such as gamification elements may have to be incorporated as well.

Is the interface usable and motivating? Are all necessary features available, or is something missing? The interface covers all important aspects which are relevant for users who have to interact with the app while executing and adapting their therapy plans. However, the first anamnesis, where the type of the

patient is being determined, may have to be incorporated at a later stage. (For now, it is planned that this anamnesis will be conducted by the doctors before the patient interacts with the smartphone app.) In this anamnesis, the user could interact with the smartphone app, and select the type of exercise he likes most from a set of available pictures.

Conclusion: Summarizing, the interview with the domain expert from the Otto-Wagner hospital in Vienna suggests that change mining can indeed provide useful results for patients as well as for doctors and nurses: Using similarity measures, patients can see which change operations are similar to those they are already used to. Change trees (with all features presented in this thesis) provide a powerful tool for doctors and nurses to analyze the effectiveness of therapies for different groups of patients.

6.4 Discussion

In this chapter, we have presented five different domains where process change operations, respectively the approaches developed in this thesis have been applied, or can be of interest. As shown for the financial, medical, and higher education domain, process change trees can not only be applied to change logs, but also to process execution logs in order to gain additional information. Additionally, we have presented concepts for integrating the representation and analysis methods presented in this thesis in tools and user interfaces for the nursing (Section 6.3.1) and wellbeing (Section 6.3.2) domains. These user interfaces can be used to provide support for users who want to analyze historic data. Additionally, the approach for the wellbeing domain has been evaluated with a domain expert, thus showing the potential these concepts have in a real world setting.

Overall, the concepts developed in this thesis provide a solid basis for analyzing historic information on process change operations and corresponding execution data. By means of process change operation similarity, for example, users can analyze which change operations behave similarly in a certain context. Especially in this section, future work may be of interest: For the analysis of success rates in the nursing domain (based on the *Nursing Outcomes Classification*), similarity metrics which also include execution log information may provide additional information on process change logs: Using such a metric, users who analyze changes to a patient's therapy process can see which change operations have reached the desired effect.

In the next chapter, we will discuss a special set of process settings, where change operations are an integral part of the process design, namely *Flexible and Individual Process Settings* (FIPS). Process instances in these process settings

incorporate *flexibility by change* [78] into their basic process design, and evolve according to the requirements of one subject: Starting from a process stub, during the execution of the process instance, the instance's process model evolves according to the individual requirements of one subject. One example for such a FIPS stems from the nursing domain: Here, for each individual patient (subject) a process instance exists, which contains all therapy activities which are required for this one patient. Since these requirements can change on a regular basis, the process instance has to be adapted continuously. When a new patient is admitted into the nursing home, a new process instance is created based upon a simple process stub, which may only contain the necessary tasks for the first days, and has to be adapted to fit the needs for this one patient. Following, compared to more standardized process applications (e.g., deciding on a loan in a banking environment), FIPS create vast amounts of process change and execution data, thus providing an interesting testing environment for the approaches developed in this thesis.

Chapter 7

Flexible and Individual Process Settings

In this chapter, we discuss *flexible and individual process settings* (FIPS) as a special type of application domains. Overall, FIPS pose the *"killer application"* on the presented concepts for representing and analyzing change logs, hence they are discussed in a dedicated chapter. Figure 7.1 positions this chapter in the structure of the thesis: After describing the representations and analysis methods for process change operations and change traces, we presented the technical evaluation based upon a prototypical implementation, and we showed possible application domains. In this chapter, we analyze commonalities between some of the possible application domains: we introduce flexible and individual process settings (FIPS) as a category of process settings, where *flexibility by change* [78] is part of the underlying process design. Compared to usual process settings, where adaptations cover and handle exceptional situations, in FIPS they are used for designing the process model flexibly during runtime, thus creating a vast amount of process change data. Subsequently, due to the high amount of available change data, FIPS provide interesting and challenging settings for implementing and testing the approaches developed in this thesis, and pose the *"killer application"* on the presented concepts. Based on a description of such process settings, we present a game-based experimentation prototype which simulates flexible and individual process settings. This chapter is based upon the papers *"Flexibility Requirements in Real-World Process Scenarios and Prototypical Realization in the Care Domain"* [79] and *"Generating Data from Highly Flexible and Individual Process Settings through a Game-based Experimentation Service"* [80].

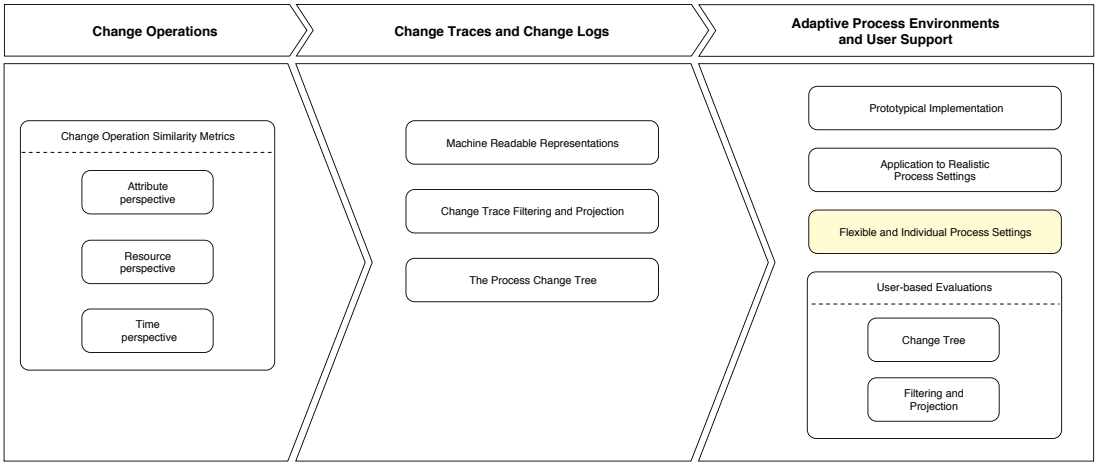


Figure 7.1: Dissertation Overview: Flexible and Individual Process Settings

7.1 Overview

Depending on the process setting there are different degrees of flexibility [78], hence, process flexibility has a different importance for the overall setting. In some settings, multiple process instances follow the same basic schema which has to be adapted if some exceptional situation occurs. In other cases, each process instance is different by design (cf. Chapter 1). Take the therapy process of a patient in a nursing home as an example (cf. [79]). As discussed in Chapter 6, for each patient there exists one long-running process instance executing the therapy steps specific to this patient. There is no common schema, but each instance develops based on ad hoc adaptations. Whenever a patient shows new symptoms, the nurse has to adapt the therapy process accordingly. Such settings are referred to by highly *Flexible and Individual Process Settings (FIPS)*.

Overall, FIPS are the *"killer applications"* regarding process flexibility. As described in [81], a killer application *"combines, in a nutshell, all the problems and challenges usually only found in different application areas"* [81]. FIPS provide such a setting for process flexibility, since here, process flexibility is the main driver of the process design. Since change operations for process instances occur frequently in a FIPS, supporting users in analyzing existing adaptations becomes crucial. This frequent use of change operations also leads to change logs in a FIPS tending to contain a lot of data. While we have shown in Chapter 6 that the developed concepts can be applied to process settings, where flexibility can be, but is not necessarily the main driver of process design, in this chapter we want to specifically focus on FIPS, since they, being the killer application for process flexibility, can serve as an excellent evaluation setting for process change

representation and analysis techniques.

In FIPS, users responsible for maintaining a process instance are regularly supposed to decide on, specify, and conduct adaptations [8]. Hence, such settings provide a solid basis for testing and evaluating representation and analysis techniques for process change operations, as they have been presented in this thesis. FIPS can be found in many domains [79]. Examples range from regular customer care in a hotel setting, over the medical domain, to schools for children with special needs. In the nursing domain, for example, nurses could be supported in finding the best adaptation for the current patient's therapy process based on his current symptoms and medical history [79]. The goal of this chapter is to provide a game-based experimentation service for FIPS which would enable the simulation of FIPS independent of any application domain. Such an experimentation service would provide a controlled testing and evaluation environment for approaches which aim at representing and analyzing process change operations with a holistic view on data. Additionally, it would enable the creation of process execution and change logs in case no real-world logs are available due to, for example, legal or privacy reasons. Such execution and change logs can then be used as a first test set for new representation and analysis approaches in this area.

To create such a controlled testing environment for FIPS, we first need to analyze the basic building blocks of FIPS. For this, we analyze real world examples. Based on such an analysis, it has to be analyzed whether and how these building blocks can be mapped and integrated into an experimentation service. In the next chapter, we will then use this experimentation service to conduct a user-based experiment, where we apply some of the concepts which have been developed in this thesis to show that they can also be used in a FIPS.

Hence, in contrast to Chapter 6, where we discussed how the concepts developed in this thesis can be directly applied to different domains, in this chapter, we take a step back and analyze the commonalities between those domains where flexibility is the main driver of the process design. This analysis then serves as a basis to create a controlled testing environment, which again serves as a basis for one of our user-based evaluation presented in Chapter 8. Overall, this chapter tackles the following research questions:

Q1: Which building blocks are common to FIPS?

Q2: How to reflect FIPS requirements in a game-based experimentation service?

Q3: Is a game-based design suitable for generating the data common to a FIPS?

The chapter is structured as follows: In Section 7.2, building blocks and concepts of a FIPS are analyzed and extended based on expert interviews and explained including data sources and components which have to be represented in

the experimentation service (→ Q1). Section 7.3 describes the design of the experimentation service: Here, we present the mapping of FIPS requirements, data sources, and components to the concepts of the game (→ Q2). Further on, the gameplay and user interfaces (Section 7.4, the service-based architecture of the game and details of the generated log files are presented (Section 7.5). The feasibility of the data generated by the game is evaluated by comparing the generated data elements with data generated by a real world FIPS (→ Q2, Q3, Section 7.6). Finally, Section 7.7 concludes the chapter.

7.2 Generalization of Highly Flexible and Individual Process Settings

FIPS can be found in many domains. In order to create a game-based experimentation setting which has the same properties as a realistic FIPS, as a first step, the basic building blocks of a FIPS need to be identified. For this, we pursued a two-phase approach: In the first phase, we analyzed two FIPS based on data available in different research projects, thus generating a preliminary set of building blocks. In the second phase, these building blocks have been evaluated with expert interviews from different domains which also incorporate FIPS. The methodology is described in detail in Section 7.2.1. This is followed by a list of the building blocks which have been identified following this two-phase approach (Section 7.2.2). Next, we present the detailed results of the expert interviews: In Section 7.2.3, we show how the building blocks are implemented in the FIPS domains of the respective domain experts.

7.2.1 Methodology

We have identified the basic building blocks for FIPS based on multiple examples stemming from different domains. In the first phase, we have analyzed two different domains we have identified as FIPS, i.e., the nursing, and the manufacturing domain. For the nursing domain, we have analyzed the Adaptive Care Planning (ACaPlan) project, which has been introduced in Chapter 6. For the manufacturing domain, we have analyzed information from the FP7 Adventure project[1]. When manufacturing individual products, each production process has to follow its own, individual process instance, since the constructed products are different. *Azevedos*, a partner of the FP7 Adventure project, is one of the leading manufacturers of machines for cork transformation. They plan and produce machines which are tailored according to the demands of one specific customer. Depending

[1] urlhttp://www.fp7-adventure.eu

on shifting requirements which are specified during the planning phase, and potentially occurring changes in the suppliers and availability of required components, flexible and individual process instance emerge for the creation of each product.

Based on this first analysis, a preliminary list of building blocks which are present in a FIPS has been developed. In the second phase, this list has been verified and extended based on expert interviews. In order to broaden our perspective, for the second phase of the elicitation of the building blocks, we chose different domains which we identified as FIPS. The expert interviews have been conducted with domain experts from the following domains:

- **Software Sales and Support:** When dealing with the wishes and problems of individual customers, flexible and individual process instances come into play: *mesonic*, a company producing enterprise resource planning software for small and medium enterprises, can manage the individual support of their customers based on flexible process instances: Depending on the needs of one specific customer, different solutions can be presented to him.

- **Hotel Management and Guest Care:** Just as individual customers have individual requirements, guests at a hotel may also require individual treatments: Depending on the wishes and needs of regular guests, hotels can decide which treatments are most relevant to these individual processes.

- **Special Needs School:** Children with special needs also have individual requirements regarding their teaching plan: Whenever some new problem occurs, the teacher has to flexibly adapt the individual teaching plan of the affected child.

- **Doctoral Program Supervision:** Each project from a PhD student can also be seen as a subject affected by a flexible and individual process setting: While the general plan for the study is laid out in the first year, it has to be adapted continuously according to changing requirements.

All of the settings mentioned above have one thing in common: They contain at least one kind of individual process instance, which has to be adapted continuously according to the needs of a certain subject. In the next section, we will discuss which building blocks can be found in such a setting.

7.2.2 FIPS Building Blocks

In this section we present the final list of building blocks for flexible and individual process settings ($\rightarrow$ Q1), which has been created based on the methodology

described in Section 7.2.1. These building blocks are domain independent abstractions from general concepts such as data sources or procedures which exist in specific domains which can be seen as FIPS. We found implementations of these concepts in any domain we investigated which we considered as a FIPS. The rest of the section defines these building blocks and gives examples from the nursing domain. Following, the domains which have been evaluated with expert interviews will be described in detail and domain specific implementations of these concepts, which have been analyzed based on expert interviews, will be presented. For the sake of completeness, those building blocks which have been identified in the first phase of our analysis (cf. Section 7.2.1) are highlighted with an asterisk (*) behind their name.

- **Subjects (*)** define the individual persons, objects or concepts that are subject to the process execution [79]. In the nursing domain, these individuals are the patients for which the process instance exists and is being adapted.
- **Process Instances (*)** [3] capture and execute the process logic for a subject. In FIPS, instances typically run for a long time and are affected by process change operations whenever necessary. In the nursing domain, the process instances contain the therapy tasks which have to be executed in order to make the patient feel good or better.
- **Organizational Units (*)** reflect the organizational perspective of a process. They can be used to define relationships between actors, define skills and possible actions for those units etc. [3]. In the nursing domain, the organizational units are the nurses, doctors, assistant nurses, and all other employees of the nursing home which are in some way related to a patient's therapy plan.
- The **Environment (*)** [79] of a FIPS comprises all data that is related to the subject, but not directly incorporated by the subject, for example, the limited resources which are required to execute the tasks related to the subject. In the nursing domain, the environment would be the nursing home.
- **Process Fragments (*)** are parts of a process which get inserted into, deleted from, or in any other way affected during a process change operation [79, 3]. In the nursing domain, these process fragments refer to therapies which are added to or removed from a patient's therapy plan.
- **Problem List:** For each fragment a set of conditions where the fragment should not be used can be defined. In [8], such a problem list item has been introduced in an example from a medical domain, where a certain adaptation could not be made because the patient has a cardiac pacemaker.

- **Bonus List:** In contrast to the problem list, the bonus list for each fragment describes situations where the fragment is known to perform well. This building block has been identified during the expert interviews. In the nursing domain, this bonus list contains details about the patient which give a hint that a certain therapy may perform well.

- **Triggers (*):** In [3], the necessity of ad hoc modifications to a process instance is defined by exceptions which have not been thought of when designing the process model. Whenever such an exception occurs, the process instance has to be changed in order to deal with this exception. In FIPS, such events occur on a regular basis, thus we argue that the term *exception* does not fit in this context. Since these situations trigger a change operation, we will use the term *trigger* instead. In the nursing domain, symptoms are the triggers. Whenever a patient shows new symptoms, something has to be changed in his therapy process.

- **Positive Goals [82]:** There are always reasons which justify a process change. These reasons can be defined as *goals* which should be reached by executing the tasks which have been inserted, or not executing the tasks which have been removed from the process instance. In the nursing domain, positive goals can be to make the patient feel better, more confident, or at least to stabilize him in his current condition.

- **Negative Goals [82]:** Process changes can also be conducted in order to avoid certain situations and hence they address negative goals. In the nursing domain, examples include the deterioration of the patient's condition or even the death of the patient.

7.2.3 Expert Interviews

Following the elicitation of the basic building blocks for FIPS using data from two scientific projects, as described in Section 7.2.1, we conducted several interviews with domain experts from multiple domains in order to verify and extend these building blocks. Highly individual settings can be considered as FIPS if they require some kind of process to be adapted regularly, thus being the main driver of the process design. For evaluating the building blocks and general properties of a FIPS, we chose *software development*, *hotel management and guest care*, *special needs school* and the *PhD program* as domains to be evaluated with domain experts. We chose different domains for those expert interviews than for the first elicitation in order to broaden our view on FIPS.

The interviews started with the experts describing the domain they are working in. We asked them for examples where they had to deal with special individual

situations, and what they could personally learn from these situations. In the following, we presented them the basic idea of FIPS based on the nursing scenario and discussed possible commonalities between their domain and the nursing domain as a FIPS. We chose the nursing domain as a reference, since it is easy to empathize with, even if the interviewed person is no expert in this domain. Based on this information, we identified in cooperation with the domain experts the basic building blocks which have been presented in the last section.

Expert Interview: Software Sales and Support

We interviewed two sales managers from mesonic, an Austrian company which develops enterprise resource planning (ERP) and customer relationship management (CRM) software for small and medium enterprises. The contact to the customers works to a big part over a network of retailers which are in geographical vicinity of their customers. The support of mesonic mainly works over an internal system where problems with the software itself (e.g., bug reports), individual wishes, and additional requirements from certain customers and retailers are gathered. Depending on the information the company's support department has about the case at hand, different measures are taken: If a bug is identified (which is already known or reported by many other customers), it will be discussed in the next developer meeting, where further steps will be defined. Depending on its priority, it may be solved right away, or as soon as resources are free. A customer's wish for an additional feature may be implemented in a future version of the product, depending on the workload of the developers, the usefulness to other customers, and several other parameters.

- **Subjects:** The individual subjects are the support cases created by the customers and retailers. Although each support case is different, some parallels such as the concerned build number, the window and the type of the support case can be found.
- **Process Instances:** For each individual support case there exists a process instance which aims at solving the problem at hand.
- **Organizational Units:** software developers, support staff, sales team, ...
- **Environment:** The environment is the software company with all its resources.
- **Process Fragments:** Process fragments are the tasks to be executed, e.g., implementing a new feature, fixing a bug, updating the documentation, informing the customer etc.

- **Problem List:** In some cases, the commonly preferable solution to a certain kind of support case may not be possible, e.g., because all resources are already otherwise occupied.
- **Bonus List:** Sometimes a certain solution to a support case can be preferred, e.g., because a requested feature is of use to many customers.
- **Triggers:** During the lifecycle of a support case, different triggers can appear: For example if it becomes evident that a certain problem is indeed a bug and not a problem on the customer's side, the support case has to be handled differently.
- **positive Goals:** solve the problem
- **negative Goals:** procrastinate solution, make requesting customer angry, loose this customer
- **Individualism:** Each support case is unique regarding its parameters: Although some may tackle the same problem or wish, the handling might still be different.
- **Limitation of available resources:** The developer, support and sales team only consist of a limited number of people who cannot handle everything instantaneously
- **Ambiguity of triggers:** The same problem description of two or more support cases does not mean that it is the same problem (in one case, there may be an actual bug; in the other, the customer did not read the manual).
- **Diversity of possible solutions:** There is never only one way to deal with a certain situation; for example, depending on the workload, a new feature may be handled either instantaneously and directly in the company, later, or not at all.

Discussion: After introducing the nursing domain as a point of reference, we talked about commonalities between the software sales and support and the nursing domain on an abstract level. It became clear that commonalities between the individual problems of a patient in the nursing home and the problem description in a support case exist indeed: In both scenarios, there is a problem which has to be solved. In order to identify the best course of action (i.e., the best process fragment), parameters from the case itself as well as from the environment are analyzed. While in the nursing home the circumstances of the patient's problem are analyzed, including his medical history and common diseases, in the software

domain the circumstances of the support case are identified, including the targeted version of the software and other, similar reported problems. When the problem itself is identified and the goals are clear, the next steps are planned. In the software domain these steps include work to be done by software developers, the support and sales departments. After these steps have been conducted the person in charge of the case evaluates whether the goals have been reached or not.

Expert Interview: Hotel Management and Guest Care

Hotel management also deals with individual subjects (i.e., guests) having specific requirements. We interviewed two receptionists from a local hotel and seminar location which focuses on business as well as leisure guests. The guest profiles range from typical seminar groups with a trainer and multiple employees or managers, to couples or families who want to spend some days in a hotel. Obviously, these groups have different requirements for the location: While for the leisure guests, for example, outdoor activities or wellness offers have a higher priority, for the business guest a well-equipped seminar location or a smooth procedure of their workshop is more important. Hence, the hotel has to provide different offers depending on the guests' profiles and demands.

- **Subjects:** The individual subjects are the guests who stay in the hotel, i.e., leisure guests, business guests, seminar groups and trainers
- **Process Instances:** Each individual has to be treated according to his preferences. The service management plan contains all tasks which have to be done to keep seminar groups and other customers happy.
- **Organizational Units:** In a hotel, multiple roles are involved in the process of keeping the customer happy, among others the hotel manager, front desk officers, back office, housekeeping, service, cooks, etc.
- **Environment:** The environment is the hotel with all its individual features and resources.
- **Process Fragments:** Fragments vary depending on whether they deal with a business or a leisure guest; examples include to help seminar group with the equipment, prepare a special dinner, etc.
- **Problem List:** Depending on the situation, there are multiple reasons for a certain fragment not to be executed. In case of the special dinner it may be allergic guests who may not receive all kinds of food

- **Bonus List:** As with the problem list, also here we can find multiple examples. For example seminar guests which are known to have problems with the equipment will preferably be supported more.

- **Triggers:** Guest wishes and complaints have been identified as the main events which trigger some change in the service management plan.

- **positive Goals:** keep customer as a regular, surprise him, do not disappoint him (expectations were too high)
  make customer's seminar a success, solve customer's problems

- **negative Goals:** guest may not come again, negative impact on other guests, negative assessment in online portals about the hotel, ...

We have identified the following dynamic properties of the hotel management and guest care scenario:

- **Individualism:** Depending on the type of guest different parameters make them individual. For leisure guests, the number of children and dogs, age, holiday category (relaxing, sports, culture, etc.) etc. are relevant. For business guests and trainer the size of seminar group, equipment requirements, number of required rooms etc. are crucial.

- **Limitation of available resources:** In the hotel, resouces are also limited, i.e., there is only a finite number of rooms, beds, equipment, restaurant tables etc.

- **Ambiguity of triggers:** A guest's problem description does not have to mean that it is always the same problem. ("Our beamer does not work" may require a different reaction from the hotel staff, depending on the actual condition of the beamer.)

- **Diversity of possible solutions:** Depending on environmental parameters of the hotel (such as the current occupancy rate of the rooms), there may be different solutions for the same problem. For example if a trainer asks for a seminar room for two days with 10 hotel rooms for the night, but there are only 5 hotel rooms available, different solutions are possible (moving them to a partner hotel, give them another date where enough rooms are free, cancel the event, etc.) If the hotel staff already knows they are dealing with a troublesome guest who complains a lot each time he is at the hotel, their reactions to his complaints may be different than if the guest is known to be friendly.

Discussion: Each interaction with a guest can be seen as part of a FIPS. Depending on the guest profile and the current offers of the hotel and seminar location, different things can be offered to the guests. For example, if a guest is known to be quite exhausting, he will be treated with special care in order to keep him at bay. A seminar group who is known to have problems with technical equipment will receive special support right from the start. This behavior again shows commonalities to the nursing or the software development domain: First, the situation and the problem are identified, then, based on knowledge about the subject at hand and what has worked in a similar situation before, the best course of action is identified.

Expert Interview: Special Needs School

We interviewed two teachers from a special needs school who teach children with special needs from ages 10 to 15. The special needs range from language issues, over physical up to psychical conditions of various kinds. Depending on the specific needs of the children different teaching concepts have to be chosen. While there exists a general plan for the school year as a whole, it has to be presented to each child in a different way. Of course, due to resource limitations, this is not always possible. Hence, the teachers are required to balance the needs of all children.

Whenever a child shows some ad hoc needs, for example his or her concentration is going down, the teachers have to find the root causes for this symptom, and find a way to deal with it. Depending on the child and his or her needs, different causes can lead to the same symptoms. For example, learning problems may indicate family or physical issues. It is the teacher's job to find countermeasures which will help the child to feel better and more confident again. These countermeasures differ from child to child, and a great deal of experience is required to find the best one.

- **Subjects:** School children with special needs are the individual subjects which have to be treated according to their personal needs.

- **Process Instances:** The teaching plan which is used to teach children can be seen the general process instance. Depending on the requirements of each individual, this teaching plan can be different (within certain legal boundaries). Also, all other kind of necessary interaction between the teacher and the child's social environment can be seen as part of this process instance, since it also has to be individual.

- **Organizational Units:** Organizational units include, besides the teacher, psychologists, pedagogical consulting (contact to the parents), language ped-

agogic teachers (for children with speaking problem, foreign mother language etc), native speakers etc.

- **Environment:** The school with its characteristics and resources is the relevant environment.
- **Process Fragments:** Depending on the problem at hand, possible solutions include different types of teaching, like teacher-centered teaching, group or single person work, group discussions etc.. Additionally, when there are acute social problems at hand, working with the parents and the child's social environment may provide positive influence.
- **Problem List:** For some children it is known that talking to their parents can be hard - in this case, different approaches may be required (like adding an advisory teacher with special training to the discussion)
- **Bonus List:** For some children it is known that they prefer certain types of teaching or discussion when dealing with problems. In those cases, these types of working with the children will be preferred.
- **Triggers:** Triggers which require some change in the way the children are handled include observably different social behavior, difficulties with learning, etc.
- **positive Goals:** child feels better about himself, symptoms are reducing
- **negative Goals:** child shows harder symptoms, refuses to learn or socialize
- **Individualism:** Each child has its own individual circumstances which require special care of their teacher.
- **Limitation of available resources:** As in other domains, the available resources - in this case, mainly the time and number of available teachers - are limited. The teachers have to take care of all the students which are in the classroom and provide the best possible care for them.
- **Ambiguity of triggers:** Most triggers, like noticeable social behavior can be based on a number of problems: it may be problems at home or with friends, physical conditions, or problems with the teacher.
- **Diversity of possible solutions:** Depending on the situation at hand there are multiple possible solutions for one problem, including to work exclusively with the child, trying to communicate with its parents, the headmaster, a specially trained advisory teacher, or - in the worst cases, child protective services.

Discussion: In this scenario, the parallels to the nursing domain and to FIPS in general were obvious: Just as patients who have their individual problems, children with special needs also require attention for their individual needs. If a child shows some symptoms (as described above) it is the teacher's job to identify the problem and to find an effective way of dealing with it. If the teacher is experienced in his field of work, he had similar cases before, and uses this knowledge in order to find the best way to help the child.

Expert Interview: Doctoral Program Supervision

We interviewed two participants in the doctoral program of the University of Vienna. During the interviews it became clear that the doctoral program can also be seen as a FIPS. While all students have their individual project, there are some common characteristics between the different students and projects which can be utilized to analyze problems and find solutions. The basic research plan is typically laid out during the first year of the PhD studies, but it has to be adapted many times during the course of the work, whenever issues or new ideas come up. For example, if a student realizes he or she requires some special equipment which is hard to get, he or she may have to adapt his or her work plan. There might be different solutions for the same challenge. For example, if a certain kind of technical equipment, like a special microscope, is required to complete a part of the work, for some projects it may be it easier to adapt the work in a way that this equipment is no longer needed. For other projects, the students may have contacts to institutions which can provide such a microscope.

- **Subjects:** The subjects are the projects the PhD students have to complete in order to finish their studies.
- **Process Instances:** Each student has a plan he follows in order to complete his PhD project. Since this plan has to react constantly to the current situation, and one usually does not know how exactly the project is going to end until it is declared to be finished, it can be seen as a long running process instance.
- **Organizational Units:** Professors, administrative personnel, examining board, etc. are all organizational units which play a role during the course of a PhD project.
- **Environment:** The environment is mainly the university the students are working in. Possible alternatives include companies which are in some form associated with the project.

- **Process Fragments:** Things students can do to solve their problems are highly depending on the situation at hand, but clearly their is a big variety. For example, if a student has no data to evaluate his studies, possibilities include to do a literature study, ask related companies for collaboration, or generate new data on his own.
- **Problem List:** Some process fragments may not be applicable in certain situations. For example if it is known that in the field of research data is hard to get, some of the solutions mentioned above may not be applicable at all. If the student has already asked all possible contacts which could provide him data, such steps may not be the optimal way to proceed.
- **Bonus List:** In some situations, a certain fragment may return better results. For the above mentioned fragment of asking some companies for data, a bonus feature could be that there are some personal connections to a company which can provide the data.
- **Triggers:** Examples for triggers which make an adaption necessary include new problems which have to be solved, new ideas which may change the course of the dissertation etc.
- **positive Goals:** Problem is solved, new data is available, idea is integrated
- **negative Goals:** Time is lost, the student gives up on his project, etc.
- **Individualism:** Each PhD student has his own, individual thesis he has to complete. Although many theses can be approached in a similar way, each thesis has its individual requirements and work plan which has to be completed.
- **Limitation of available resources:** Many resources (like lab equipment) are only available in a limited fashion and students may have to plan their work according to the accessibility of such resources.
- **Ambiguity of triggers:** A student's problem description may not always indicate for the same problem. For example, a student who is not able to get any data for his current project may actually not have access to any data sources, or may not be able to use them correctly.
- **Diversity of possible solutions:** There are always multiple solutions to a given problem; for example, if the student has no access to data sources, he may try to redesign the project so that the data sources are not needed any more, try to get new contacts which may provide him with the required data, or find some other solution.

Discussion: During the interviews it became clear that the doctoral program can also be seen as a FIPS. In contrast to other domains however, each student only has one individual project (i.e., one subject in terms of a FIPS) he is working with. While in the other three domains people were identifying and handling problems for several different subjects (e.g., guests, support cases, children) each student only deals with one subject.

Conclusion: Altogether 8 experts from 4 different domains were interviewed. All of these experts were able to identify some commonalities between the domains they are working in, and a FIPS. Together with the domain experts, we identified the basic building blocks of a FIPS which we discussed in the last section. For each of the four domains described above, these building blocks have been discussed in detail and allocated with data from each domain.

In addition to the building blocks which are based on related work, and the *bonus list* identified during the expert interviews, the following special properties of a FIPS were encountered while analyzing these basic building blocks with the domain experts. The following list defines these properties:

- **Individualism:** Each subject, and each situation where a trigger occurs, has its very individual properties. For example, in the software development domain, each support case has its own set of data elements which describe the situation and the case itself.

- **Limitation of available resources:** In each setting, the resources which can be used to solve the problematic situation are limited. This usually leads to a prioritization of the problems: For those which are more important, more resources are spent. In the software domain, a crucial support case naturally receives more resources in order to solve it. On the other hand, in the special needs school setting, such a prioritization is not that simple: Every child needs attention, and the available resources have to be split up in a good way.

- **Ambiguity of triggers:** The same trigger does not have to mean that the same problem has to be solved: In the special needs school, two children can show the same symptoms (e.g., they cannot concentrate any more), but the reasons behind those symptoms can be very different.

- **Diversity of possible solutions:** If we know what the actual problem is, still multiple solutions are possible. It highly depends on the current situation which one is chosen, i.e., the workload of the available resources, the history of the individual etc. In the special needs school it depends to a big part on the condition of the child which solutions can be chosen in order to make him or her feel better.

7.3 Designing a Tower Defense Game as an Experimentation Service for FIPS

The goal of this chapter is to create a game-based experimentation service for FIPS, which can be used to (a) evaluate analysis and representation techniques for process change operations, and to (b) create process change and execution logs which can be used as a basis for such an evaluation. Similar approaches, where simulators have been used already in existing literature, e.g., the Alaska simulator [83, 84], which serves as an exploration and analysis tool for different types of process flexibility. Based on the building blocks and properties discovered in the last section, the goal of the remainder of this chapter is to create a game-based experimentation setting for FIPS, which enables the simulation of FIPS in a generalized way ($\rightarrow$ Q2, Q3). Using such an experimentation service as a reference, researchers can develop and test new approaches for supporting users who work in such a setting using a controlled environment. Additionally, the service enables the collection of data that is essential for user support features [8, 37, 12], i.e., process execution and change logs.

The following section first discusses why we decided to implement a tower defense game as an experimentation service for FIPS. Following, we introduce the basic ideas, game mechanics, and the user interface for the player. After that it is explained how this gameplay relates to FIPS building blocks and procedures. Finally, the basic architecture are shown and we introduce the data players generate while playing the game.

Tower defense games (cf. e.g., [85]) are strategy games where the player has to defend a set of objectives, e.g., some kind of base. Usually, the player has some defense systems to choose from (build a cannon, train some soldiers, etc.) which can then be placed at certain locations (i.e., towers) to defend those bases. In certain intervals, waves of enemies attack these towers and try to destroy them. By placing his defense systems, the player tries to counter the enemy forces and avoid his bases from being destroyed.

We chose a tower defense game as a basis for our evaluation service for several reasons: First, tower defense games are a well known type of game, thus, the basic game mechanics will be quite easy to understand for new players. Second, tower defense games can be round based. This means that the player can take as much time as he requires to make his decisions. This stands in contrast to real-time strategy games, where the player has to make his decisions very fast in order to keep up with the game. A very popular representative of a round based strategy game is chess: Here, both players can usually take a long time to decide what to do next. Since the goal of our game is to represent a FIPS, we decided to implement a round based strategy game: As in a real FIPS, where the responsible actors have

to plan their next steps, the player should have enough time to plan what to do next. Third, a tower defense game does not depend on a real domain: For most FIPS, the person responsible for deciding what to do next has to be a domain expert. Depending on the domain, it can take a long time to acquire the required knowledge. In a domain independent setting such as a tower defense game, no long training is required: Of course, it takes some time for new players to know exactly when to use which defense system, but in contrast to learning the details of nursing science, this can be done very fast. Fourth, the domain independence also leads to the advantage that anyone interested can download and play our game. Thus, we have a broad base of possible players who generate the data we can use for further evaluation.

Aside from the players who actually play the game, our targeted audience are scientists: First, the generated data can be used to develop and evaluate algorithms which work with FIPS data. Second, using our game as an experimentation service, scientists can evaluate approaches to support users in FIPS domain independently. Imagine a service which aims at supporting FIPS users. Our game service could be utilized to compare two groups of FIPS users: Those who receive some kind of support when making decisions, and those who do not receive any support. If the supported group outperforms the group who did not receive support, it can be a hint that the support service might be useful. Due to the domain independence of our game service, a broad range of users can be included in such an experiment.

7.4 Gameplay and User Interfaces

The idea of *Apelands*, the game to be developed[2] is that the player slips into the role of a commander who has to conquer and defend villages on an island. (The villages are equivalent to the bases in the definition given above.) Each of these villages has its own set of parameters which make them individual. While some villages are closer to the sea, others are in the middle of the woods, and others close to mountains. These villages are attacked by enemy armies in varying intervals. Who these armies are and how they attack largely depends on the parameters and the history of the villages. Some enemies will preferably attack villages closer to the sea, while other enemies are more often seen in the woods and will preferably plan their attacks there.

7.4.1 The Basic Game Loop

The game is round based, meaning that the player can take as much time for his decisions as he needs. In the game each round is referred to as a day. In order to

[2] Available at: `http://cs.univie.ac.at/project/apes`

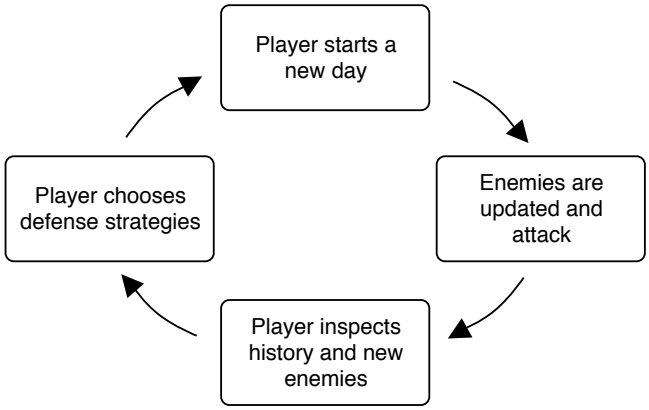


Figure 7.2: The Basic Game Loop of Apelands

defend his villages, the player can plan the defenses for the days to come. For each village, there exists exactly one defense plan, which is implemented as a highly flexible and individual process instance. In this process instance, all the actions which have to be carried out in order to defend the village are defined. The game loop is depicted in Figure 7.2.

When the player starts a new game, he immediately starts the first day. Each day, as a first step the enemy armies are updated. This works in two steps: First, villages which have not yet been under attack have a certain chance of being attacked by a new enemy army. Second, villages which are already under attack, will be attacked by a new wave of the currently attacking enemy army. How the enemy armies and the attacking waves are defined is explained in more detail in Section 7.4.2. The success or failure of an enemy army attacking a village is determined by the defense strategy chosen by the player: If the player did not set up any warriors who should defend the village, or chose the wrong warriors, the enemy army will attack the village, find no or little resistance, and kill some of the population of the village. Whenever the population of a village reaches zero, the player loses the game; thus, he is interested in protecting his population as good as possible. The player sees the status of his villages (i.e., which are under attack, and what has happened in this previous step) after the attack is over (cf. Figure 7.2). Now, he can inspect the history of his villages, and take a look at which enemies will attack him on the next day. Based on this information, he can decide which defense units he wants to put in place. As described in Section 7.4.3, there exist many different ways of defending a village, including warriors from different races typical to a fantasy setting, spells, weapons and magical oils. *Apelands* is designed based on a concept similar to the popular game *Rock-Paper-Scissors*: For each game unit (warrior, spell, oil and weapon) there exists a set

of game units against which it is strong, and a set of game units against which it performs weakly. Thus, the player has to decide for a good set of units to counter the incoming enemy. Knowing which unit is a good countermeasure against an incoming enemy can be deduced from the unit's documentation, or based on the experience of a player: For example, all warriors of the race *dwarf* are known to be strong in the mountains, but weak on grass or beach terrain. Thus, such warriors should be preferably played in mountainous regions. Once the player has put all his defense units in position, he starts the next day, and the next cycle begins with updating and attacking of the enemy armies, which will fight the defense units the player has put into place in the previous round.

7.4.2 Enemy Armies

The enemy armies attack the villages over multiple days. On the first day, they usually show up with light units who do not do much damage, but the player already notices them. The longer the player does not react properly to the threat, the stronger the enemy army gets; ultimately destroying the village. Depending on the choices a player makes, the enemy army gets discouraged or encouraged to continue their attacks.

Enemy armies in *Apelands* are built upon the following information:

- a *name*, so that the army can be recognized by the player,
- a *process model*, describing the waves of the incoming enemy armies, and the conditions when the defense will be successful, as described below,
- a set of *morale modifier*, which define when the enemy army will be driven away by the defense of the village,
- a *lore* text, describing the enemy army in a way fitting the fantasy setting of the game,
- a set of *presence* modifiers, which define the probability of an enemy army showing up at a certain village

Figure 7.3 shows an example for an enemy army process: Here, an enemy army attacks the village of a player over maximally four rounds: If the enemy army is defeated on one day (i.e., their morale is below a certain threshold), they stop their attacks; otherwise, they continue to attack the village with more or stronger warriors.

In order to decide, if an enemy army will continue to attack the village of a player, or stop attacking, a `morale` value is introduced: At the beginning of the

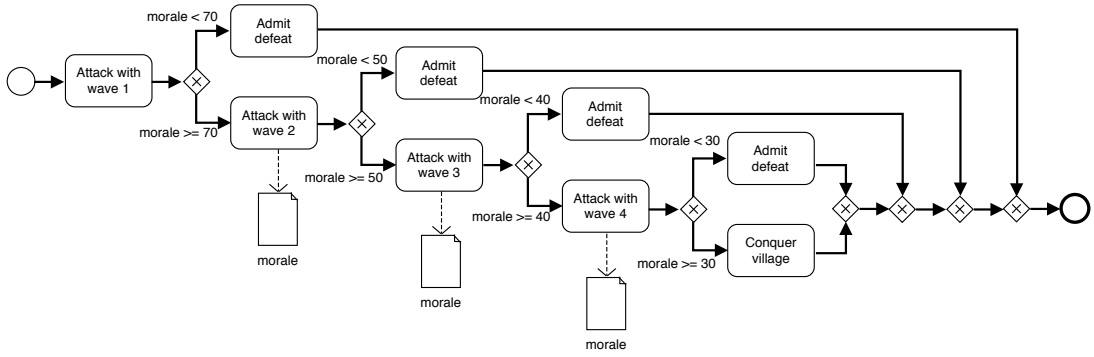


Figure 7.3: An Example of an Enemy Army Process in Apelands

first round, it is set to 100; depending on the defense strategy the player chooses, it can be incremented or decremented. Thus, if a player chooses a good defense, it increases the chance of routing the enemy army out. In the example given in Figure 7.3, a morale value of at least 71 is required for the enemy to attack for a second time. Factors which influence the morale of an enemy army are based on different concepts, e.g. *magical alignment* (cf. Section 7.4.4), or a certain race being present in the defense of a player. It has to be noted that it is not directly visible for the player which choices will have which effect on the morale of the attacking enemy army. However, certain hints in the description are present. For example, the description of an army reads as follows: *"The Troll Raid is bound to the forces of corruption. Thus, any spell which contradicts their basic needs increases the chance of routing them out."* Thus, if players focus on nature spells, they have a good chance of removing the threat from the village. A more detailed description on spell domains is given in Section 7.4.4.

Following, a player has to choose his defense not only to counter the incoming enemy units, but also to find a good way of decreasing the overall morale of the enemy army.

7.4.3 Building a Defense

The player controls four different buildings which cannot be attacked by an enemy directly. Their sole purpose is to produce different parts relevant for defending the villages. All buildings have a maximum number of items they can produce on a single day. If their workload is fully used, they cannot produce any more on that day.

- **Barracks:** In the barracks, warriors of different types are trained. Among others, the player can choose between a knight with a lot of health points,

an archer who does additional ranged damage, a mage who is a proficient spellcaster, and a soldier who does melee damage.

- **Forge:** A warrior cannot do much without a weapon. In the forge, the player can create multiple weapons such as swords, maces, axes, polearms, longbows etc. Each of these weapons has individual properties, which make them more or less effective depending on the warrior who carries them.

- **Mage Tower:** In the mage tower the player can create spells which are cast by the defense units either to protect themselves or to harm their enemies.

- **Alchemy Lab:** In the alchemy lab the player can prepare oils which add an additional effect to the weapons, and thus enhance their effectiveness.

Each time a new enemy army shows up at a village for which the player has not planned any defenses yet, he is supposed to do the following steps:

1. **Choose Warriors:** In a first step, the player chooses the warriors which make up his army. These warriors are from one of eight different races (human, dwarf, elf, caplani, troll, fishman, elemental and machine) which all have advantages and disadvantages depending on the terrain of the village and the enemy they are facing (cf. Section 7.4.4). The player can choose as many warriors as he wants for each day, as long as the total number of warriors doesn't exceed the maximum workload for the barracks for this day.

2. **Prepare the Army:** Before sending his army into battle, the player has to allocate the weapons and spells to his warriors. Basically, all infantry can carry all kinds of weapons and spells, but some will perform better with certain types of weapons than others. For example, an archer will do more damage using a longbow than a mage would do with the same weapon. Also, the effectiveness of the weapons depends on the enemy the player is facing: Some weapons will perform good against certain races, while others will not yield good results. Preparing the army for battle consists of the following steps:

   - Choose Spells
   - Choose Weapons
   - Choose Oils

   Note that these steps can be executed in (almost) any order. The only constraint is that oils have to be applied to certain weapons; thus, in order for a warrior to carry a certain oil, he has to have a weapon equipped first.

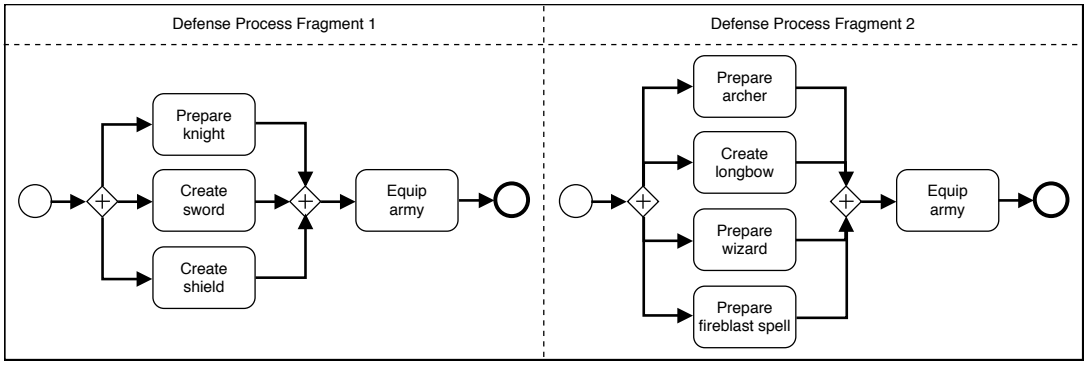


Figure 7.4: Two fragments for a village's defense process

3. **Execute Adaptation:** When the player has set up his army for the following days, he adapts the village's process instance and sends his troops to the village which is attacked.

Once the player has chosen his warriors, and has equipped them with weapons, oils and spells, he can add his choices to the defense of the village. In this step, the choices he made are sent to the game server as a process adaptation.

For each village, there exists exactly one process instance which contains all defense activities which have been added to this village's defense. Figure 7.4 shows two process fragments which represent possible adaptations to a village's defense process: In the left pane, one warrior with two weapons is created; in the right pane, in total two warriors, one spell and one weapon are created. The last step of each defense process fragment is always the *Equip army* activity: When this is executed, the game is notified about the created warriors and items, and which warriors are equipped with which spells.

Since the training of the warriors, and the creation of the spells, oils and weapons can be done in parallel, each activity resides in its own parallel branch, containing only one activity. In order for the game to know which item respectively spell is supposed to belong to which warrior, this information has to be present when the item (warrior) is created. Thus, the activities which are used to create the spells, weapons and warriors contain such information. The XML snippet in Listing 7.4.3 shows the first two activities from the process fragment from the left pane of Figure 7.4, i.e., adding the archer and his longbow to a village's defense process instance. Each activity which adds a warrior to a process instance has a parameter tag called `warrior_id`, which references the warrior for the weapons and spells which will be equipped later. Each activity referencing a spell, oil or weapon has two specific tags: `belongs_to` references the `warrior_id` of the warrior which is supposed to equip this item, and `belongs_slot` references the slot this item will

be equipped to. For the example given in Listing 7.4.3, the weapon *Longbow* will be equipped in the left weapon slot of the warrior with the `warrior_id` 0, which is the archer created in the activity above.

Listing 7.1: Change Log in MXML

```
<parallel>
  <parallel_branch>
    <call id="id1" endpoint="village">
      <parameters>
        <name>'Archer'</name>
        <type>'warrior'</type>
        <day>3</day>
        <resources>
          <resource>
            <name>'barracks'</name>
            <count>'1'</count>
          </resource>
        </resources>
        <warrior_id>0</warrior_id>
      </parameters>
    </call>
  </parallel_branch>
  <parallel_branch>
    <call id="id2" endpoint="village">
      <parameters>
        <name>'Longbow'</name>
        <type>'weapon'</type>
        <day>3</day>
        <resources>
          <resource>
            <name>'Forge'</name>
            <count>'1'</count>
          </resource>
        </resources>
        <belongs_to>0</belongs_to>
        <belongs_slot>'left'</belongs_slot>
      </parameters>
    </call>
  </parallel_branch>
  <!-- two more parallel branches for the wizard and the
     fireblast spell -->
</parallel>
```

As the game continues, multiple fragments may be added to a village's defense

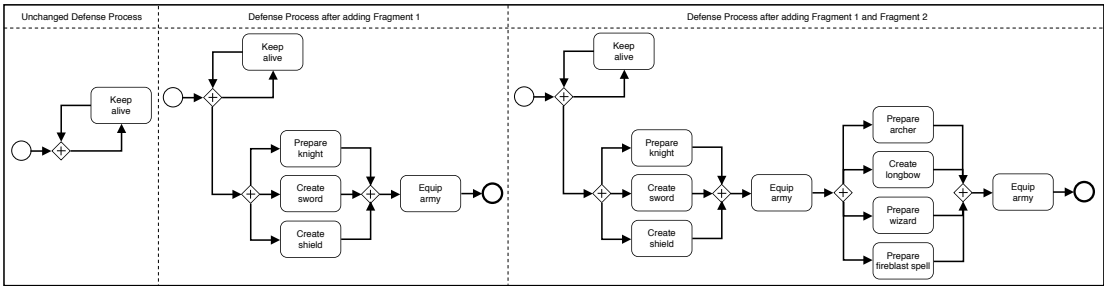


Figure 7.5: Adaptation example of a village's defense process

process: This is visualized in Figure 7.5. The first pane depicts the unchanged defense process, which only contains one task, i.e., `Keep alive`. The only job of this activity is to keep the process instance alive, even when no other activity is to be executed. This can happen when (a) a new defense process instance is created, but no defense activities have been added yet, or (b) if defense activities have already been added, but all have been executed. In the CPEE [64], the workflow engine this game is based on, such an instance would immediately switch to the state *finished*; thus allowing no further adaptations or activity executions. In order to prevent such a scenario, this loop exists. After the first adaptation, which adds process fragment 1 from Figure 7.4 to the village's process instance, the BPMN representation is depicted the second pane of Figure 7.5. The third pane depicts the process instance after fragment 2 has also been added to the village's process instance. If this fragment is scheduled to be executed after fragment 1 has been finished (e.g., because it has been added for a later day in the game) it would be possible to add it into a different branch parallel to fragment 1, for reasons of understandability we chose to add it after the preceding fragment has been executed, as depicted in the third pane.

When a player adapts a defense process instance, it is still possible to add additional defense mechanics (i.e., additional process activities), or to remove or swap existing process activities, as long as they have not been executed yet. Thus, aside from inserting new process fragments, the following high-level process change operations [21] are supported in *Apelands*: *delete* and *replace*. Figure 7.6 shows examples for each of these change operations. In the following, we will discuss the semantics of each of these change operations in detail.

Delete: Whenever a player decides that he does not want a certain weapon, spell, oil or warrior to be present in the defense of one of his villages, he can remove them by clicking the *Remove* button in the user interface (cf. Section 7.4.5). Here, some special cases have to be taken care of:

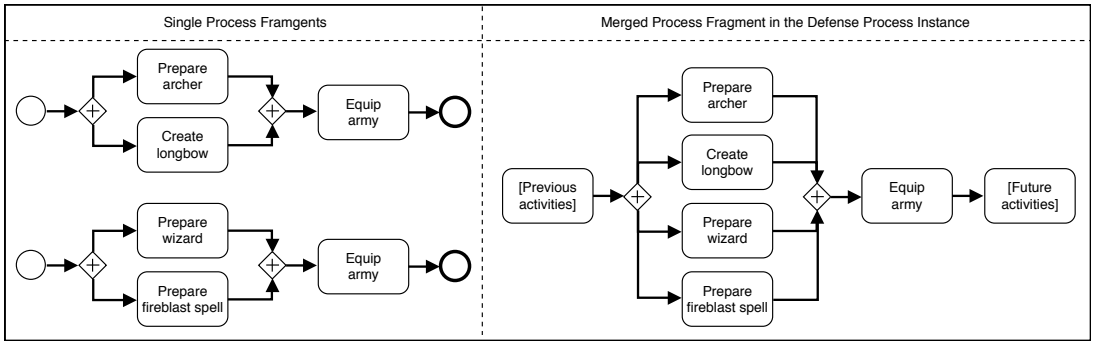


Figure 7.6: High-level change operations supported in Apelands: Delete, swap and replace

- When a warrior is removed, all weapons, spells and oils which are assigned to this warrior, have to be removed as well from parallel branches. Thus, each `call` element, which contains a `belongs_to` tag referencing the `warrior_id` of the warrior which is to be deleted, has to be deleted as well (cf. Listing 7.4.3).

- Similarly, whenever a weapon is removed, the oil, if it is applied to the weapon, has to be removed as well.

- If the `parallel` tag containing the defense activities is empty after the (cascading) delete operation(s), the whole branch has to be removed. This prevents single `Equip army` activities being executed, without any definition of an army before.

Replace: In *Apelands*, replacing activities is necessary whenever the assignments of a weapon, spell or oil changes; e.g., if a weapon has been assigned to the left weapon slot before, but is now supposed to be in the right weapon slot, or when the weapon, spell, oil or warrior is to be replaced with a different one.

For each day in the game, one *Equip army* activity exists, telling the game to equip the armies and prepare them for battle. Thus, when two fragments have to be inserted for the same day, because the player added two fragments consecutively, these fragments have to be merged before insertion. This is visualized in Figure 7.6. Here, two process fragments as depicted in the left pane (one adding an archer and a longbow, the other adding a wizard and a fireblast spell), are added to the process instance. It does not matter, which fragment is added first: In the end, the second `Equip army` activity is removed, and the two parallel splits are merged into one, containing all defense process activities.

Race Fight Bonus Overview

	Human	Dwarf	Elf	Caplani	Troll	Fishman	Elemental	Machine
Human	100%	150%	130%	100%	50%	80%	70%	120%
Dwarf	70%	100%	150%	80%	130%	50%	120%	100%
Elf	100%	150%	100%	130%	120%	80%	70%	50%
Caplani	150%	50%	80%	100%	100%	130%	120%	70%
Troll	70%	80%	50%	120%	100%	150%	100%	130%
Fishman	100%	50%	70%	100	130%	120%	150%	80%
Elemental	120%	150%	130%	100%	80%	50%	100%	70%
Machine	100%	120%	150%	130%	80%	70%	50%	100%

Races and Terrains

	200%	130%	100%	70%	30%
Human					
Dwarf					
Elf					
Caplani					
Troll					
Fishman					
Elemental					
Machine					

Legend: Grassland Woods Mountains Desert Beach

Figure 7.7: Races and Terrain Types in Apelands

7.4.4 The Basic Game Mechanics of Apelands

As mentioned above, *Apelands* is built upon the concept of *Rock-Paper-Scissors*: For each enemy army, counter measures exist which can be used to defend a village successfully. In this section, we will describe how this concept has been implemented on multiple levels: Based on the races, the spells, weapons, the terrain, and magical alignment of a certain area.

Terrain Types and Races

In *Apelands*, there exist five different terrain types, including *woods*, *grassland*, *mountains*, *beach*, and *desert*. These terrain types have a huge influence on the performance of the races, as depicted in Figure 7.7. The races are explained in the remainder of this section.

In *Apelands*, there exists a total of eight different races. When the user chooses which warriors he wants to send into the fight against an incoming enemy army, he has to consider two things: First, the terrain of the village has an influence on how much damage the warrior can do. For example, dwarfs do double their standard damage on mountainous terrains, but only 30 percent when they have to fight on the beach. Second, the incoming enemy army also consists of warriors of a certain race. Some races are strong against other races; for example, elves receive a bonus of 40 percent on their standard damage when fighting against trolls. Figure 7.7 summarizes the different races, and the influence of the terrain types. Additionally, it is shown which races perform strong or weak against which other races. Thus, when choosing the races for a set of warriors, the player has to balance the terrain the fight will take place in with the races of the incoming enemy in order to find a good middle ground.

The Warriors and the Weapons

The warriors in *Apelands* are the units which carry weapons, cast spells, and try to defeat the enemy. Besides his race, each warrior has a specific class which defines his strengths and weaknesses. For each race in the game, there exists one warrior for each class. Each class has a preferred style of combat, thus resulting in different weapons or abilities being more or less effective. In *Apelands*, in total 22 weapons of the types *melee* (swords, axes), *ranged* (bows, crossbows), and *shield* (for absorbing damage) exist. Additionally, *spells* can be cast, which are described in more detail in the next section. The weapons and spells perform differently for the four classes available:

- *Melee:* The melee is a warrior optimized for close quarter combat. He deals 30 percent increased damage with melee weapons, but the effectiveness of ranged weapons, shields and spells are reduced by 30 percent.
- *Tank:* The tank is also a close quarter combat fighter, but more experienced in avoiding damage. Thus, the effectiveness of shields (which absorb damage) is increased by 50 percent, the effectiveness of melee weapons is reduced by 30 percent, and the effectiveness of spells is reduced by 50 percent.
- *Caster:* The caster is profound in all magical ways, making him the ideal unit for casting spells. Thus, all weapons (melee and ranged), and shields receive a reduction of effectiveness by 50 percent, but spells are twice as effective.
- *Ranged:* The ranged warrior receives a bonus for all ranged combat weapons by 30 percent, but melee weapons, shields and spells are reduced by 30 percent.

Based on the incoming enemy army, different types of warriors are more or less effective: For example, it could be advantageous to focus on ranged damage against an army which only consists of melee fighters, as long as they can be defeated before they reach the own line of defense. In the next section, the weapons used by the warriors are introduced.

Spells and Oils

In *Apelands*, there exists a total of 25 spells, split up in six domains, which are depicted in Figure 7.8. As for the races and the terrains, it is based upon a concept similar to *rock-paper-scissors*: There are two groups of spell domains with three domains each; one "good", the other "evil". For each good domain, there exists exactly one "evil" counterpart; e.g., for the *nature* domain, there exists *corruption*.

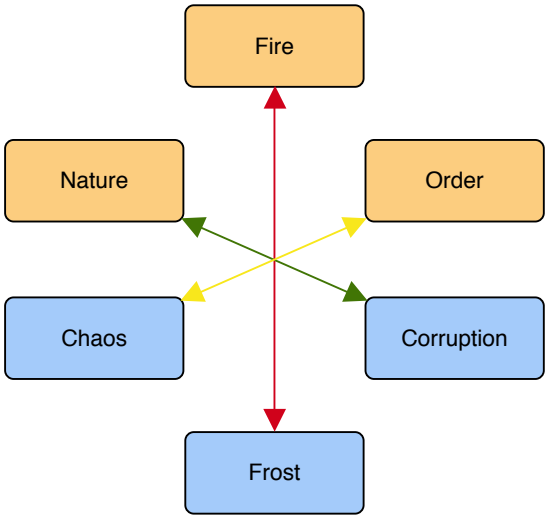


Figure 7.8: Magic domains in Apelands

The spells of two contradicting domains are more effective against each other; so a character who has recently cast a *corruption* spell will receive more damage from *nature* spells during the remainder of the fight. Thus, for any incoming enemy army, the player has to know which spells they will use, and act accordingly. Thus, if an enemy is known to attack with *corruption* spells, the player might want to consider using *nature* spells, but must be aware of the fact that the *corruption* spells will be more effective against his own units. Spells of contradicting domains receive a bonus of 50 percent, while spells of contradicting groups receive a bonus of 20 percent. Following, a *nature* spell against a unit who has recently cast a *corruption* spell receives a bonus of 50 percent, while it only receives a bonus of 20% against a target who has recently cast a *frost* spell. In addition to their bonus against units, spells also shift the *magical alignment* of a village: For each village, the alignment is set by the spells being cast, just as it is done for the warriors. Thus, if multiple *frost* spells have been cast at a village, its alignment will shift towards the frost domain, and frost spells will receive a bonus depending on how far the village's alignment has been set towards the frost domain. The value ranges between 0 and 100; thus, at its highest point, frost spells will receive a bonus of 100 percent. Alignments of villages are kept over several rounds; thus, the player can use this information to prepare his army for future rounds.

Spells are created in the *mage tower*; thus, they require resources from the mage tower when a player wants to create a spell for his defense.

Oils work similar to spells: In *Apelands*, there exist a total of eight oils, divided over the same six domains as the spells (cf. Figure 7.8). They are applied to the unit's weapon; thus, it is only possible to equip a unit with an oil, if a weapon

has been equipped as well. Oils are created in the *alchemy lab*; thus, they require resources from the alchemy lab when a player wants to create an oil for his defense.

Summary

In this section, we have described the different game mechanics which are relevant for deciding for a good defense strategy to counter an incoming enemy army. By implementing several concepts (races, classes, spell domains, terrain and weapon types) which have an effect on each other during the fight, finding the best adaptation for a given situation is not a straightforward task. However, the more experienced players become, the easier it will be for them to find a good adaptation for any given situation. Following, just as in a realistic setting such as the nursing domain, where experienced nurses already know situations and how to handle them, experienced players will know what to do in a certain situation, while new players will rely on support (either by experienced players, or a support system).

7.4.5 The Game Interface

In this section, the game interface is described in detail. Players interact with the game using this interface only.

Island overview

The game interface is implemented based on a 3D view on the island.

Figure 7.9 shows the user interface in the game of the island: Here, the player controls one specific village by the sea. As can be seen from the alert box, and the swords icon, the village is currently under attack. The bottom part of the screen is divided in three panels: The left panel shows details of the currently selected field. Since in this case the village is placed on the beach, the *main terrain* is set to *beach*. This information is a hint to the player, when he decides for which races will perform well, and which won't (cf. Section 7.4.4). The middle panel of the bottom part of the screen shows the alert box, and a button (currently hidden by the alert box), which allows for activating the next round. The rightmost panel contains the resources available.

Overview over one Village

Figure 7.10 shows the overview over one village, as seen by the player who is currently inspecting one individual village. The 3D view on the screen shows the wave of the enemy army which will attack the village on the next day (top row). Thus, the player has to find a good defense for this enemy. In the bottom row, the

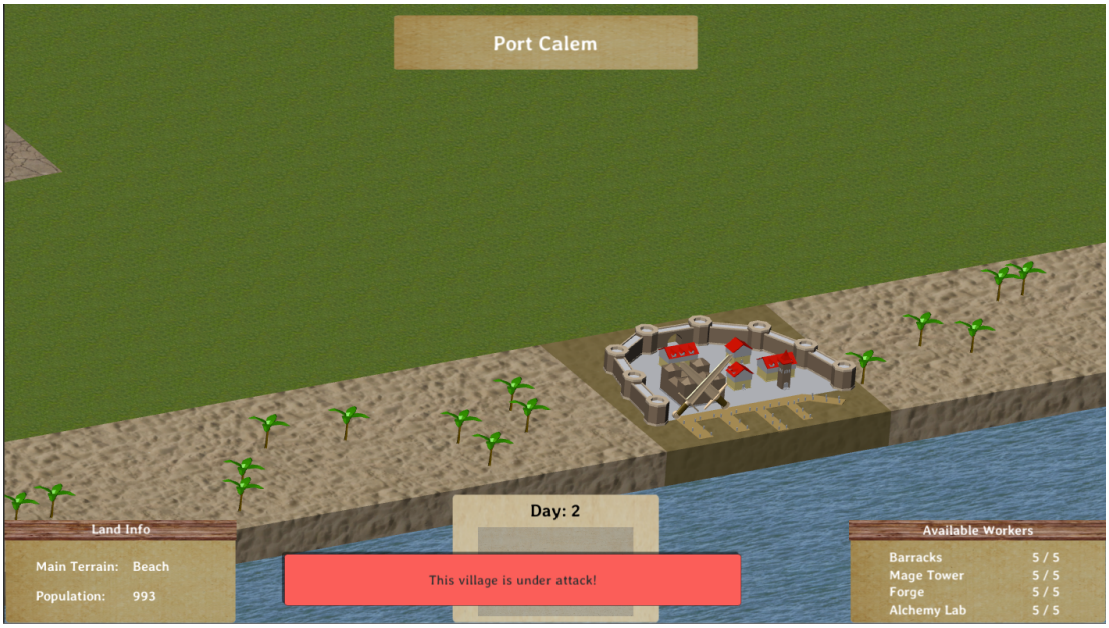


Figure 7.9: Overview over the island with one village being attacked

defense of the player is shown, as soon as the player has added some warriors to his defense. In the interface shown in Figure 7.10, three human warriors attack the village, and no defense has been set up yet (since there are no 3D models shown in the bottom row).

Below the 3D interface, several panels are shown: On the left, the main terrain of the village and its current population are sown. Next, the *magical alignment* indicator shows the current alignment of the village for the three axes described in Section 7.4.4. On the right side, the workers available in the four different buildings (barracks, mage tower, forge and alchemy lab) are shown. Additionally, buttons exist for inspecting the history, adapting the process instance, showing details of the attacking enemy army, and going back to the island overview described above.

Constructing the defense

As described in Section 7.4.3, constructing the defense is built upon the following steps:

1. Choose a warrior
2. Choose spells
3. Choose weapons

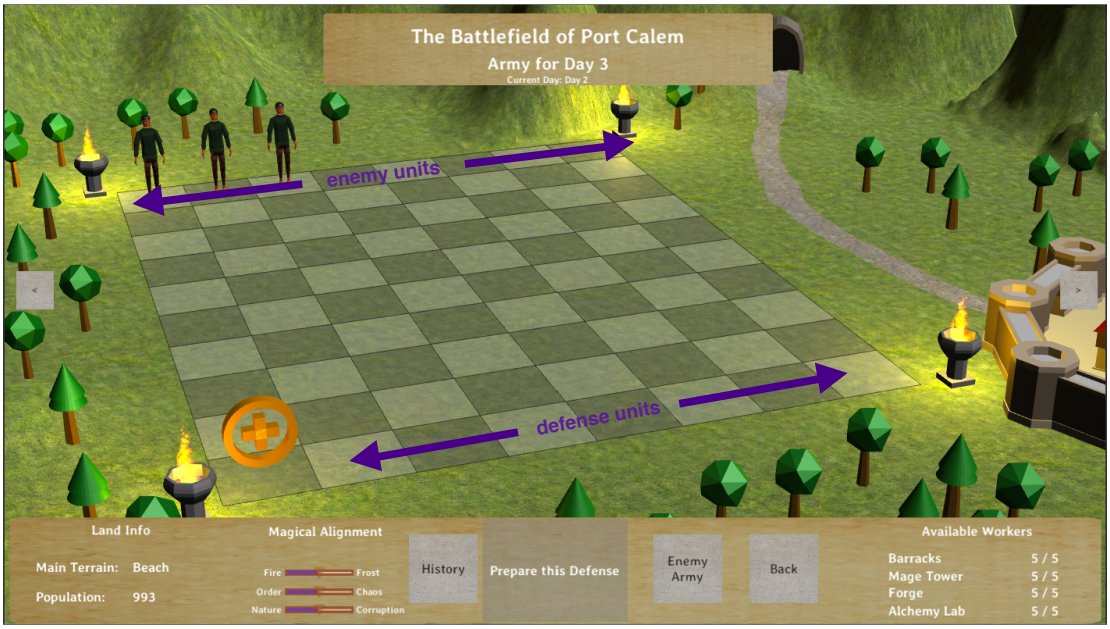


Figure 7.10: Overview over one specific village

4. Choose oils

In this section, we will show how these steps are supported by the game's interface.

The first step, *Choose a warrior*, is being initiated by clicking the orange *plus* button shown in the island's overview (cf. Figure 7.9). When a player clicks this button, the interface as shown in Figure 7.11 is displayed. For each race available to the player, the possible warriors are displayed. By clicking the arrow icons next to the race name, the race to be displayed can be switched. Figure 7.11 shows all units available for the *Elemental* race. Whenever a player hovers with the mouse over one specific warrior, a short text description of his strengths and weaknesses is displayed, thus indicating, for which situations a certain race and class might be favorable.

Once the player has chosen a specific warrior, a 3D model is displayed in the row where the defense warriors are positioned. As a next step, the player has the possibility to equip this warrior with weapons, spells and oils. For this, he can inspect the warrior's profile by clicking his 3D model, which opens the interface depicted in Figure 7.12. In the profile, multiple slots for adding weapons, spells and oils are available. For each slot, a button exists which, when clicked, removes the current item equipped in this slot, if it is not empty.

When the player clicks an icon, he can add weapons, spells or oils. Spells and oils are both divided into six different domains, as explained in Section 7.4.4.

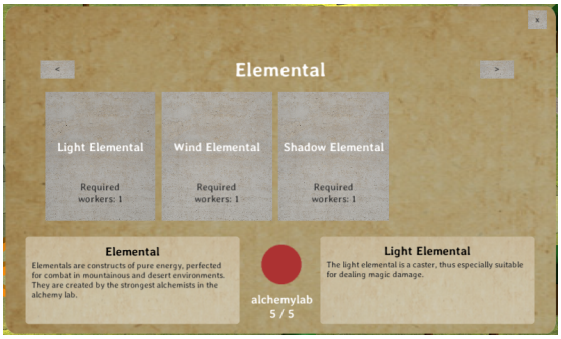


Figure 7.11: Selecting the basic unit for the defense of the village

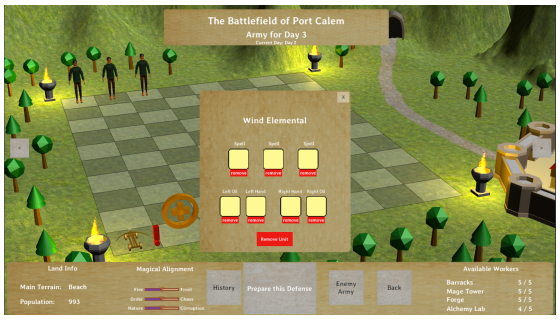


Figure 7.12: Clicking the 3D model of a warrior opens his profile

Figure 7.13 shows the domains available for spells. If the player hovers over a certain domain, a text field below the selection displays a short information, against which other domains the chosen domain is particularly strong.

When a player has chosen a spell domain, all spells present in this domain are shown. For each spell, the effects are described as text whenever the player hovers over the spell. When a player clicks a spell, it is assigned to the slot the player has started with.

Figure 7.13: Displaying the available spell domains

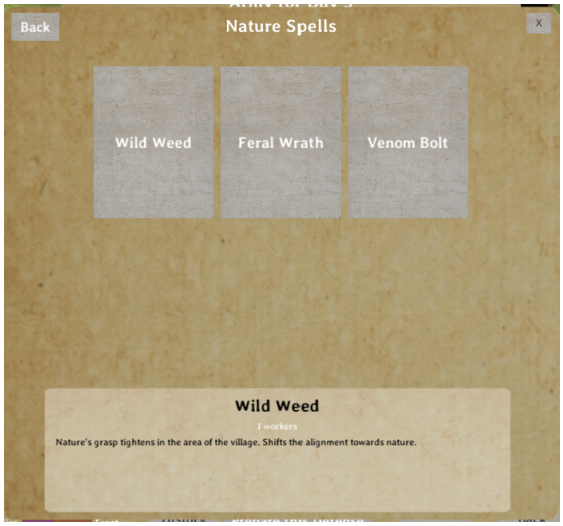


Figure 7.14: Displaying the spells for one specific spell domain

The interface for adding oils to a warrior's weapon looks similar to the one for adding spells: Thus, players who know how to add spells should be immediately

able to know where to find the oil they want to add to a warrior's weapon.

Aside from spells and oils, the player can also add weapons to a warrior's equipment. To do so, all he has to do is click one of the two slots for the weapons (left and right). In contrast to spells and oils, weapons are not divided into several domains. Instead, the player sees all possible weapons in one screen, as displayed in Figure 7.15.

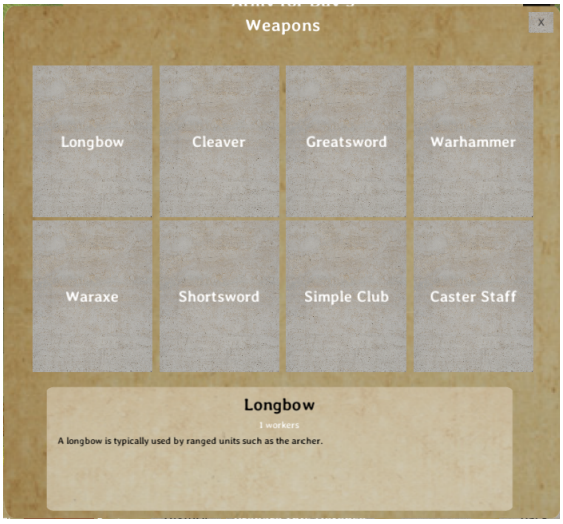


Figure 7.15: Selecting weapons for a warrior

Figure 7.16: The equipment of a warrior is displayed in his profile

The weapons, oils and spells the player chooses for his warriors are shown in the warrior's profile, as displayed in Figure 7.16. By hovering over the different slots which are equipped (indicated by a red circle), the warrior can see the weapon, spell or oil he has chosen. Clicking the slot opens the respective panel responsible for selecting the appropriate item type: Thus, the player can always change his spells, oils and weapons.

Inspecting the History

In order to see previous fights, and react accordingly, we have provided an interface for the player to see the history of each individual village. Figure 7.17 shows an overview over a village's history. Here, three fights were registered, whereas for days 2 and 4, the defense successfully defended the village against the incoming enemy army, and on day 3, the enemy army did overcome the defense due to poor choices of the responsible player.

Being able to inspect the history is an important interface in the game, as it helps the player to evaluate whether his choices of defense have reached the desired goal (i.e., driving the enemy away as fast as possible) or not. Aside from the summary, Figure 7.18 shows a more detailed analysis of the results of the fight: It can be inspected by clicking the respective row in Figure 7.17. This interface visualizes the change in the magical alignment during the fight, how many units participated in the fight, and which side won. By clicking the *Show Fight* button, the player can see an animation of the fight; i.e., the warriors fighting each other, and sees exactly which warriors survived the fight. Based on this information, he can see whether the adaptation he made has been a good choice against the current wave of the incoming enemy army, or not.

Figure 7.17: Overview over the village's history

Figure 7.18: Details of one specific fight in the history

7.5 Apelands as a FIPS

In this section, we will discuss how Apelands can be seen as a game-based experimentation service for flexible and individual process settings (FIPS).

7.5.1 Mapping the Tower Defense Game Concepts to FIPS Building Blocks

In the following we describe how the gameplay is mapped onto FIPS.

- **Subjects** map to villages which differ according to their topology, magical alignment and their history of enemy attacks.
- A **process instance** maps onto exactly one defense plan for each village which should be adapted every time a new enemy army shows up.

- **Organizational units** map onto buildings under the player's command that carry out the steps defined in the process instance, namely the barracks, the forge, the alchemy lab and the mage tower.
- An **environment** subsumes all resources and organizational units that have to be shared for the execution of a certain process, for example, a nursing home. In the tower defense game, all production buildings have to be shared for the set of villages in the game. Hence, each game forms its own environment.
- **Process fragments** map onto the defense plans a player creates in order to stop the enemy army. These defense plans / process fragments are inserted into the village's defense process instance.
- **Problem list:** For each fragment which can be inserted into a process instance there exist some situations where it will not be optimal to execute its steps, for example if the magical alignment of the village is contradicting the spells' alignment.
- **Bonus list:** While some spells may be problematic in some situations, they may perform well in others. If a village's magical alignment is the same as the spells in a fragment, carrying out the adaptation may yield more promising results.
- **Triggers:** Every time a new enemy army shows up at a village, its defense process instance should be adapted.
- **Positive goals:** If the enemy is destroyed or chased away, the battle is won.
- **Negative goals:** If the village loses population or too many resources have been used, the battle may not have found its optimal outcome.
- **Individualism:** Each village has its own, individual properties. This includes the terrain, number of rivers, and magical alignment.
- **Limitation of available resources:** The buildings can only produce a limited amount of soldiers and weapons to defeat the enemies each day. Their resources have to be split up among all villages.
- **Ambiguity of triggers:** When a certain enemy shows up, it does not necessarily mean that always the same kind of enemy will follow. For example, if some troll scouts show up at a village, it does not always mean that the same troll army is about to attack. The players will have to take a look at the details of the incoming enemy army to know which defense mechanisms will result in a good outcome.

- **Diversity of possible solutions:** Each enemy can be faced with many diverse defense tactics. Choosing the optimal one depends on the current situation of the village and the workload of the environment. Depending on the magical alignment of a village and the enemy army, different solutions may perform better or worse.

7.5.2 Comparing the game's adaption planning procedure to a real world FIPS

In Section 7.4 we presented the basic procedure of planning a defense strategy in our game. We showed what a player has to do as soon as he finds out that an enemy is attacking one of his villages, and how his plan is executed. In this section, we show how this basic procedure of planning a defense strategy for a certain village is similar to planning the upcoming actions in a real world FIPS. Table 7.1 shows the mapping between planning upcoming actions in a FIPS and in the game. This is shown by an example from the special needs school and compared to the actions a player has to take in the game.

Table 7.1: Mapping between the Tower Defense Game and a real world FIPS

Step	Special Needs School	Game Setting
Step 1: Trigger	teacher notices problems with a student's behavior	player sees attack of the village by some enemy
Step 2: Adaptions	teacher plans what to do to help his student	player plans the defenses for the next days
Step 3: Execution	teacher executes his plan	buildings send the planned units into battle
Step 4: Evaluation	teacher evaluates whether the problems have been solved or not	player gets feedback about the result of his defense

First, some problem has to be identified. In the special needs school, this is done by the teacher: Whenever he discovers some problems with one of his students, he has to do something about it. In our game, this is implemented as the arrival of a new army: The player knows he has to plan some kind of defense in order to defeat it. Following this initial analysis, some adaptions have to be planned: The teacher has to prepare some kind of support for the child, and the player has to plan his defense strategy. Third, the plans are executed, followed by the evaluation: In the special needs school, the teacher has to find out whether his plan has worked and his student's problems are solved or not. In the game setting, the player immediately sees whether the enemy has been defeated or not.

7.5.3 Architecture and Data Generation

In this section, we will discuss the general architecture, which log files are created, and how they can be accessed via the given architecture.

7.5.4 Generated Log Files

While players play the game, three types of log files are generated and updated for each village: The general log, the change log and the execution log. In this section we describe these log files in detail.

In the **general log** all information, ranging from the arrival of new enemies, over which defense units have been added, up to magical alignment shifts are shown. This log file sums up the development of a certain individual - on the one hand contingent on the decisions the player has made, on the other hand contingent on internal factors, on which the player has no influence (such as new enemy armies which arrive, population lost etc.). In our nursing example, this log file would contain some data about the development of the patient, maybe including data like his body temperature, weight, blood sugar level, or other parameters which are relevant for his treatment. Listing 7.2 shows an example general log of the village:

Listing 7.2: General log example

```
<?xml version="1.0"?>
<log>
  <event id="1" type="started" day="" text="Village founded on day "/>
  <event id="2" type="newday" day="2" text="New day: 2"/>
  <event id="3" type="incoming" army="3" day="2" text="Army 3 incoming on day 2"/>
  <event id="4" type="alignmentshift" day="2" alignment="heat" shift="10" newvalue="70"/>
  <event id="5" type="defense" day="2" defensetype="weapon" defensename="Longbow" building="
       forge"/>
  <event id="6" type="defense" day="2" defensetype="weapon" defensename="Round Shield" building
       ="forge"/>
  <event id="7" type="defense" day="2" defensetype="weapon" defensename="One Handed Sword"
       building="forge"/>
  <event id="8" type="defense" day="2" defensetype="unit" defensename="Knight" building="
       barracks"/>
</log>
```

The **change log** shows the process change operations the player has conducted while planning his defenses. These change operations represent the next steps which will be taken in order to save the village from the threat. Coming back to our nursing example, this log file represents the planned therapy steps which will be taken in the future in order to support the patient in dealing with a certain condition. Listing 7.3 shows an example for a change log from the game as it has been logged by the logging service. Here, the change shows the insertion of a new defense strategy for a specific village.

Listing 7.3: Change log example

```
<?xml version="1.0"?>
<log xes.version="2.0" xes.features="arbitrary-depth">
  <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
  <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
```

```
  <extension name="Change" prefix="change" uri="http://leonardo.wst.univie.ac.at/acaplan/change
       .xesext"/>
  <global scope="trace">
    <string key="concept:name" value=""/>
  </global>
  <classifier name="Change" keys="change:type change:subject"/>
  <trace>
    <string key="concept:name" value="Change Log of Village 1"/>
    <event>
      <date key="time:timestamp" value="2016-05-11 17:36:09 +0200"/>
      <date key="day" value="2"/>
      <int key="change:transaction" value="0"/>
      <string key="change:type" value="insert"/>
      <string key="change:position" value="root"/>
      <string key="change:fragment" value="17"/>
      <string key="change:rationale" value="troll scouts"/>
      <string key="change:goal" value="defeat army"/>
    </event>
  </trace>
</log>
```

The **execution log** sums up the process steps which have been executed in order to help the village to overcome the enemy army. These steps have been planned beforehand (as can be seen in the change log). In the nursing domain, there has to exist an execution log for each patient due to legal obligations: For each patient, each therapy step which has been executed has to be strictly logged in order to trace back any errors which have may been done if something goes wrong. Listing 7.4 depicts an execution log that reflects how the units are added to the villages defense.

Listing 7.4: Execution log example

```
<?xml version="1.0"?>
<log xes.version="2.0" xes.features="arbitrary-depth">
  <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
  <trace>
    <event>
      <int key="day" value="1"/>
      <string key="concept:name" value="archer trained"/>
      <string key="unit" value="archer"/>
      <string key="building" value="barracks"/>
    </event>
    <event>
      <int key="day" value="1"/>
      <string key="concept:name" value="knight trained"/>
      <string key="unit" value="knight"/>
      <string key="building" value="barracks"/>
    </event>
    <event>
      <int key="day" value="1"/>
      <string key="concept:name" value="fireball created"/>
      <string key="spell" value="fireball"/>
      <string key="building" value="alchemylab"/>
    </event>
  </trace>
</log>
```

These logs provide valuable data sources for deriving insights on the effects of previously applied changes based on process analysis techniques. In this section we have shown some parallels between our log files and the nursing domain. In the next section, we will evaluate the log files with equivalent data from the software sales and support domain.

7.5.5 Service Based Architecture

The game architecture shown in Fig. 7.19 consists of several independent services which interact via RESTful interfaces. It consists of a *game server*, which provides the calculations for the fights and when which enemy army shows up where, a *process engine* which manages the defense process instances for the villages, a *logging service* which logs all information into the *data repositories* and a *game interface*. In the following, we will describe each service in detail.

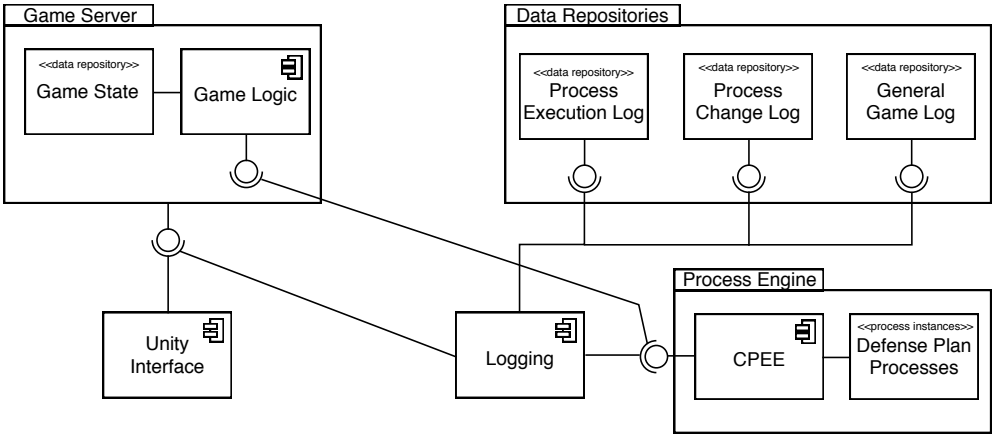


Figure 7.19: Service Components and their Interactions

The Cloud Process Execution Engine (CPEE) [64] is a service-oriented process execution engine which itself is a RESTful web service. It manages the defense process instances and provides multiple interfaces for interacting with the process instance and logging various kinds of information.

The game server provides all calculations for the fights and the villages and stores all relevant information for the games which do not directly belong to the defense process instances. This includes among others the population, magical alignment and topography of the villages. It also communicates with the CPEE when the player updates the defense plan of a village and controls the enemy: Each round it calculates which villages of a player will be attacked by which enemy, and calculates the damage these enemies do.

The logging service generates the change and execution logs[3] for the defense plan processes which are provided by the CPEE. Additionally, it logs relevant information from the game server, which can be used to further understand and evaluate these change and execution logs. For example, the logging service logs when which enemy arrives at a certain village, how many units were lost during the fights, how the magical alignment shifts etc. In combination with the change and

[3]Log format: XES, cf. `http://www.xes-standard.org/`

execution logs, the logging service provides a complete picture of each situation, which then can be used for further studies.

The user interface has been developed in Unity. It accesses the data and services provided by the game server web service over RESTful interfaces.

7.6 Evaluation

The goal of the game-based setting is to provide log data that are not subject to any disclosure restrictions and a test setting which can be used to develop algorithms and approaches which support users when adapting process instances in a FIPS. The different types of log files, namely change, execution and general logs, have been introduced in the last section. In this section we want to show that the information gathered in these log files are actually comparable to the set of information which is gathered in a real world FIPS. We claim that if the data generated in our game matches the data generated during an individual situation in a real world FIPS on a conceptual level, our data can be used to evaluate approaches and algorithms to support process adaptions in FIPS.

For our evaluation we analyzed support cases from mesonic[4], the ERP / CRM software developer introduced in the expert interviews in Section 7.2.3. In their *support network*, mesonic has over 200.000 different support cases describing the respective problem in free text form. Each support case is classified in one of four groups, ranging from bugs, where the development team has some work to do, over wishes, where it has yet to be decided whether the desired feature will be implemented or not, over general support cases up to questions which are already answered in the manual. Depending on the case's classification, the next steps are decided: If it is a real bug or a wish, it will be discussed with the responsible developer or, if necessary, the whole team. If it is a general support case, it will be handled in the support department internally. If it is something which can also be found in the manual, the responsible supporter handles the case on his own. This classification represents the first steps in the planning stage: Based on a cases "symptoms", the next steps are planned. In addition to this classification, information such as the customer's name and id, the retailers name and id, the version and build number of the product are logged.

Figure 7.20 summarizes the comparison of the data elements in the real world FIPS with the data elements in the game: The top section shows an example of such a support case: Here, some problems with the OLAP component have appeared which have been classified as a bug. On the bottom, example log files

[4]Due to data privacy issues we only publish anonymized parts of the actual data gathered from these support cases in this thesis. Since the original support cases are in German free text form, we shortened them and translated the relevant parts.

from the game are shown. The mappings between the data elements are indicated by the arrows. The general data of the support case (i.e., customer id, retailer id etc.) is not part of any log file, but static information which is saved separately.

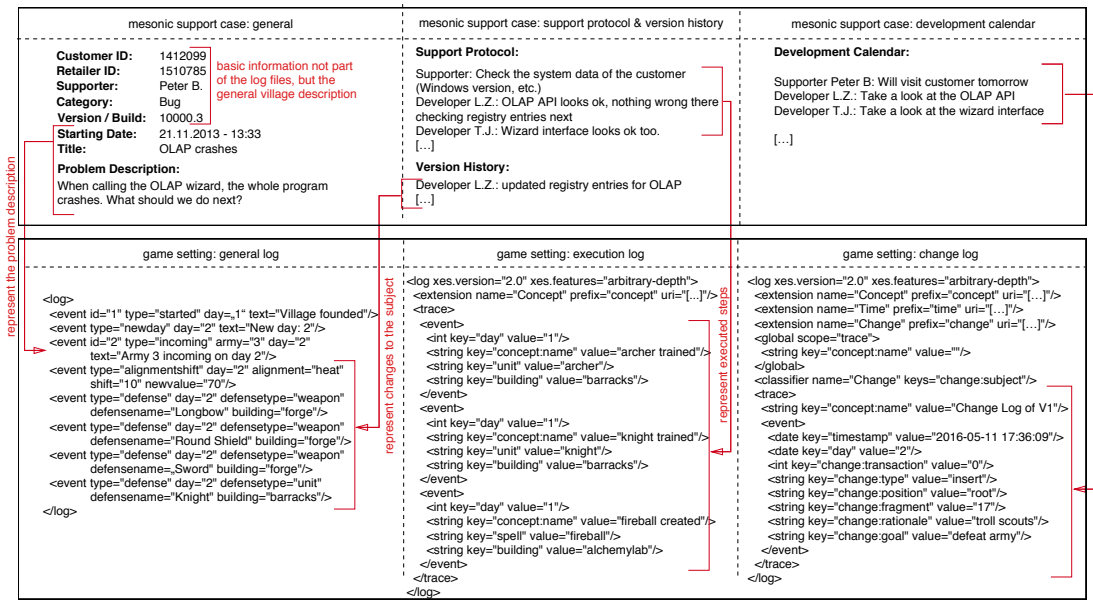


Figure 7.20: Mapping of the log files to a real world FIPS scenario

Data Source Comparison 1: The change log: At mesonic, next steps are planned with the responsible employees and shown in their calendars. In the game, the change log shows the planned steps for solving a village's problem (i.e., the incoming enemy army).

After the next steps for handling the case are planned, they are executed by the responsible employees. The executed steps are documented in the support case itself in free text format.

Data Source Comparison 2: The execution log: Mesonic's support case documents what has been done, who has done it, and when it has been done in order to solve the problem at hand. The execution log from our game represents the very same type of information: It is shown when which step has been executed in order to deal with the village's problem. This information includes the time, what has been done, and which role has executed the steps.

All actions influence the state of the product. These state changes are documented in the support protocol as well as the company's version control system.

Data Source Comparison 3: The general log: The version history of mesonic logs all information about changes to the product's source code. These changes may be because of actions which have been applied because of planned steps within the process, or factors not related to the support case. In the game, the general log summarizes all state changes of a village's parameters. These changes happen because of the player's actions, or because of other events which are not influenced directly by the player (but e.g., by the enemy army).

Conclusion: The comparison between the three generated log files from the game and the data gathered in a real world FIPS, namely the software sales and support setting, shows that all relevant information to a FIPS can be found in the game logs. In the real world setting, however, many data sets are free text, thus making further analysis difficult. Here we see a big potential of the game: The structured data in XES respectively XML can be easily used for further analysis, while representing all information relevant to a real world scenario.

7.7 Discussion

FIPS can be found in many different domains, ranging from the nursing sector, over hotel guest and software customer interaction to special needs schools and the PhD program. Due to the high degree of process flexibility, they pose the killer application for the concepts presented in this thesis. Since all of these domains deal with highly sensitive data, it is almost impossible to get a full picture of a situation where such a process instance has to be adapted without running into data privacy and legal issues. For this reason we have developed a tower defense game where the players are put into a comparable situation: They adapt highly individual process instances in order to deal with problematic situations. The data thus generated can be used without any issues, may they be legal or otherwise. Additionally, the setting itself can be used as an evaluation method for support methodologies: A set of players who received support while adapting their process instances can be compared with a set of players who did not receive any support. If the supported group of players performs way better, it may be an indicator that the support is actually helpful. In the future we will use the game in order to develop and enhance support and analytical capabilities for FIPS. This includes analyzing change logs, finding out when the desired goals are reached, and, ultimately, finding the best adaptation for a given situation.

In the next chapter, we will present two user-based experiments for the approaches developed in this thesis: While the first experiment focuses solely on the understandability of change trees and aggregated change trees, the second experiment is set in the context of the game discussed in this chapter: Players who play

the game will receive information about previously conducted change operations based on the concepts developed in this thesis.

Chapter 8

User Experiments

In the last chapters, we have presented the technical feasibility of the concepts developed in this thesis, and their applicability for various application domains in realistic scenarios. In the last chapter, we have set a special focus on *flexible and individual process settings* as the killer application for the application of process change operations. Besides the technical feasibility, and the applicability, the *usability* of the presented techniques is a major part which we want to address as part of our evaluation (cf. Figure 8.1). Hence, in this chapter, we present two user experiments which show that the concepts developed throughout this thesis can be used by users to improve their efficiency and effectiveness when working with process change operations. This chapter is based upon the paper *"Improving the Usability of Process Change Trees based on Change Similarity Measures"* [50].

8.1 Overview

The representations and analysis methods presented in this thesis provide different methods of insight both into single process change operations, as well as change traces and change logs. The goal of these methods is to provide support for users who have to analyze information about previously conducted process change operations contained in a process change log, or who have to decide on a new change operation. In previous chapters, we have shown a prototypical implementation (thus providing proof of technical feasibility), as well as how these approaches can be applied to realistic settings, thus discussing their applicability. To show that users actually profit from the representation and analysis methods developed in this thesis, in this chapter we provide proof of usability: We will show that the approaches developed in this thesis can be used and understood by users who work with process adaptations. As discussed in [86], usability is a term which comprises multiple aspects of the interaction between users and systems. Depending on

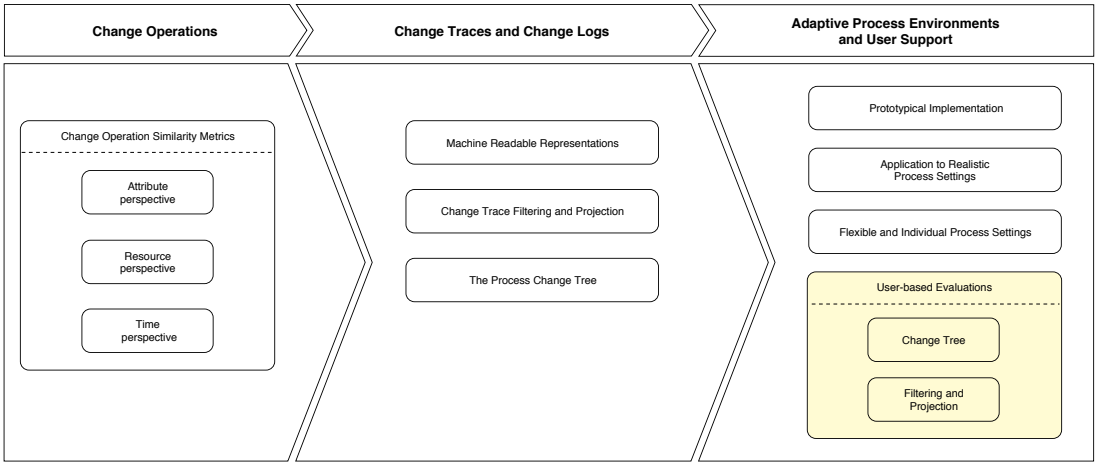


Figure 8.1: Dissertation Overview: User Experiments

the definition, different aspects of usability are defined [86]. In [87], usability is comprised of the five aspects *efficiency*, *learnability*, *memorability*, *errors/safety*, and *satisfaction*. In our usability evaluation, we set our focus on two aspects which we considered to be most important. When analyzing process change logs, we argue that efficiency and minimizing errors of users who analyze these logs are very important aspects: On the one hand, efficiency describes how many resources are required when interacting with a certain system: When a domain expert adapts a process instance, the amount of resources (e.g., the time) required to analyze a change log should be kept to a minimum. For example, when a nurse in a nursing home changes the therapy plan of a patient, the time she needs to analyze previous changes should be kept to a minimum. Hence, the efficiency is a central aspect when analyzing process change operations. On the other hand, the number of errors users who analyze change operations make based on the presented data should be kept to a minimum as well: When a domain expert decides for a new change operation based on historic data contained in a change log, the correct data should be presented to him, and he should make the right decisions based on such data. For the nursing home example, historic data which is used to analyze what has been done in the past to solve a certain problem should match the given problem. For these reasons, we focused on these two aspects of usability in our experiment design.

In Chapter 4 we have presented the change tree, which can be used as a representation method for process change logs which highlights the chronological order of applied change operations. By aggregating nodes in a change tree by means of process change similarity (Chapter 2), the number of nodes in a change tree can again be reduced, thus allowing users to focus on the information they require.

The first part of the usability analysis presented in this chapter tackles exactly this assumption: We analyze, whether users who interact with the (aggregated) change tree outperform users who directly interact with process change logs.

Additionally, in Chapter 3, we have presented filtering and projection techniques for change traces: Here, the idea is to reduce a change log to a set of change traces, such that all change operations in a trace belong to the same case instance, and the remaining change traces evaluate to true for some filtering condition. This filtering condition can refer to some data about the process instances (e.g., the gender of a patient in a nursing home), or it could refer to some effect the change operation may have triggered (e.g., the time it took the patient to feel better again). The second part of the usability evaluation tackles the usability of these projected and filtered change traces: Given a certain problem which can be solved with a set of change operations, users who have to choose such a set of process change operations can immediately see the change traces which have led to the desired effects in the past (based on the filtering condition), and use this information as a basis for their decision. However, users who decide on such process adaptations do not necessarily have experience with process analysis techniques. Think about the nurse in the nursing home, or the teacher in the special needs school: Since technical aspects of process change operations are not part of their professional training, they may feel overwhelmed in learning handling new tools which aim at such techniques, such as the prototypical implementation *APES* presented in Chapter 5. Hence, in these settings, an integration of the information provided by the analysis techniques into a familiar user interface may help the person to actually use the provided information.

Overall, this chapter addresses the following research questions:

Q1: Does the change tree improve efficiency of users when analyzing process change traces? Which additional benefits can be gained by aggregating change tree nodes by incorporating process change operation similarity measures?

Q2: Can filtering and analysis techniques of change traces be used as a basis for presenting useful adaptations? Can those techniques be integrated into user interfaces?

These research questions are tackled in two experiments with users: In the first experiment, users work directly with change logs, change trees, and aggregated change trees. Thus, we can show that the change tree as presented in this thesis is a data structure which can be used on its own to represent change log information more clearly than the basic process change logs, and we show that change similarity measures can be used to increase its usability even further ($\rightarrow$ Q1). In the second

experiment, relevant data is filtered and projected from a process change log, and integrated seamlessly into a user interface. Based on this user interface, users had to interact with a prototype and make decisions with and without information from filtered and projected change logs. Thus, we show that by applying filtering and projection techniques, information contained in a process change log can be used to support users based on historic data when changing a process instance ($\rightarrow$ Q2). The two experiments also show that the presented techniques can be used both for users who work directly with process change logs, as well as for users who may prefer integration of the provided information into familiar user interfaces. Overall, the presented experiments incorporate the representation and analysis techniques for process change operations and traces presented in this thesis: Change trees and change operation similarity metrics are tackled by the first experiment; filtering and projection techniques of change traces by the second. Additionally, the prototypes used for the experiments are built upon the XES based representation of change logs which have been presented in Chapter 3.

The rest of this chapter is structured as follows: Section 8.2 presents the experiment for the understandability of change trees as a data structure. This is followed by the experiment which targets filtering and projection techniques, and integrates the resulting change traces into a user interface, which is discussed in Section 8.3. Section 8.4 concludes the chapter.

8.2 User Experiment: Change Trees

Process change trees are a data structure which aim at highlighting the chronological order of conducted change operations. This stands in contrast to existing change log representation mechanisms such as change processes [9], which focus on causal dependencies between conducted change operations, as shown in Section 4.3. Since change trees and change processes target different analysis questions, these representations have not been compared in a user based experiments. The advantage of using change trees in comparison to change logs represented in XES however, is the fact that change trees aggregate similar or equal process change operations which are on the same level. The idea is that based on such an aggregation users should retrieve the required information faster than when they are inspecting change logs directly. Thus, the first experiment tackles the usability of change trees as a data structure. This experiment aims at two goals, i.e., to analyze whether (a) the change tree can be used as a basis for analyzing change traces and (b) certain questions can be answered faster by participants using an aggregated change tree, than using a change tree.

8.2.1 Experiment Goals and Hypothesis

The main goal of the experiment was to analyze whether the (aggregated) change tree provides a better representation for change traces than their representation as pure XES log files. This leads to the following hypotheses:

Null Hypothesis Change Tree: Users who do use the change tree do not find the relevant information faster than users who use XES representation of change logs.
Alternative Hypothesis Change Tree: Users who do use the change tree find the relevant information faster than users who use XES representation of change logs.

The aggregated change tree (ACT) reduces the number of nodes in comparison to the change tree. Thus, for analyzing the usability of the aggregated change tree, the following hypotheses have been constructed:

Null Hypothesis ACT: Users who do use the aggregated change tree do not find the relevant information faster than users who use change tree.
Alternative Hypothesis ACT: Users who do use the aggregated change tree find the relevant information faster than users who use change tree.

8.2.2 Method and Experiment Design

The first draft of the experiment was refined using a two stage pretest: During stage one, the questions of the experiment were discussed with three peers who are familiar with the concept of process change operations and change trees. In the second phase, persons from the target audience were asked to participate in the experiment. The target audience of the aggregated change tree are persons from different fields of work, who do not necessarily have to be familiar with process change operations. During the pretest, two major parts of the experiment were optimized: First, the user interface was optimized to be more intuitive to users who are not familiar with the setting, second, the wording of two questions and their explanation was optimized.

The experiment consists of two parts: Part 1 addresses the question whether the change tree (CT) is a better representation of change logs than their raw XES format (XES). Specifically, we want to find out if certain questions regarding the change log (e.g. *How many resources have been used in this change log?*) can be answered faster using a change tree than an XES based log file. The participants were split in two groups and had to answer the same questions based on those two representations. The questions can be found in Appendix A. In each question, we measured the time until the (correct) answer was found. The questions for

the two representations were based on different, but similar log files to prevent bias due to the participants knowing the correct answer: While group 1 answered questions for the CT based on log A, and questions for XES based on log B, group 2 received the CT questions based on log B, and the XES questions based on log A. This setup also helped us to mitigate the risk of a bias due to different log files: If there was only one group who answered all CT questions based on log A, and all XES questions based on log B, the differences in speed and correctness could be due to the slight differences between those log files, and not due to their representation. To mitigate the risk of distortion due to a learning effect, the order of the questions and representations was randomized. Part 2 of the experiment addresses the usability of the aggregated change tree (ACT) in comparison to the change tree. The setup and the participants were the same as in Part 1; however, two log files C and D (different from those of Part 1) were used[1].

At the beginning of the experiment, the participants were introduced into the topics of processes, process adaptations, change logs and (aggregated) change trees. Following this 10 minute presentation, the participants had the chance to ask questions if anything was unclear, and received a link to the experiment. They had three days to participate in the experiment. We have decided for such an *uncontrolled experiment setup*, since it resembles a realistic setting, in which such aggregated change trees would be used, more closely.

8.2.3 Questions and Interface Design

The experiment is based upon the APES prototype, the prototypical implementation of the (aggregated) change tree discussed in detail in Chapter 5. Since the APES server works with any user interface, it was very easy to adapt it to the requirements of this experiment: Instead of using the standard APES user interface (presented in Section 5.2.3), we have created a user interface tailored to the specific needs of this experiment.

Before the experiment started, the participants were shown an introductory web page containing the slides from the 10 minute presentation. Additionally, they were given the possibility to contact the author of the experiment via email, if any questions remained unsolved. This interface is shown in Figure 8.2.

Following this introduction, the participants received eight questions which targeted the change log in XES, the change tree as well as the aggregated change tree. A detailed summary of all questions, and how they were split up among the two groups is shown in the appendix in Chapter 10.3.

Figure 8.3 shows how such a question has been presented to the user: In order

[1]The log files can be found on our project homepage `http://cs.univie.ac.at/project/apes`

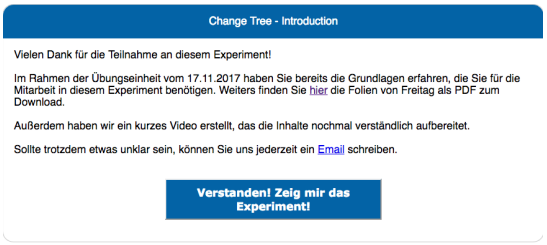


Figure 8.2: Introduction into the experiment on (aggregated) change trees

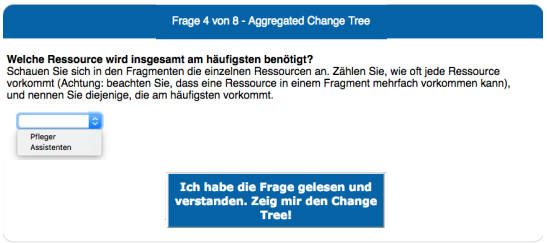


Figure 8.3: One question from the change tree experiment

to increase the comparability of the results only the question and the possible answers were shown to the users in the beginning. Only after pressing a button, the respective representation (either the (aggregated) change tree, or change log in XES) was shown. Once the representation was shown to the user, we started measuring the time until an answer was given. By removing the time the users needed to read and understand the question, we again increased the comparability of our results.

8.2.4 Results

In total, 64 participants participated in the experiment. These participants were divided in two groups as described above. 31 participated in group 1, and 33 participated in group 2. First, the data set was cleaned for participants who obviously entered random numbers. The check was based on two parameters: (1) most of the answers were wrong, and (2) all answers were given in less than two seconds. Answers given that quickly can be seen as an indicator for a participant entering random answers on purpose. These patterns were found for one participant from group 1, resulting in 30 valid participants in group 1, and 33 valid participants in group 2.

Table 8.2 summarizes the results from the experiment. The mean and median times refer to the amount of time needed by the participants to find the correct answers. Using a two sample t test, the differences regarding the amount of correct answers given are not significant ($p > 0.05$): the correctness of the answers was not negatively influenced when using CT compared to the XES log and ACT compared to CT respectively. However, the time required to find the correct answer is statistically significant for both parts of the experiment ($p < 0.05$): In the first part of the experiment, it can be seen that the mean and median times required to find the correct answer for the CT is much lower than for the XES; in the second part, it can be seen that participants using the ACT require even less time to find the correct answer than participants who use the CT, thus improving

Table 8.1: Results of the experiment

XES Log vs CT	**Overall**		**Group 1**		**Group 2**	
	Log	CT	Log	CT	Log	CT
Correct Answers	61.11%	68.25%	61.67%	68.33%	60.61%	68.18%
Mean Time (sec.)	90.02	35.43	95.28	33.12	83.37	39.97
Median Time (sec.)	58.69	29.2	57.51	27.32	59.43	32.14
CT vs ACT	**Overall**		**Group 1**		**Group 2**	
	CT	ACT	CT	ACT	CT	ACT
Correct Percentage	65.08%	67.46%	71.67%	68.33%	59.09%	66.67%
Mean Time (sec.)	112.54	43.97	138.08	52.84	84.39	35.71
Median Time (sec.)	51.73	22.43	61.82	24.18	48.28	21.4

their efficiency regarding the required time.

8.2.5 Discussion and Threats to Validity

The experiment shows that the (A)CT can be used as a representation of change logs, since the amount of correct answers is not statistically different from the amount in the XES log. This effect may also be due to the XES log being very simple: For Part 1 of our first experiment we chose a log with only four change operations in total, in order to not demoralize participants who had to work with the XES representations. Thus, the change log was designed in favor of the XES representation, which may pose a threat to the validity. However, even in such a setting, the time required for the correct answers was statistically significantly lower for the change tree than for the XES log. For the aggregated change tree, the number of correct answers is not statistically significantly different compared to the change tree, but significantly less time is required to find a correct answer. Although the uncontrolled setup of our experiment increases the resemblance of our experiment to a realistic setting in contrast to a controlled setup, this decision also poses a threat to validity, since participants have been able to communicate with each other during the execution of the test without our knowledge. However, participants who completed the experiment did not receive the correct answers to the questions, and such behavior may also be found in a realistic setting, where users would have to analyze change trees. Thus, although this threat exists, the results can be seen as being more realistic, and the impact of the threat has been kept to a minimum. Overall, we can conclude that the (aggregated) change tree can be used as a representation for change logs, when the time required to analyze a change log should be kept to a minimum.

8.3 User Experiment: Filtering and Projection

In our second experiment, we analyzed whether the filtering and projection techniques presented in Chapter 3 can be used as a basis for presenting useful adaptations to the user. The idea is that based on historic data in form of change logs which contain information about the adaptations in the past which have been conducted to solve a certain problem (projection step), those change traces are filtered, where a desired effect has been reached (filtering step). Thus, by presenting the user change operations which have led to a desired effect in the past, we want to reduce the amount of errors she commits when adapting process instances. Such situations can be found in different domains, e.g., in the nursing domain, where the nurse is looking for possible changes to the patient's therapy plan, which help him to cure a fever (projection) as fast as possible (filtering). Since personnel working in such settings usually are not trained in the technical aspects of process change operations, an integration of the resulting change operations to a familiar user interface might enhance the usability of the results. For this experiment, we have created a setting which adheres to this problem description for *Apelands*, the game-based experimentation setting for flexible and individual process settings described in Chapter 7.

As described in Chapter 7, in this game, the player has to face different computer-controlled armies, which attack one of his villages. Depending on the incoming enemy, different adaptations to the village's defense process instance have to be carried out. If the defense activities these change operations add to or remove from the process instance perform well against the enemy army, this army's attack will last fewer rounds. If the player chooses defense activities which are not successful, the enemy army will stay for more rounds, and ultimately defeat the player in its last round. For the enemy we have chosen for this experiment, a total of five rounds exists. Depending on the player's choices, the enemy will attack the player's village for two, three, four or five rounds, whereby it will destroy the player in the last round by attacking with a special *hero*, which leads to the player losing the game. Based on this setup, the projection and filtering techniques for change traces as described in Chapter 3 can be used: The village's defense process instance's change log contains all change events which have been applied independent of the attacking computer-controlled enemy. When projecting these change traces to the enemy described above, the projected change traces contain all change operations which have been applied for this specific enemy. Thus, for each time an enemy attacks a village, there exists exactly one corresponding projected change trace which contains the adaptations the player has made. Next, we want to filter these projected change traces for those changes where the enemy army has attacked for two, three, four or five days. As described in Chapter 7, a day in the game is equal to a round. Those change traces which remain with

the filtering condition `days == 2`, i.e., the best possible result for the player, are those which have led to the player defeating the computer-controlled enemy army with the fewest required game rounds. In Apelands, similar change traces applied to the same problem often lead to similar results. Hence, we argue that when a player receives information about previously executed change traces where this filtering condition has been met, and he implements the same change operations to his process instance, the result should be similarly positive.

The subjects who participated in the experiment played the game in two sessions; first without, later with information from the data retrieved from these projected and filtered change traces. For the duration of the experiment, the participants received a version of the game in which they only played against the one computer-controlled enemy described above. In the first session, they received only the information players usually receive about an enemy who attacks a village, i.e., general information such as its name, and hints at which kind of defense may work, although, no historic data from change logs is presented here. In the second session, the players received information about historic data stemming from the process change logs, which has been preprocessed using the filtering and projection techniques as described above: players were shown a new user interface as part of the game's user interfaces which showed the change traces meeting the desired filtering condition `days == 2`. This experiment setup serves two purposes: First, we analyze whether projection and filtering techniques as presented in this thesis can be used as a basis for presenting adaptations which lead to the desired effects; second, we want to test whether such data can be integrated seamlessly into a user interface, thus allowing users to analyze the information directly in their familiar environment.

The rest of this section is structured as follows: Section 8.3.1 discusses the hypotheses and the applied statistical methods, Section 8.3.2 describes the setup of the experiment, and 8.3.4 discusses the results.

8.3.1 Experiment Goals and Hypothesis

We expect persons who play the game while receiving historic information encoded in the projected and filtered change traces to reach better results than players who do not receive such information. This leads to the following hypotheses:

Null Hypothesis: If the users follow the adaptation suggestions based on information from the filtered and projected change traces, their decisions do not lead to better results than their decisions without such information.

Alternative Hypothesis: If the users follow the adaptation suggestions based on information from the filtered and projected change traces, their decisions do lead to better results than their decisions without such information.

8.3.2 Method and Experiment Design

The experiment was conducted with 22 business informatics students at the University of Vienna. Although the participants have technical experience, they had no prior knowledge on the particular research of this work, i.e., change log filtering and projection techniques. After an introduction to the game and its scientific background, the participants were shown how the game interface works, where they can find the information required to decide what a good adaptation may be, and how they can adapt process instances.

This was followed by two sessions with approximately 15 minutes each, in which the participants played the game. For comparability, each participant had the exact same setup, as described above: Each participant had one village to defend against one incoming enemy army of the same type. In the first session, the participants had to play *Apelands* without support. They could come up with their own solutions on how a process instance should be adapted in order to efficiently counter this incoming enemy army. In the second round, the players were facing the same enemy, but this time they received support from the filtered and projected process change logs.

The projection of the change log returned a set of change traces, where each trace contained all actions the players had done in order to defend against an enemy of this type. For this enemy, the best result is to defeat the enemy army after two days. Hence, the filtering condition was set to `days == 2`.

For the enemy army, which can be seen as a case (i.e., a problem description for the process instance which has to be solved by means of process change operations), four different filtering conditions evaluate how well the players did counter the enemy. To increase readability of the plots in this evaluation, we assign a label to each filtering condition. For players who found an optimal solution against the enemy, the trace containing the player's actions will be filtered by the filtering condition `days == 2` (label: *"2 days"*). If it took the players 3 (4) days (rounds in the game) to remove the enemy from the village, the filtering condition `days == 3 (4)` (label: *"3 days"* or *"4 days"*). If the player did not defeat the enemy army after four days, he loses the game, and his actions will be filtered by the filtering condition `days == 5` (label: *"hero"*).

8.3.3 User Interface Design

For the evaluation of the filtering and projection techniques, *Apelands* has been extended by an additional user interface: The support interface (cf. Figure 8.4) shows one projected change trace which fulfills the filtering condition. It tells the user to add two *Dark Iron Golems*, one with the spell *Fireball*, the other with the spell *Feral Wrath*. As can be seen, this user interface abstracts from the technical

Figure 8.4: Support interface showing a good defense against the current threat

details of process change operations. Instead, it highlights the information which is important to the user, i.e., the defense activities he should add to the process instance of the village.

8.3.4 Results

In total, the participants finished 77 games, from which 47 have been played without support, and 30 with support. A game is seen as finished, if one of the four filtering conditions described above evaluates to true for the corresponding process execution log; i.e., the enemy has been defeated after two, three or four days, or the enemy hero has appeared in the fifth round of the game. Due to the fact that the participants were allowed to quit the game client without finishing the game, of the 47 unsupported games, 36 were completed such that the corresponding execution log contained the information required for one of the filtering conditions, and thus can be used for evaluation. All process settings where none of the filtering conditions evaluate to true cannot be used for evaluation of the filtering and projection techniques, since the required information is missing. Of the 30 games which have been finished using the filtered and projected change trace, for 25 the necessary information can be found in the corresponding execution traces. Following, from the 77 finished games in total, 61 can be used as a basis to evaluate the usefulness of projected and filtered change traces in a flexible and individual process setting.

As described above, when filtering the projected change traces for the case of this experiment, one of four different filtering conditions will be met. These filtering conditions describe how well the player has defended his village against the incoming enemy army. As described above, these filtering conditions are `days == 2`, `days == 3`, `pdays == 4`, and `days == 5`, describing the results of the case instances in descending order. The game is designed such that for this set of filtering conditions, for each case instance, exactly one of the filtering conditions can be reached. Thus, when, e.g., the filtering condition `days == 2` is met, it is guaranteed that other filtering conditions will not be met.

Without any information from the projected and filtered change traces (n=36), 1 game has been filtered using the filtering condition `days == 2`, (*2 Days*, 2.78%), 11 have met the second filtering condition `days == 3` (*3 Days*, 30.56%), 19 met the third filtering condition `days == 4` (*4 Days*, 52,78%), and 5 have met the worst possible filtering condition `days == 5` (*Hero*, 13,89%).

While receiving additional information by the projected and filtered change traces (n=25), 17 met the best filtering condition *2 Days* (68%), 7 met the second filtering condition *3 Days* (28%), no one has met the third filtering condition *4 Days* (0 %), and one participant still met the worst possible filtering condition *Hero* (4%).

Figure 8.5 summarizes these results: The blue bar shows the number of games which have been finished while the players received information from the projected and filtered change traces, and the green bar shows the number of games which have been finished without such support. As can be seen from the diagram, participants who received support tend to perform better than those who did not receive any support.

We applied a Wilcoxon Signed Rank Test to the experiment, since we are dealing with paired samples of data (for each player, we have results with and without filtering and projection support), and the data is not normally distributed. Each participant was able to play multiple rounds of the game, while (not) receiving information from the filtered change traces. This is due to the fact that we limited the time the participants had for playing the game, instead of the number of rounds. Following, for each participant multiple results exist, which are summarized in Table 8.2. The resulting ranking is represented by the ranking of the filtering condition reached by the participant, i.e., when a case instance can be filtered using the best filter `days == 2`, a 1 can be found in the table. The second best filtering condition `days == 3` is represented with a 2, the third best condition `days == 4` with a 3, and the worst filtering condition `days == 5` with a 4.

Based on the data gathered, we have created four measures relevant for further inspection: (a) the best filtering condition can be met with and without support, (b) the worst filtering condition can be met, (c) the mean and (d) the median of

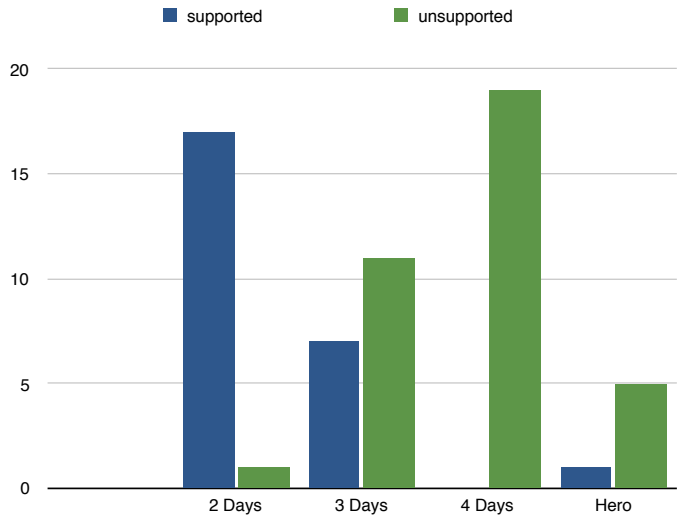


Figure 8.5: Summary of the reached goals

Table 8.2: Results of the experiment

Player	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Mean S.	1.0	1.0	1.0	1.0	1.0	2.0	1.0	1.0	2.0	1.0	1.0	1.5	4.0	1.0	1.5	1.0	2.0	2.0	1.0	1.0	2.0	1.0
Mean N.	2.66	2.5	2.5	2.5	2.0	2.5	3.0	3.0	3.5	2.5	3.66	2.0	3.0	2.0	3.0	3.0	3.0	3.0	3.0	2.0	4.0	3.0
Median S.	1.0	1.0	1.0	1.0	1.0	2.0	1.0	1.0	2.0	1.0	1.0	1.5	4.0	1.0	1.5	1.0	2.0	2.0	1.0	1.0	2.0	1.0
Median N.	2.0	2.5	2.5	2.5	2.0	2.5	3.0	3.0	3.5	2.5	4.0	2.0	3.0	2.0	3.0	3.0	3.0	3.0	3.0	2.0	4.0	3.0
Best S.	1	1	1	1	1	2	1	1	2	1	1	1	4	1	1	1	2	2	1	1	2	1
Best N.	2	2	2	2	2	2	3	3	3	2	3	2	3	2	3	3	3	3	3	2	4	3
Worst S.	1	1	1	1	1	2	1	1	2	1	1	2	4	1	2	1	2	2	1	1	2	1
Worst N.	4	3	3	3	2	3	3	3	4	3	4	2	3	2	3	3	3	3	3	2	4	3
S. = Supported, N. = Not Supported																						

all filtering conditions met with and without support. For example, if a player has played five games without support, and met filtering condition `days == 2` (1) once, `days == 4` (3) three times, and `days == 5` (4) once, the best filtering condition would be `days == 2`, thus being represented by a 1 in the table. The median would be 3, since `days == 4` is the median, the mean would be 2.8, and the worst filtering condition `days == 5` would be represented by a 4.

For each of the four groups (best results, worst results, median results, mean results) the Wilcoxon Signed Rank Test was below the significance threshold of $p = 0.05$, thus the null hypothesis can be rejected. Subsequently, receiving information from the filtered and projected process change logs has a positive impact on the results of the players. Figure 8.6 visualizes the results in box plots: As can be seen, for each measure with support significantly more results are closer to 1 than without support, which represents the filtering condition which indicated that players have won the game the best way possible.

8.3.5 Discussion and Threats to Validity

Conclusion: The goal of the evaluation was to find out, whether filtered and projected process change logs can be used as a data basis to provide users with information for situations in which they have to adapt process instances. When receiving such information about change operations which have led to the desired effects before, the amount of errors made by the users responsible for the adaptation should be reduced. The results of our evaluation show the following: First, in the given setting, historic data can be used to provide users with the necessary information about which change operations might lead to the desired results when making adaptation decisions. Second, our representation of the suggestions provided by the filtering conditions has led to the fact that the users who are not used to work with filtered and projected process change logs did less mistakes when adapting their processes, thus, reaching better overall outcomes. Notably, users are not forced to follow the suggestions, i.e., they are free to decide for the given situation against the suggestion. For a real world process setting, this makes sense: In a medical setting, it could be fatal to force doctors to follow a predefined therapy adaptation, only because it has worked in other cases before.

Threats to validity: The experiment has been conducted with 22 participants who had no experience playing Apelands, and who had limited time to get used to the game's user interface. Thus, the results from the group of players who did not receive any information about which change operations might lead to the desired results is even more clearly below those of the group of users who did receive such information than it may have been if the participants had already known the process setting. However, this leads to the conclusion that such information provided by the filtered and projected process change logs can even be effective for users who have no experience in the process setting. Further on, the experiment was conducted with participants having a technical background; however, they were neither familiar with provided representation of process change logs, including the techniques for filtering and projecting nor with the process setting.

8.4 Discussion

In this chapter, we have presented two user experiments for evaluating the usability of the concepts presented in this thesis. As outlined in the introduction of this chapter, we have set our focus on the efficiency and the reduction of errors made by the users.

As we have shown in our first experiment, using the (aggregated) change tree as a basis to visualize process change logs in contrast to the change logs being

present in the raw XES format can lead to users answering questions about the basic process change logs faster. Hence, in this experiment, the efficiency of users who analyze process change logs has be enhanced.

Additionally, the filtering and projection techniques could be used as a basis for user interfaces which aim at reducing the amount of errors users make when adapting process instances: In our second experiment we have shown an example where resulting change traces of the presented filtering and projection techniques have been integrated to a user interface. As a result, users have decided for adaptations which have led to the desired outcomes in more situations, thus reducing the amount of wrong decisions made. In the experiment presented in this chapter, a simple visualization which only depicts the change operations which have been applied has been chosen. However, based on such a technique, more advanced visualizations can be created.

In the next chapter, we will discuss related work, followed by a chapter discussing future work, and concluding this thesis.

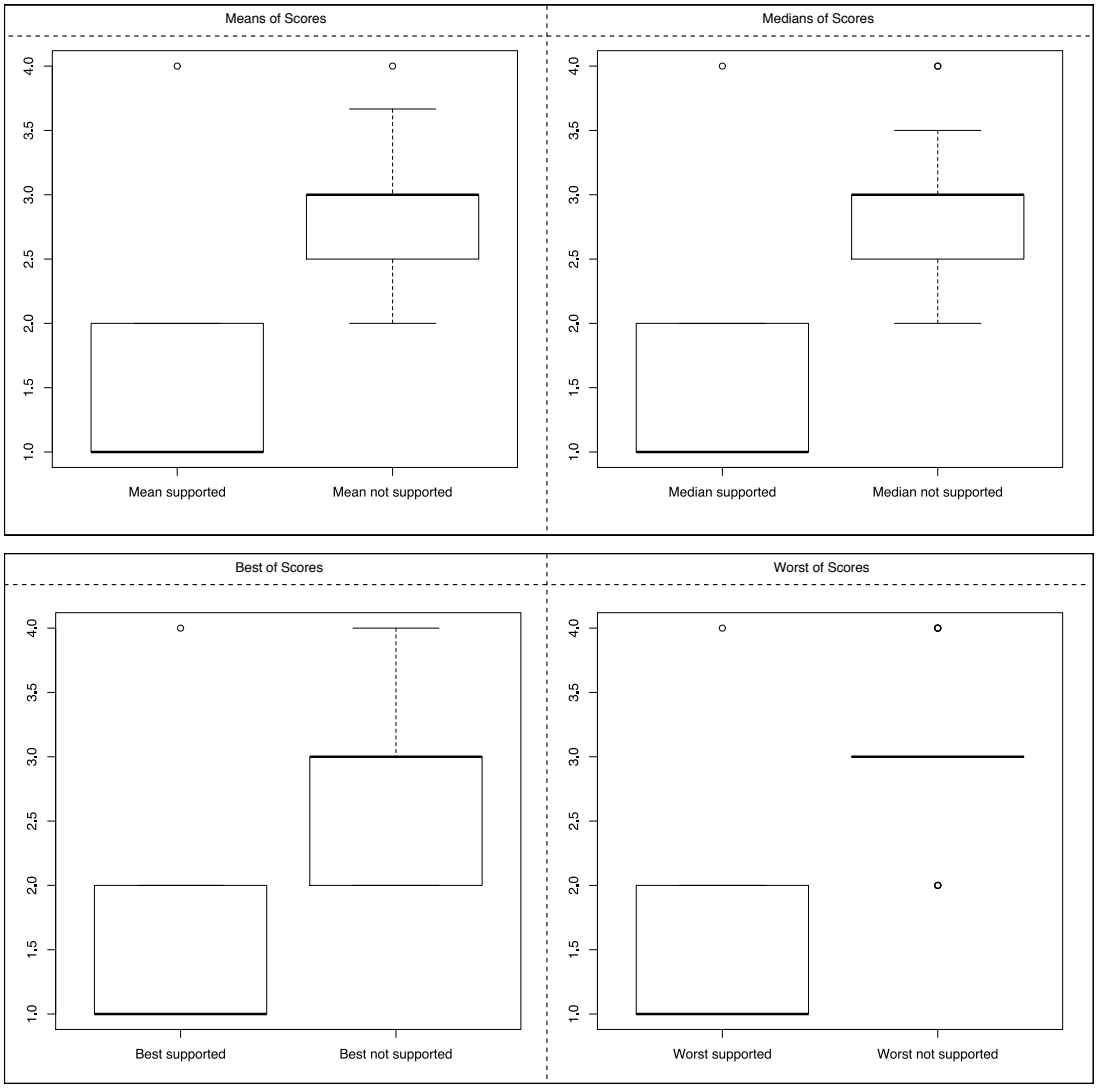


Figure 8.6: Results of the Evaluation

Chapter 9

Related Work

In this thesis, we have covered several topics relevant for representing and analyzing single process change operations, change traces, and their application in realistic settings. In this chapter, we provide an overview over research areas which we have considered as related work. These topics are summarized in Figure 9.1.

Process mining [88] has been a major topic of business process management for several years. Many mining algorithms focus on extracting process models from event logs [42, 43, 44, 89]. This can be useful for several topics, e.g., auditing organizations regarding their business processes [90], supporting health care [13], or for industrial applications [91]. Aside from mining execution logs from already finished instances, change mining techniques can also be applied to *running instances*, thus predicting upcoming events, or checking the conformance of executed activities [14]. Other approaches first cluster execution traces into subsets based on certain criteria, thus increasing understandability of the mined process models.

However, change mining has received attention as well: [92] presents a framework which determines the impact of a process change operation on other, possibly also affected models: Thus, if a change operation affects a process model, subsequently necessary changes can be analyzed as well. In [93], the authors propose a method to learn which changes are common in a change log. Based on the results, the underlying process model should be changed, such that for future instances, thus decreasing the necessity and costs of adaptations. The C3Pro project deals with a similar kind of change mining: Whenever a partner in a process choreography changes his process, this change has a potential impact on the private processes of his partners [22, 25, 23, 24, 94]. In addition to its effects on process choreographies, impacts of change operations on compliance rules have to be considered as well. This topic is studied in [5].

While process mining and change mining have been considered as important topics in the last decades, none of the work deals with the topics as presented in this thesis.

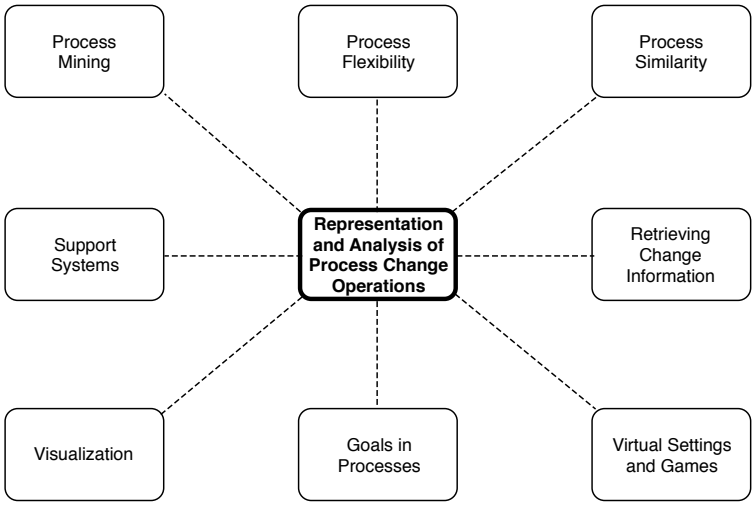


Figure 9.1: Related Work: Overview over relevant research areas

Process flexibility has been discussed in literature for several years [3]. In [21], the authors discuss the formal semantics of fourteen different change patterns, including patterns like insertion, deletion, movement or swapping of process fragments.

Soundness notions and checks for the application of change operations to process models and instances have been subject to several approaches. Structural soundness with respect to control and data flow has been tackled in [95], where the authors discuss the requirements change operations must fulfill in order to keep the process instance's correctness and consistency intact. Correctness criteria for the behavioral soundness of change operations are provided and compared in [96]. An overview on how to define and apply change operations on business processes is provided in [3].

[97] discusses a more relaxed compliance notion for such a situation. In this work, relaxed compliance notions for process change and evolution are presented.

[9] and [12] present change processes as a mean to represent process change logs. This method of analyzing change logs provides valuable information, especially for process instances which are based on a common schema, for example, on how to improve the process schema itself. Change processes focus on causal relations between process change operations. If a change operation Δ_1 is followed by Δ_2, it means that Δ_2 may be caused by Δ_1. For parallel change operations, such a condition cannot be drawn from the change log.

Adaptive Case Management (ACM) [98, 99, 100] is useful for flexible settings where a case is central to the environment such as a patient in a medical setting.

Further case-centric applications are social enterprises [101], or integrated care services [102], where a case evolves over time and always has to be adapted according to the current requirements. The concept of case-centric business processes has first been described in 2005 in [103]. Whereas ACM is a special approach towards representing processes, EPCT provides a data structure to represent and recommend process changes based on change and execution logs. Whenever this information is available, the EPCT can be applied, e.g., for procedural processes.

Declarative processes [104, 105, 106] allow for a high degree of flexibility as well. They are built on the idea of allowing all combinations of activity execution traces, except for those where specific constraints have been implied. Hence, a change log for a declarative process instance does not contain the insertion or deletion of activities, but the insertion or deletion of constraints to the process instance. Following, this information cannot be used to directly generate an EPCT.

While change logs provide a solid basis for analyzing conducted change operations and their consequences, they may not always be available. Other approaches for *retrieving change information* exist as well: Concept drift [56, 107, 57, 108, 109, 110, 111] focuses on analyzing process execution logs to detect any executed changes on the basic schema. By analyzing the activities present in the change log over a long period of time, this approach can estimate if and which changes have been applied to a process schema.

In contrast to change operation similarity, process equivalence and *similarity* have been analyzed frequently in current literature. Graph similarity algorithms [112], which can also be applied to process graphs, have been discussed in literature. [61] discusses label equivalence, attribute equivalence, position equivalence and regional equivalence for processes which execute web services. [113] uses behavioral profiles of processes to compare their similarity. Comparing these profiles leads to a behavior based matching of processes.

Van Dongen et al. discuss causal footprints, *"which is a collection of the essential behavioral constraints imposed by a process model"* [34] as a representation of a process model's behavior. Using causal footprint's *look-back* and *look-ahead* links, one could analyze and compare the positions of process change operations. [29] defines three metrics for measuring the similarity between process models, namely (a) node matching similarity, (b) structural similarity and (c) behavioral similarity. [114] defines the difference between two process models by a difference model that is visualized based on a difference graph. Doing so, differences between models can be visually inspected. [30] proposes similarity metrics between process instances, taking into consideration different perspective as well. [115] presents an approach to generate clusters of process models based on activity similarity, or pattern analysis. [116] discusses how semantically annotated business process models can be compared, where ambiguities, which are caused by different abstraction levels, or

the usage of synonyms are solved by using ontologies. Similarities between process models can also be used for process modeling: [117] develops an approach to create a new process model based on information from a base model, and an assessment of the similarities to the parameters of the situation, where the model which has to be created is going to be applied in.

[118] focuses on analyzing process variants, which are derived from a common process schema. The authors' goal is to find the process schema, where the smallest set of changes has to be applied to in order to obtain the schema of the individual process instances. In a first scenario, the existence of a reference process schema is assumed. This reference schema is adapted such that as few as possible additional changes will be necessary to reflect the process instances. In the second part, a process schema is generated solely based on the process instances and their changes. This approach does not construct analysis models from change logs, but it can serve as valuable complement to change operation similarity, as presented in this thesis. For example, one could find the change with the smallest set of required changes and use it as a basis for future changes.

In [35], the authors present an overview over existing approaches to determine the similarity of two process models. The presented approaches range from label matching similarity, over feature-based similarity estimations up to comparisons of common nodes and edges in process model graphs. These approaches can also be interesting for change operation similarity measures, since they can be used to compare attributes of change operations, like the schema, the applied fragment, or the position of the change.

Process change operations are usually applied in order to reach a desired effect, i.e., a goal. *Goals* which should be reached during process execution have been discussed in existing literature. [82] puts goals into the center of business processes. Goals can be seen as positive, i.e., some state which should be achieved, or negative, i.e., some state which should be avoided at all costs. The authors argue that each process is designed to reach or avoid some goals, so they propose a method of designing business processes around its goals. [119] combines processes modeled in BPMN with goals modeled in KAOS [120] to ensure that processes adhere to the high level stakeholder goals in the background. They make use of traceability links between goal nodes and process tasks to define that the goal has some impact on the task at hand, and satisfaction links to define that the task at hand satisfies a certain goal. These satisfaction links are defined over the concept of effects a process task has. An effect consists of a label, a designation, an informal and a formal definition, which help when mapping the tasks to the goals defined in KAOS. [121] analyzes event logs to find out, which process variants align to a certain set of goals which have been defined for the process instances at hand. This way, one can find out if a certain set of process instances still adhere to the

goals they should reach. [122] describes how business goals can be used to derive a process which aims at fulfilling those goals. It relies on a notation of effects of the tasks which are the basic building blocks of the resulting process which has to be connected with the ontology of the goals. Process goals are usually defined in Conjunctive Normal Form (CNF). In [123], the authors use a goal model consisting of goals and related sub-goals and an execution log to identify process variants, align subgoals to these process variants and to discover the relevant contextual factors. They analyze the effects of process tasks on impacted objects as effect events, which are denoted by an effect as an assertion in a first-order language (e.g., in CNF) and a timestamp. [31] defines the effects of business process tasks using Controlled Natural Language (CNL), such as Attempto Controlled English [124] and accumulates the effects of previous tasks to get a picture of what the actual effects until that tasks have been.

Goals can be seen as relevant related work to the contexts of the presented filtering and projection techniques for change traces, as well as the *enhanced process change tree (EPCT)*: In these representation and analysis methods, filter conditions can be used to filter those change traces, possibly projected to a certain case, where the data in the corresponding execution log matches the filter condition. This data can also reference a desired effect, i.e., a goal, which has been reached in the past (cf. Chapter 3). Following, these representation and analysis methods do not rely on complex dependencies between goals such as they can be defined using KAOS, nor do they require effects to be defined for each executed process task. Instead, all required information should be directly obtainable from a process instance's execution log.

Support systems, i.e., approaches which aim at supporting users during process adaptation have been developed before: ProCycle [8] uses case-based reasoning to determine the problem, and suggesting which adaptations could be applied in order to solve it. Using question-answer pairs, the user is guided to a specific case containing exactly one change operation. Thus, ProCycle only supports single process change operations, while the EPCT supports traces of change operations. In contrast to the EPCT, by implementing a case-based reasoning approach, ProCycle starts with the symptoms: A user first answers questions such as *Does the patient cough?* or *Is the patient's body temperature above 37 degrees Celsius?*, thus providing an in-depth analysis of the symptoms. In contrast, the EPCT assumes that the problem is already known to the user responsible for the adaptation, and provides user support from this point onward. However, existing case-based reasoning approaches [125] for finding the correct case based on the symptoms could be easily added without affecting the architecture of the EPCT. The evaluation of an adaptation's effectiveness is measured regarding a *reputation score* in ProCycle. Thus, the rating of an adaptation is not only not automated, but also dependant

on the individual opinion of the person who assigns the score. In contrast to this, in the EPCT the evaluation of the successfulness of a (trace of) change operation(s) is done automatically based on historic data. In ProCycle, it is also not possible to change the definition of the desired effects: A single change operation is assigned to a case, and this case describes which effect has to be reached (e.g., to treat a patient's flu). In contrast to this, for the EPCT effects can be defined based on boolean conditions for all data available in the processes execution log, thus allowing for very flexible effect definitions.

Summarizing, while ProCycle provides good support for single change operations, the EPCT supports traces of change operations, flexible effect definitions, and thus automated evaluation of applied change operations.

AgentWork [126] uses rules to define which situations make adaptations necessary, and which adaptations should be conducted in such a situation. These situations range from hardware and software problems to general exceptions which can occur while a system is running.

Visualization of processes has been an important topic of business process management since the very beginning. [127] from 1986 discusses some of the most common issues when designing process visualization in a way that also persons who are not experienced with programming languages can construct process visualizations in a way that data related to the process can be monitored whenever it changes.

Approaches for visualizing process change information exist as well: While [128] discusses the basic requirements for visualizing process adaptations, [129] presents an approach to visualize changes in a processes control flow by means of a *change tracking graph* or differencegraph [130]: When applying several change primitives to a process model (i.e., a graph), nodes and edges are inserted (removed) to (from) the graph. These changes can be visualized using properties such as the color, or the brightness of the respective elements of a model. In the presented approach, removal of edges and nodes is represented as showing these edges and nodes in orange, while insertion is represented by coloring respective nodes and edges green. [131] compares different approaches which can be used to visualize change operations in a change graph, including brightness, font size, line width, edge patterns, symbols, background color, color, edge endings and node shape. They found that *color* resulted in the best expressiveness.

An important aspect to be considered when visualizing change operations is the concept of the user's *mental map* [132, 133] of the graph: In order for users working with the graph to know immediately what has changed and which aspects have remained the same, it is important that the graphs before and after the change are as similar as possible (*layout stability* [134]). Following, unnecessary reordering of nodes and edges in the graphs should be avoided.

Aside from visualizing control flow changes in process models, [135] shows a method for visualizing changed instance traffic in combination with control flow changes: Using colors and the width of the connecting arrows as indicators, users can see not only which edges and activities have been added or removed, but also how the instance traffic has been affected. In addition to graph visualizations, other visualization concepts can come into play as well: [136] presents an approach based on stacked bar charts to visualize conducted change operations between multiple versions of a process model.

Business process compliance is another interesting topic for visualization, especially when it comes to process change operations: [137, 138] present a visualization framework which enables users to monitor process compliance along different perspectives.

Virtual settings have already been used for process modeling. In [139] the authors have created a 3D virtual world to enhance the collaborative modeling of business processes - even if the participants are not in geographical vicinity. By using avatars - 3D representations of humans who engage in the modeling of business processes, this approach offers various forms of communication which would be typically only possible if the participants were in geographical vicinity, i.e., speech, artifacts, gestures etc. *Ross Brown et al.* [140, 141] present another virtual world approach for generating process models. In contrast to other approaches, participants do not have to know any modeling language in order to generate a process model. Instead, they behave as they would do in the real world. Based on their actions in a realistic environment, a process model is generated.

While developing the game-based environment, we had to design the game mechanics according to the FIPS requirements, while still making the game fun to play. This balancing act is also an issue in serious games such as educational games [142, 143], where fun game mechanics have to be balanced with a serious background [144]. Analyzing user behavior is also an important aspect of game development. [145] presents an approach to visualize player behavior, thus providing a method to present information about player actions for games, where those actions cannot be captured by process change and execution log files.

Data obtained from computer games, especially from strategy games, has been used for the development and validation of artificial intelligence approaches on multiple occasions. In [85], the authors show that tower defense games provide a good test environment for numerous computational intelligence scenarios, ranging from map generation, over enemy strategies, good defense strategies up to dynamic game balancing which keeps the player engaged and the enemies challenging. In [146], the authors provide an overview over the possibilities of artificial intelligence in a real time strategy environment.

Virtual environments have been exploited as experimentation settings for busi-

ness processes before. [140] presents role-playing as a way to generate process models: Users are in a virtual airport and have to take all necessary actions to go from the entrance to their plane. Based on the users actions, a process model is generated automatically. In the *Alaska Simulator* [83, 84] users can plan, simulate and analyze different approaches to process flexibility based on the example of a journey to Alaska. This setting helps researchers to set up controlled experiments to find out about advantages and disadvantages of different approaches for decision deferral, i.e., traditional workflows, late binding, late modeling and late composition.

Chapter 10

Conclusion

This chapter concludes the thesis. First, we give a summary of the contributions this thesis has created for the field of business process management in general, and process change operations, traces and logs in particular (Section 10.1). This is followed by a reflection of the research questions as set out in the introduction, and how they have been answered throughout this thesis (Section 10.2). Additionally, we discuss limitations of the approaches provided in this section, and provide an outlook on future work (Section 10.3).

10.1 Summary

The goal of this thesis is to bridge the gap between the change information which is present in process change logs at a technical level on the system side, and the domain experts who have to interact with change operations on the user side. Hence, in this thesis, we have provided several approaches for representing and analyzing single process change operations as well as process change traces from various perspectives. For single process change operations (Chapter 2), we have introduced several *change similarity metrics*, which can be used as a basis to compare two change operations. While the *attribute-based* similarity metric focuses on the formal description of the change operations (what has been changed at which position of a certain schema), the *change resource similarity* and the *timed change resource similarity* metrics both focus on different aspects, i.e., the temporal and resource perspective. Using such process similarity metrics, domain experts can, for example, compare possible adaptations when they have to decide for a change operation which should be applied to a process instance.

Single process change operations are the basic building blocks of change traces and change logs. We have analyzed change traces (Chapter 3) and provided an approach for representing them in XES, thus tackling the machine readable rep-

resentation for process log files, as well as algorithms for filtering, and projecting. These approaches pursued the goal of overcoming the information overload problem [11]: Domain experts should be able to filter and zoom to the data which is relevant to their current analysis question, without being distracted by non related data. Finally, we have analyzed an existing technique for analyzing change traces, i.e., change processes, and highlighted some shortcomings of this representation when the chronological order of change operations are of interest. As discussed in Chapter 1, being able to analyze the chronological order of change operations can be of interest for a variety of different situations: If, for example, users have to analyze the documentation of the evolution history of a process instance, they could consult a representation which highlights these aspects: Based on such a representation, they could analyze the reasons the process instances have evolved up to their current state. Based on the shortcomings we have found for existing representations regarding the representation of the chronological order of applied change operations, we have developed a new representation of change traces, i.e., the change tree (Chapter 4). By aggregating change tree nodes using change similarity metrics, the amount of nodes in a change tree can be reduced, such that the domain expert can focus on one aspect (defined by the similarity metric which has been used for aggregation). By applying filtering and projection techniques, the change tree can be enhanced with possible effects of the applied change operations: Domain experts can see how many times a certain effect has been reached when a certain path in the change tree has been followed. The approaches developed in this thesis have been evaluated in three ways: First, we have created a prototypical implementation (Chapter 5), and discussed several application settings (Chapter 6). We have discussed *flexible and individual process settings (FIPS)* in detail (Chapter 7), since there, change operations are an integral part of the process design, which leads to a high need of representation and analysis techniques for change operation traces. In order to simulate and evaluate our approaches in a FIPS, we have created a game-based experimentation setting, which has been used as a basis for a part of our user-based evaluation. Here, we have conducted two experiments (Chapter 8): In our first experiment, we analyzed whether change trees and aggregated change trees (by applying change similarity metrics to change trees) provide a better understanding of change traces than the raw data provided in XML. In our second experiment, we analyzed whether filtered and projected change traces can be used as a basis for user support.

10.2 Reflection

In the introduction, we stated four research questions, which have been addressed in this thesis. In this section we summarize the results of the thesis for those

research questions:

Q1 **How can domain experts compare process change operations, such that a detailed analysis of single process change operations is possible?**

In Chapter 2, we have discussed how single process change operations can be compared. For this, two types of metrics have been developed: Attribute-based change operation similarity metrics can be used to compare two change operations based on their attributes. By weighting the influence on the overall similarity score, attributes can be weighted higher or lower, thus allowing for additional flexibility. Effect-based change operation similarity addresses the effects a change operation has on the corresponding process instance. We have developed two metrics, i.e., *Change Resource Similarity* and *Timed Change Resource Similarity*, which addresses resource and time perspectives of the affected process activities. Finally, we have discussed the importance of the different weights, especially for attribute-similarity metric, and we have shown that both effect similarity metrics discussed in this thesis correlate with the similarity of the adapted process schemata.

When developing the change similarity metrics, we first set our focus on the attribute-based similarities. However, the choice of the correct weights has proved to be a tough point for discussion: As we have shown, depending on the chosen weight, one can influence the overall similarity substantially. Hence, when using this similarity metric, one always needs to present the reasons and arguments for the chosen weights. Based on such discussions, we have developed the effect-based similarity metrics: Here, no weights are required which influence the outcome of the similarity metric.

Q2 **How can we overcome the information overload problem for process change logs? Based on which data?**

The information overload problem is a common obstacle, especially for visual analysis of data [11]. In this thesis, we have presented filtering and projection techniques for change traces. The goal of these techniques is to project the change operations in the change traces such that each change trace only contains those adaptations which have been executed due to the same reason. Additionally, by implementing filtering techniques, we have presented a tool for reducing the amount of change traces present in a process change log. Using such filtered and projected change logs as a basis, domain experts can focus on the data which is important for their current analysis question: nurses in a nursing home, for example, can project the change traces such that each trace contains all activities which have been executed because a

patient with diabetes and a certain set of allergies has the flu. Based on such a projected and filtered data set, necessary information can be deduced more easily. Additionally, the node aggregation in the change tree also allows users to focus on the necessary information: By choosing a fitting similarity metric, one can reduce the number of nodes present in a change tree, thus allowing to focus on the required information.

When analyzing change logs during the development of this thesis, and when developing the process change tree, the differences in the applied process change operations always posed a certain challenge: If one single attribute of an applied change operation was different from the rest - even, if it was not relevant to the current analysis question - a new branch in the change tree has been generated. This soon led to the problem of change trees becoming very broad, rendering a useful analysis practically impossible. When analyzing raw change logs, this was even more problematic: Without the possibility of setting the focus on the relevant data, change logs contained too much irrelevant information, thus rendering a useful analysis practically impossible. By applying filtering and projection techniques, as well as similarity metrics, we have been able to provide techniques for solving such problems.

Q3 **Which properties are required for a representation of process change logs, which highlights the evolution of a set of process instances from their original schema to their current state?**

The change tree is a representation of process change logs which highlights the chronological order in which process change operations have been applied. In this thesis, we have developed the representation, as well as algorithms for updating, and creating subsets based on n-grams. Using the change tree as a basis, domain experts can analyze the evolution of a set of process instances based on change log data: Production manager in a manufacturing environment can see, how the production process instances of a certain set of products have evolved over time. When combining such data with additional information, e.g., when a supplier did not deliver, or some production machine stopped working, the reasons for the process instances' evolution can be deduced. However, change trees can grow in size very easily.

While the change tree is a great tool for analyzing the chronological order of applied process change operations, we soon saw the need for setting the focus on the relevant data: By creating subsets of the change tree using n-grams, we have provided such a method: Using n-grams, domain experts can analyze which change operations have followed a certain (set of) change operation(s): In combination with similarity measures, one could, for example, analyze which resources have been used in the past after a certain adaptation has

been made.

Q4 **What are possible application scenarios for process change operations? Can the concepts developed in this thesis also be applied to settings where change operations occur frequently? How can users benefit from the concepts developed in this thesis?**

The application of the presented concepts to different domains, especially to the killer applications, flexible and individual process settings, has been a central part of the evaluation of this thesis: Here, we have analyzed which data is available in such a setting, and what can be gained from applying the representation and analysis methods.

Overall, we have discussed various application scenarios for the representation and analysis techniques developed in this thesis. First, we have shown that the change tree as a basic data structure is not only limited to process change logs, but can also be applied to analyze execution log. Second, we have presented possible tools and user interfaces which can be developed based on the concepts developed in this thesis for two distinct domains. For the wellbeing domain, we have shown how the concepts can be integrated into a mobile application which can then be used by patients to reduce the amount of pain they experience on a daily basis. For the nursing domain, we have shown how these tools can be integrated into *ACaPlan*, a project which focuses on developing tools and user interfaces for nurses who plan and interact with therapy process instances. Third, we have discussed *flexible and individual process settings* as the *killer application* for flexible process instances, and we have presented a game-based experimentation service for such settings. This experimentation service can be used both for the creation of change and corresponding execution logs, as well as a controlled environment for conducting user evaluations.

10.3 Future Work

This thesis provides concepts for analyzing single process change operations, as well as change traces and change logs. In this section, we discuss possible extensions and future work.

The similarity of single process change operations can be extended with additional features, such as similarity metrics which focus on the control or data flow effects of applied process change operations: Such metrics can then analyze how similar the effects of two change operations have been regarding those two dimensions: Do these two change operations affect the same, or totally different data items? How many activities have been inserted, and what about the position

of the change operations? Are the conducted change operations part of a loop? If yes, what does that mean for the overall similarity? Partly, such questions could be answered by applying the attribute-based similarity metric as presented in this thesis with respective weights: For example, if one is only interested in the activities which are affected by the change operations, such an analysis can partly be completed by assigning the respective weights as discussed in Chapter 2. However, additional work here may improve user performance.

Visualization concepts can additionally improve the usability of process change similarity, especially in combination with change trees and process change log filtering and projection techniques: For process changes, visualization techniques already exist [128, 130], however, extending them to also cover similarity metrics may increase understandability of the similarity of applied change operations a lot.

Besides visualization techniques, sonification tackles how data can be represented by sound waves, *"in order to enhance process visualizations"* [147]. Especially for flexible and individual process settings, where change operations are the main driver of the process design, and thus appear on a regular basis, such a combination of visual and audio effects can provide additional support when users decide for a process adaptation.

Overall, this thesis presents several representation and analysis methods for single process change operations, as well as change traces and change logs. These concepts allow for a variety of extensions and future work, which can help to bridge the gap between the domain experts on the user side and the technical implementation on the system side even more.

Appendices

The appendix lists all information which were too long for the chapters.

A User Experiment: Change Trees

For group one, the following questions were asked:

- **Question 1:**
  - **Question:** How many nurses are required overall in this change log?
  - **Explanation:** Calculate, how many nurses occur as resources in the change log over all activities. The exact days when these resources are being required are not of interest for this question and can be ignored.
  - **Possible answers:** *free text answer*
  - **Correct answer:** 8
  - **Representation:** Change Tree
- **Question 2:**
  - **Question:** Which resource is being used the most?
  - **Explanation:** Take a close look at the resources required for each process fragment. Count, how many times each resource appears. (Take care that a resource can appear more than once in one fragment), and choose the resource, which appears the most.
  - **Possible answers:** doctor, nurse, assistant
  - **Correct answer:** doctor
  - **Representation:** Change Tree
- **Question 3:**
  - **Question:** How many assistants are required overall in this change log?

- **Explanation:** Calculate, how many assistants occur as resources in the change log over all activities. The exact days when these resources are being required are not of interest for this question and can be ignored.
- **Possible answers:** *free text answer*
- **Correct answer:** 8
- **Representation:** Aggregated Change Tree - Change Resource Similarity

- **Question 4:**
  - **Question:** Which resource is being used the most?
  - **Explanation:** Take a close look at the resources required for each process fragment. Count, how many times each resource appears. (Take care that a resource can appear more than once in one fragment), and choose the resource, which appears the most.
  - **Possible answers:** doctor, nurse, assistant
  - **Correct answer:** assistant
  - **Representation:** Aggregated Change Tree - Change Resource Similarity

- **Question 5:**
  - **Question:** Which resource is being used the most?
  - **Explanation:** Take a close look at the resources required for each process fragment. Count, how many times each resource appears. (Take care that a resource can appear more than once in one fragment), and choose the resource, which appears the most.
  - **Possible answers:** doctor, nurse, asssistant
  - **Correct answer:** nurse
  - **Representation:** XES Change Log

- **Question 6:**
  - **Question:** How many nurses are required overall in this change log?
  - **Explanation:** Calculate, how many nurses occur as resources in the change log over all activities. The exact days when these resources are being required are not of interest for this question and can be ignored.
  - **Possible answers:** *free text answer*

  - **Correct answer:** 3
  - **Representation:** XES Change Log

- **Question 7:**
  - **Question:** Which resource is being used the most?
  - **Explanation:** Take a close look at the resources required for each process fragment. Count, how many times each resource appears. (Take care that a resource can appear more than once in one fragment), and choose the resource, which appears the most.
  - **Possible answers:** doctor, nurse, asssistant
  - **Correct answer:** assistant
  - **Representation:** Change Tree

- **Question 8:**
  - **Question:** How many assistants are required overall in this change log?
  - **Explanation:** Calculate, how many assistants occur as resources in the change log over all activities. The exact days when these resources are being required are not of interest for this question and can be ignored.
  - **Possible answers:** *free text answer*
  - **Correct answer:** 3
  - **Representation:** Change Tree

For group two, the ordering of these questions were mixed: For example, question 1 and 2 from group one target the change tree representation, while question 3 and 4 target the aggregated change tree respresentation. Group two received the same questions, but here, question 1 and 2 target the aggregated change tree, while questions 3 and 4 target the change tree based upon label equivalence. The same system is applied for the comparison of change trees and change logs in the XES format: While for group one, questions 5 and 6 target the change log in XES, and questions 7 and 8 target the change tree, for group two, questions 5 and 6 target the change log in XES, while questions 7 and 8 target the change tree.

By keeping the questions similar, but presenting the questions to the participants in a ranomized order, the comparability of the results is increased.

Bibliography

[1] Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A., et al.: Fundamentals of business process management. Volume 1. Springer (2013)

[2] Chriusty Pettey, R.v.d.M.: Gartner says enterprises can experience as much as a 20 percent cost savings within first year of using bpm to improve business processes. Gartner Press Release (2009)

[3] Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer (2012)

[4] Creasey, T.: The costs and risks of poorly managed change. Prosci Press Release (2018)

[5] Fdhila, W., Rinderle-Ma, S., Knuplesch, D., Reichert, M.: Change and compliance in collaborative processes. In: Services Computing (SCC), 2015 IEEE International Conference on, IEEE (2015) 162–169

[6] Ly, L.T., Rinderle, S., Dadam, P.: Integration and verification of semantic constraints in adaptive process management systems. Data & Knowledge Engineering **64** (2008) 3 – 23 Fourth International Conference on Business Process Management (BPM 2006) 8th International Conference on Enterprise Information Systems (ICEIS' 2006).

[7] Ly, L.T., Rinderle-Ma, S., Göser, K., Dadam, P.: On enabling integrated process compliance with semantic constraints in process management systems. Information Systems Frontiers **14** (2012) 195–219

[8] Weber, B., Reichert, M., et al.: Providing integrated life cycle support in process-aware information systems. Int. J. Cooperative Inf. Syst. **18** (2009) 115–165

[9] Günther, C.W., Rinderle, S., Reichert, M., van der Aalst, W.: Change mining in adaptive process management systems. In Meersman, R., Tari, Z., eds.: On the Move to Meaningful Internet Systems 2006: CoopIS, DOA,

GADA, and ODBASE, Berlin, Heidelberg, Springer Berlin Heidelberg (2006) 309–326

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

[11] Keim, D., Andrienko, G., Fekete, J.D., Görg, C., Kohlhammer, J., Melançon, G.: Visual analytics: Definition, process, and challenges. In: Information visualization. Springer (2008) 154–175

[12] Günther, C., Rinderle-Ma, S., Reichert, M., van der Aalst, W.: Using process mining to learn from process changes in evolutionary systems. International Journal of Business Process Integration and Management **3** (2008) 61–78

[13] Mans, R., Schonenberg, M., Song, M., van der Aalst, W., Bakker, P.: Application of process mining in healthcare - a case study in a dutch hospital. In Fred, A., Filipe, J., Gamboa, H., eds.: Biomedical Engineering Systems and Technologies. Volume 25 of Communications in Computer and Information Science. Springer Berlin Heidelberg (2009) 425–438

[14] van der Aalst, W., Pesic, M., Song, M.: Beyond process mining: From the past to present and future. In Pernici, B., ed.: Advanced Information Systems Engineering. Volume 6051 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2010) 38–52

[15] Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. Journal of management information systems **24** (2007) 45–77

[16] Kaes, G., Rinderle-Ma, S.: On the similarity of process change operations. In: Advanced Information Systems Engineering. (2017) 348–363

[17] Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases **16** (2004) 91–116

[18] Lanz, A., Reichert, M., Weber, B.: Process time patterns: A formal foundation. Inf. Syst. **57** (2016) 38–68

[19] Rinderle, S., Reichert, M., Jurisch, M., Kreher, U.: On representing, purging, and utilizing change logs in process management systems. In: Business Process Management. Springer (2006) 241–256

[20] Li, C., Reichert, M., Wombacher, A.: On Measuring Process Model Similarity Based on High-Level Change Operations. In: Conceptual Modeling - ER 2008: 27th International Conference on Conceptual Modeling, Barcelona, Spain, October 20-24, 2008. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2008) 248–264

[21] Rinderle-Ma, S., Reichert, M., Weber, B.: On the formal semantics of change patterns in process-aware information systems. In: Conceptual Modeling - ER 2008. Volume 5231 of LNCS. Springer (2008) 279–293

[22] Fdhila, W., Rinderle-Ma, S., Reichert, M.: Change propagation in collaborative processes scenarios. In: Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on, IEEE (2012) 452–461

[23] Fdhila, W., Indiono, C., Rinderle-Ma, S., Vetschera, R.: Finding collective decisions: Change negotiation in collaborative business processes. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer (2015) 90–108

[24] Fdhila, W., Rinderle-Ma, S., Indiono, C.: Change propagation analysis and prediction in process choreographies. International Journal of Cooperative Information Systems **24** (2015) 1541003

[25] Fdhila, W., Indiono, C., Rinderle-Ma, S., Reichert, M.: Dealing with change in process choreographies: Design and implementation of propagation algorithms. Information systems **49** (2015) 1–24

[26] Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data & Knowledge Engineering **68** (2009) 793–818

[27] Paurobally, S., Tamma, V., Wooldrdige, M.: A framework for web service negotiation. ACM Transactions on Autonomous and Adaptive Systems (TAAS) **2** (2007) 14

[28] Vigne, R., Mangler, J., Schikuta, E., Rinderle-Ma, S.: Ws-agreement based service negotiation in a heterogeneous service environment. In: Service-Oriented Computing and Applications (SOCA), 2012 5th IEEE International Conference on, IEEE (2012) 1–8

[29] Dijkman, R., Dumas, M., van Dongen, B., Käärik, R., Mendling, J.: Similarity of business process models: Metrics and evaluation. Information Systems **36** (2011) 498 – 516 Special Issue: Semantic Integration of Data, Multimedia, and Services.

[30] Pflug, J., Rinderle-Ma, S.: Process instance similarity: Potentials, metrics, applications. In: OTM 2016 Conferences. (2016) 136–154

[31] Hinge, K., Ghose, A., Koliadis, G.: Process seer: A tool for semantic effect annotation of business process models. In: Enterprise Distributed Object Computing Conference. (2009) 54–63

[32] Xu, H., Savarimuthu, B.T.R., Ghose, A., Morrison, E., Cao, Q., Shi, Y.: Automatic BDI plan recognition from process execution logs and effect logs. In: Engineering Multi-Agent Systems. Springer (2013) 274–291

[33] Russell, N., ter Hofstede, A.H.M., et al.: Workflow data patterns: Identification, representation and tool support. In: Conceptual Modeling. Springer (2005) 353–368

[34] Van Dongen, B., Dijkman, R., Mendling, J.: Measuring similarity between business process models. In: Seminal Contributions to Information Systems Engineering. Springer (2013) 405–419

[35] Becker, M., Laue, R.: A comparative survey of business process similarity measures. Computers in Industry **63** (2012) 148 – 167

[36] Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions and reversals. In: Soviet physics doklady (In Russian). Volume 10. (1966) 707–710

[37] Kaes, G., Rinderle-Ma, S.: Mining and querying process change information based on change trees. In: Service-oriented Computing, Springer (2015) 269–284

[38] van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The prom framework: A new era in process mining tool support. In Ciardo, G., Darondeau, P., eds.: Applications and Theory of Petri Nets 2005, Berlin, Heidelberg, Springer Berlin Heidelberg (2005) 444–454

[39] Aigner, W., Miksch, S., Müller, W., Schumann, H., Tominski, C.: Visual methods for analyzing time-oriented data. IEEE transactions on visualization and computer graphics **14** (2008) 47–60

[40] van Dongen, B.F., Van der Aalst, W.M.: A meta model for process mining data. EMOI-INTEROP **160** (2005) 30

[41] Verbeek, H., Buijs, J., van Dongen, B., van der Aalst, W.: Xes tools. In: CAiSE Forum. (2010) 1–8

[42] Van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering **16** (2004) 1128–1142

[43] van Dongen, B.F., van der Aalst, W.M.P. In: Multi-phase Process Mining: Building Instance Graphs. Springer Berlin Heidelberg, Berlin, Heidelberg (2004) 362–376

[44] van Dongen, B.F., Van der Aalst, W.M.: Multi-phase process mining: Aggregating instance graphs into epcs and petri nets. In: PNCWB 2005 workshop, Citeseer (2005) 35–58

[45] ter Hofstede, A., Ouyang, C., La Rosa, M., Song, L., Wang, J., Polyvyanyy, A.: Apql: A process-model query language. In Song, M., Wynn, M., Liu, J., eds.: Asia Pacific Business Process Management. Volume 159 of Lecture Notes in Business Information Processing. Springer International Publishing (2013) 23–38

[46] Polyvyanyy, A., ter Hofstede, A.H., La Rosa, M., Ouyang, C.: Process query language: Design, implementation and evaluation. BPM Center Report BPM-15-06, BPMcenter. org (2015)

[47] Polyvyanyy, A., Ouyang, C., Barros, A., van der Aalst, W.M.: Process querying: Enabling business intelligence through query-based process analytics. Decision Support Systems **100** (2017) 41–56

[48] Beheshti, S.M.R., Benatallah, B., Motahari-Nezhad, H., Sakr, S.: A query language for analyzing business processes execution. In Rinderle-Ma, S., Toumani, F., Wolf, K., eds.: Business Process Management. Volume 6896 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 281–297

[49] Sikorski, R.: Finite joins and meets. In: Boolean Algebras, Berlin, Heidelberg, Springer Berlin Heidelberg (1960) 3–54

[50] Kaes, G., Rinderle-Ma, S.: Improving the usability of process change trees based on change similarity measures. In: International Working Conference on Business Process Modeling, Development, and Support. (2018)

[51] Brown, P., Desouza, P., Mercer, R., Della Pietra, V., Lai, J.: Class-based n-gram models of natural language. Computational linguistics **18** (1992) 467–479

[52] McCreight, E.M.: A space-economical suffix tree construction algorithm. Journal of the ACM **23** (1976) 262–272

[53] Sagot, M.F.: Spelling approximate repeated or common motifs using a suffix tree. In Lucchesi, C.L., Moura, A.V., eds.: LATIN'98: Theoretical Informatics, Berlin, Heidelberg, Springer Berlin Heidelberg (1998) 374–390

[54] Ukkonen, E.: On-line construction of suffix trees. Algorithmica **14** (1995) 249–260

[55] Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., Shan, M.C.: Business process intelligence. Computers in Industry **53** (2004) 321 – 343 Process / Workflow Mining.

[56] Bose, R.P.J.C., van der Aalst, W.M.P., Žliobaitė, I., Pechenizkiy, M.: Handling concept drift in process mining. In Mouratidis, H., Rolland, C., eds.: Advanced Information Systems Engineering, Berlin, Heidelberg, Springer Berlin Heidelberg (2011) 391–405

[57] Carmona, J., Gavaldà, R.: Online techniques for dealing with concept drift in process mining. In: Advances in Intelligent Data Analysis XI. (2012) 90–102

[58] Vogelgesang, T., Kaes, G., Rinderle-Ma, S., Appelrath, H.J.: Multidimensional process mining: Questions, requirements, and limitations. In: CAISE 2016 Forum. CAISE 2016 Forum (2016) 169–176

[59] Kaes, G., Mangler, J., Rinderle-Ma, S., Vigne, R.: The NNN formalization: Review and development of guideline specification in the care domain. CoRR **abs/1404.2162** (2014)

[60] Ingvaldsen, J.E., Gulla, J.A.: Industrial application of semantic process mining. Enterprise Information Systems **6** (2012) 139–163

[61] Rinderle-Ma, S., Reichert, M., Jurisch, M.: On utilizing web service equivalence for supporting the composition life cycle. International Journal of Web Services Research **8** (2011) 41–67

[62] Bron, C., Kerbosch, J.: Algorithm 457: Finding all cliques of an undirected graph. Commun. ACM **16** (1973) 575–577

[63] Richardson, L., Ruby, S.: RESTful web services. " O'Reilly Media, Inc." (2008)

[64] Mangler, J., Rinderle-Ma, S.: Cpee - cloud process exection engine. In: Int'l Conference on Business Process Management. CEUR-WS.org (2014)

[65] Kaes, G., Mangler, J., Stertz, F., Vigne, R., Rinderle-Ma, S.: Acaplan - adaptive care planning. In: BPM'15 Demo Track, 13th International Conference on Business Process Management. (2015)

[66] Berson, A., Smith, S.J.: Data Warehousing, Data Mining, and Olap. 1st edn. McGraw-Hill, Inc., New York, NY, USA (1997)

[67] Böhmer, K., Rinderle-Ma, S.: Multi instance anomaly detection in business process executions. In: Int'l Conference on Business Process Management 2017. (2017) 77–93

[68] Böhmer, K., Rinderle-Ma, S.: Multi-perspective anomaly detection in business process execution events. In: International Conference on Cooperative Information Systems (CoopIS) 2016. (2016)

[69] Ly, L., Indiono, C., Mangler, J., Rinderle-Ma, S.: Data transformation and semantic log purging for process mining. In: Advanced Information Systems Engineering. Volume 7328 of LNCS. Springer (2012) 238–253

[70] Stertz, F., Mangler, J., Rinderle-Ma, S.: Nfc-based task enactment for automatic documentation of treatment processes. In: Enterprise, Business-Process and Information Systems Modeling. Springer (2017) 34–48

[71] Farmer, M.A., Baliki, M.N., Apkarian, A.V.: A dynamic network perspective of chronic pain. Neuroscience letters **520** (2012) 197–203

[72] Flor, H.: New developments in the understanding and management of persistent pain. Current opinion in psychiatry **25** (2012) 109–113

[73] Mansour, A., Farmer, M., Baliki, M., Apkarian, A.V.: Chronic pain: the role of learning and brain plasticity. Restorative neurology and neuroscience **32** (2014) 129–139

[74] Zeidan, F., Martucci, K.T., Kraft, R.A., Gordon, N.S., McHaffie, J.G., Coghill, R.C.: Brain mechanisms supporting the modulation of pain by mindfulness meditation. Journal of Neuroscience **31** (2011) 5540–5548

[75] Zeidan, F., Emerson, N.M., Farris, S.R., Ray, J.N., Jung, Y., McHaffie, J.G., Coghill, R.C.: Mindfulness meditation-based pain relief employs different neural mechanisms than placebo and sham mindfulness meditation-induced analgesia. Journal of Neuroscience **35** (2015) 15307–15325

[76] Hilton, L., Hempel, S., Ewing, B.A., Apaydin, E., Xenakis, L., Newberry, S., Colaiaco, B., Maher, A.R., Shanman, R.M., Sorbero, M.E., et al.: Mindfulness meditation for chronic pain: systematic review and meta-analysis. Annals of Behavioral Medicine **51** (2016) 199–213

[77] Cole, L.J., Bennell, K.L., Ahamed, Y., Bryant, C., Keefe, F., Moseley, G.L., Hodges, P., Farrell, M.J.: Determining brain mechanisms that underpin analgesia induced by the use of pain coping skills. Pain Medicine (2018)

[78] Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.: Process flexibility: A survey of contemporary approaches. In: Advances in Enterprise Engineering I. Springer (2008) 16–30

[79] Kaes, G., Rinderle-Ma, S., Vigne, R., Mangler, J.: Flexibility requirements in real-world process scenarios and prototypical realization in the care domain. In: OTM 2014 Workshops. (2014) 55–64

[80] Kaes, G., Rinderle-Ma, S.: Generating data from highly flexible and individual process settings through a game-based experimentation service. In: Datenbanksysteme für Business, Technologie und Web. (2017) 331–350

[81] Dadam, P., Reichert, M., Kuhn, K.: Clinical workflows—the killer application for process-oriented information systems? In: BIS 2000. Springer (2000) 36–59

[82] Kueng, P., Kawalek, P.: Goal-based business process models: creation and evaluation. Business Process Management Journal **3** (1997) 17–38

[83] Weber, B., Pinggera, J., Zugal, S., Wild, W. In: Alaska Simulator Toolset for Conducting Controlled Experiments on Process Flexibility. Springer Berlin Heidelberg, Berlin, Heidelberg (2011) 205–221

[84] Weber, B., Pinggera, J., Zugal, S., Wild, W.: Alaska simulator – a journey to planning. In Abrahamsson, P., Marchesi, M., Maurer, F., eds.: Agile Processes in Software Engineering and Extreme Programming, Berlin, Heidelberg, Springer Berlin Heidelberg (2009) 253–254

[85] Avery, P., Togelius, J., Alistar, E., van Leeuwen, R.P.: Computational intelligence and tower defence games. In: 2011 IEEE Congress of Evolutionary Computation (CEC). (2011) 1084–1091

[86] Van Welie, M., Van Der Veer, G.C., Eliëns, A.: Breaking down usability. In: Interact. (1999) 613–620

[87] Nielsen, J.: Usability engineering. Elsevier (1994)

[88] van der Aalst, W., et al.: Process mining manifesto. In Daniel, F., Barkaoui, K., Dustdar, S., eds.: Business Process Management Workshops, Berlin, Heidelberg, Springer Berlin Heidelberg (2012) 169–194

[89] Wen, L., Wang, J., van der Aalst, W.M., Huang, B., Sun, J.: A novel approach for process mining based on event types. Journal of Intelligent Information Systems **32** (2009) 163–190

[90] van Aalst, W., et al.: Auditing 2.0: Using process mining to support tomorrow's auditor. Computer **43** (2010) 90–93

[91] van der Aalst, W., Reijers, H., Weijters, A., van Dongen, B., de Medeiros, A.A., Song, M., Verbeek, H.: Business process mining: An industrial application. Information Systems **32** (2007) 713 – 732

[92] Dam, H.K., Ghose, A.: Mining version histories for change impact analysis in business process model repositories. Computers in Industry **67** (2015) 72 – 85

[93] Li, C., Reichert, M., Wombacher, A.: Mining based on learning from process change logs. In: International Conference on Business Process Management, Springer (2008) 121–133

[94] Fdhila, W., Rinderle-Ma, S., Indiono, C.: Memetic algorithms for mining change logs in process choreographies. In: International Conference on Service-Oriented Computing, Springer (2014) 47–62

[95] Reichert, M., Dadam, P.: Adeptflex—supporting dynamic changes of workflows without losing control. Journal of Intelligent Inf. Systems **10** (2011) 93–129

[96] Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems—a survey. Data & Knowledge Engineering **50** (2004) 9 – 34

[97] Rinderle-Ma, S., Reichert, M., Weber, B.: Relaxed compliance notions in adaptive process management systems. In: Conceptual Modeling. Springer (2008) 232–247

[98] Motahari-Nezhad, H.R., Swenson, K.D.: Adaptive case management: Overview and research challenges. In: 2013 IEEE 15th Conference on Business Informatics. (2013) 264–269

[99] Swenson, K.: Taming the unpredictable: real world adaptive case management: case studies and practical guidance. Future Strategies Inc. (2011)

[100] Herrmann, C., Kurz, M.: Adaptive case management: Supporting knowledge intensive processes with it systems. In Schmidt, W., ed.: S-BPM ONE - Learning by Doing - Doing by Learning, Berlin, Heidelberg, Springer Berlin Heidelberg (2011) 80–97

[101] Motahari-Nezhad, H., Bartolini, C., Graupner, S., Spence, S.: Adaptive case management in the social enterprise. Service-Oriented Computing (2012) 550–557

[102] Cano, I., Alonso, A., Hernandez, C., Burgos, F., Barberan-Garcia, A., Roldan, J., Roca, J.: An adaptive case management system to support integrated care services: lessons learned from the nexes project. Journal of biomedical informatics **55** (2015) 11–22

[103] Van der Aalst, W.M., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data & Knowledge Engineering **53** (2005) 129–162

[104] Pesic, M., Schonenberg, H., Van der Aalst, W.M.: Declare: Full support for loosely-structured processes. In: Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International, IEEE (2007) 287–287

[105] van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science - Research and Development **23** (2009) 99–113

[106] van der Aalst, W., et al.: Declarative workflows: Balancing between flexibility and support. Computer Science-Research and Development **23** (2009) 99–113

[107] Bose, R.J.C., Van Der Aalst, W.M., Zliobaite, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE transactions on neural networks and learning systems **25** (2014) 154–171

[108] Hompes, B., Buijs, J.C., van der Aalst, W.M., Dixit, P., Buurman, H.: Detecting change in processes using comparative trace clustering. In: SIMPDA. (2015) 95–108

[109] Luengo, D., Sepúlveda, M.: Applying clustering in process mining to find different versions of a business process that changes over time. In: International Conference on Business Process Management, Springer (2011) 153–158

[110] Martjushev, J., Bose, R.J.C., van der Aalst, W.M.: Change point detection and dealing with gradual and multi-order dynamics in process mining. In: International Conference on Business Informatics Research, Springer (2015) 161–178

[111] Weber, P., Bordbar, B., Tino, P.: Real-time detection of process change using process mining. In: ICCSW, Citeseer (2011) 108–114

[112] Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern Recognition Letters **19** (1998) 255 – 259

[113] Kunze, M., Weidlich, M., Weske, M.: m3 – a behavioral similarity metric for business processes. Services und ihre Komposition (2011)

[114] Kriglstein, S., Wallner, G., Rinderle-Ma, S.: A visualization approach for difference analysis of process models and instance traffic. In: Business Process Management. Springer (2013) 219–226

[115] Jung, J.Y., Bae, J.: Workflow clustering method based on process similarity. In Gavrilova, M., Gervasi, O., Kumar, V., Tan, C., Taniar, D., Laganá, A., Mun, Y., Choo, H., eds.: Computational Science and Its Applications - ICCSA 2006. Volume 3981 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2006) 379–389

[116] Ehrig, M., Koschmider, A., Oberweis, A.: Measuring similarity between semantic business process models. In: Proceedings of the Fourth Asia-Pacific Conference on Comceptual Modelling - Volume 67. APCCM '07, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2007) 71–80

[117] Lu, J., , Gao*, F.: Process modeling based on process similarity. Industrial & Engineering Chemistry Research **47** (2008) 1967–1974

[118] Li, C., Reichert, M., Wombacher, A.: Mining business process variants: Challenges, scenarios, algorithms. DKE **70** (2011) 409 – 434

[119] Koliadis, G., Ghose, A.: Relating Business Process Models to Goal-Oriented Requirements Models in KAOS. In: Advances in Knowledge Acquisition and Management: Pacific Rim Knowledge Acquisition Workshop, PKAW 2006, Guilin, China, August 7-8, 2006, Revised Selected Papers. Springer Berlin Heidelberg, Berlin, Heidelberg (2006) 25–39

[120] van Lamsweerde, A.: Goal-oriented requirements engineering: a guided tour. In: Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on. (2001) 249–262

[121] Ponnalagu, K., Ghose, A., Narendra, N.C., Dam, H.K. In: Discovering and Categorizing Goal Alignments from Mined Process Variants. Springer International Publishing, Cham (2015) 144–157

[122] Ghose, A.K., Narendra, N.C., Ponnalagu, K., Panda, A., Gohad, A.: Goal-Driven Business Process Derivation. In: Service-Oriented Computing: 9th International Conference, ICSOC 2011, Paphos, Cyprus, December 5-8, 2011 Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2011) 467–476

[123] Ponnalagu, K., Ghose, A., Narendra, N.C., Dam, H.K. In: Goal-Aligned Categorization of Instance Variants in Knowledge-Intensive Processes. Springer International Publishing, Cham (2015) 350–364

[124] Schwitter, R., Fuchs, N.E.: Attempto - from specifications in controlled natural language towards executable specifications. CoRR **cmp-lg/9603004** (1996)

[125] Kolodner, J.: Case-based reasoning. Morgan Kaufmann (2014)

[126] Müller, R., Greiner, U., Rahm, E.: Agentwork: a workflow system supporting rule-based workflow adaptation. Data & Knowledge Engineering **51** (2004) 223–256

[127] Foley, J., Mcmath, C.: Dynamic process visualization. Computer Graphics and Applications, IEEE **6** (1986) 16–25

[128] Kriglstein, S., Rinderle-Ma, S.: Change visualization in business processes - requirements analysis. In: International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP?IVAPP), Rome, Italy (2012)

[129] Kabicher, S., Kriglstein, S., Rinderle-Ma, S.: Visual Change Tracking for Business Process Models. In: Conceptual Modeling – ER 2011: 30th International Conference, ER 2011, Brussels, Belgium, October 31 - November 3, 2011. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2011) 504–513

[130] Gall, M., Wallner, G., Kriglstein, S., Rinderle-Ma, S.: Differencegraph - a prom plugin for calculating and visualizing differences between processes. In: BPM'15 Demo Track, 13th International Conference on Business Process Management. BPM Demo Session 2015 (2015) 65–69

[131] Gall, M., Wallner, G., Kriglstein, S., Rinderle-Ma, S.: A study of different visualizations for visualizing differences in process models. In: Advances in

Conceptual Modeling. Volume 9382 of Lecture Notes in Computer Science., Springer International Publishing (2015) 99–108

[132] Eades, P., Lai, W., Misue, K., Sugiyama, K.: Preserving the mental map of a diagram. Technical report, Technical Report IIAS-RR-91-16E, Fujitsu Laboratories (1991)

[133] Beck, F., Burch, M., Diehl, S.: Towards an aesthetic dimensions framework for dynamic graph visualisations. In: Information Visualisation, 2009 13th International Conference, IEEE (2009) 592–597

[134] Paulisch, F.N., Tichy, W.F.: Edge: An extendible graph editor. Software: Practice and Experience **20** (1990)

[135] Kriglstein, S., Wallner, G., Rinderle-Ma, S.: A visualization approach for difference analysis of process models and instance traffic. In Daniel, F., Wang, J., Weber, B., eds.: Business Process Management, Berlin, Heidelberg, Springer Berlin Heidelberg (2013) 219–226

[136] Kriglstein, S., Rinderle-Ma, S.: A visualization concept for high-level comparison of process model versions. In La Rosa, M., Soffer, P., eds.: Business Process Management Workshops, Berlin, Heidelberg, Springer Berlin Heidelberg (2013) 465–476

[137] Knuplesch, D., Reichert, M., Kumar, A.: Towards visually monitoring multiple perspectives of business process compliance. In: CAiSE Forum 2015. Number 1367 in CEUR Workshop Proceedings, ceur-ws.org (2015) 41–48

[138] Knuplesch, D., Reichert, M., Kumar, A.: Visually monitoring multiple perspectives of business process compliance. In: International Conference on Business Process Management, Springer (2015) 263–279

[139] Brown, R., Recker, J., West, S.: Using virtual worlds for collaborative business process modeling. Business Process Management Journal **17** (2011) 546–564

[140] Harman, J., Brown, R., Johnson, D., Rinderle-Ma, S., Kannengiesser, U.: Augmenting process elicitation with visual priming: An empirical exploration of user behaviour and modelling outcomes. Information Systems (2016)

[141] Harman, J., Brown, R., Johnson, D., Rinderle-Ma, S., Kannengiesser, U.: Virtual Business Role-Play: Leveraging Familiar Environments to Prime Stakeholder Memory During Process Elicitation. In: Advanced Information

Systems Engineering: 27th International Conference, CAiSE 2015, Stockholm, Sweden, June 8-12, 2015, Proceedings. Springer International Publishing, Cham (2015) 166–180

[142] Wallner, G., Kriglstein, S.: Design and evaluation of the educational game dogeometry: A case study. In: Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology. ACE '11, New York, NY, USA, ACM (2011) 14:1–14:8

[143] Wallner, G., Kriglstein, S.: Dogeometry: Teaching geometry through play. In: Proceedings of the 4th International Conference on Fun and Games. FnG '12, New York, NY, USA, ACM (2012) 11–18

[144] Moreno-Ger, P., Burgos, D., Martínez-Ortiz, I., Sierra, J.L., Fernández-Manjón, B.: Educational game design for online education. Computers in Human Behavior **24** (2008) 2530 – 2540

[145] Wallner, G., Kriglstein, S.: A spatiotemporal visualization approach for the analysis of gameplay data. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '12, New York, NY, USA, ACM (2012) 1115–1124

[146] Ontañón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., Preuss, M.: A survey of real-time strategy game ai research and competition in starcraft. Computational Intelligence and AI in Games **5** (2013) 293–311

[147] Hildebrandt, T., Kriglstein, S., Rinderle-Ma, S.: Beyond visualization: On using sonification methods to make business processes more accessible to users. In: 18th International Conference on Auditory Display (ICAD 2012), Atlanta, Georgia Institute of Technology (2012) 248–249