ulm university universität uulm

Ulm University | 89069 Ulm | Germany

Faculty of Engineering and Computer Science
Institute of Databases and Information Systems

Design and Implementation of Task Management Lifecycle Concepts based on Process Mining

Master's Thesis at Ulm University

Submitted by:
Florian Beuter
florian.beuter@uni-ulm.de

Reviewer:
Prof. Dr. Manfred Reichert
Prof. Dr. Franz Schweiggert

Supervisor:
Nicolas Mundbrod

2015

Version June 7, 2015

Satz: PDF-LaTeX 2_ε

Abstract

In a globalized world, knowledge work and especially Knowledge-intensive Business Processes (KiBPs) become increasingly important in highly developed countries. As a consequence, knowledge workers increasingly require an appropriate system support. Due to the more complex nature and the different characteristics of KiBPs, the Business Process Management (BPM) approach established to support traditional business processes, cannot be applied to KiBPs in the same way. As knowledge workers often rely on paper-based task lists (e.g. checklists, to-do lists) to collaboratively manage their work, a system supporting KiBPs should provide digital task lists based on a lifecycle to achieve a sustainable support.

This thesis discusses new concepts to enable an improved KiBP lifecycle support for task management through the application of process mining techniques. The KiBP lifecycle features the definition of so called collaboration templates, the instantiation of these templates to collaboration instances at run time and the evaluation of collaboration records. In particular, collaboration records are leveraged to automatically derive appropriate templates and to optimize existing templates.

In this context, an optimization approach for task list templates is proposed that incorporates the most frequently applied changes into the corresponding template using a change mining technique. Furthermore, an approach to automatically generate a task list template based on the records of comparable, completed task list instances through the application of a cluster mining algorithm is proposed as well. Additionally, the issue of providing knowledge workers with valuable task recommendations at run time is discussed and an approach addressing this problem is presented. Finally, selected excerpts of the implementation are presented to demonstrate the feasibility of the proposed approaches as a proof-of-concept prototype.

Contents

1 Introduction

Nowadays, companies are facing various challenges to keep and sustain their competitive advantages. In a globalized world, there is more pressure than ever for businesses to act timely, to make the most of resources and to cut costs [3]. While the steady progress in technology is creating a wide range of opportunities for both established companies and new competitors, it is also a driving force for continuously increasing business efficiency in turn. Processes, which had been accomplished manually and paper-based in the past, became supported by information systems managing data digitally and, thereby, significantly improving the availability of information as well.

Through the introduction of BPM, companies are able to further boost effectiveness by systematically managing and standardizing their business processes [20]. This is illustrated by the BPM lifecycle [4] shown in Figure 1.1. Initially, existing processes are analyzed and optimized by domain experts. In particular, the activities, which are part of the business process, have to be identified as well as their order, and the decisions to be made during the course of the business process. These insights are then leveraged to specify a process template as blueprint for a certain kind of business process. Whenever a business process needs to be started and performed, the corresponding process template is instantiated to automatically generate an individual process instance based on this template. This way, business processes in a company become transparent as they are explicitly specified and managed based on a lifecycle. In this context, process instances are monitored to keep track of performance factors like the duration of instances and their resource usage.

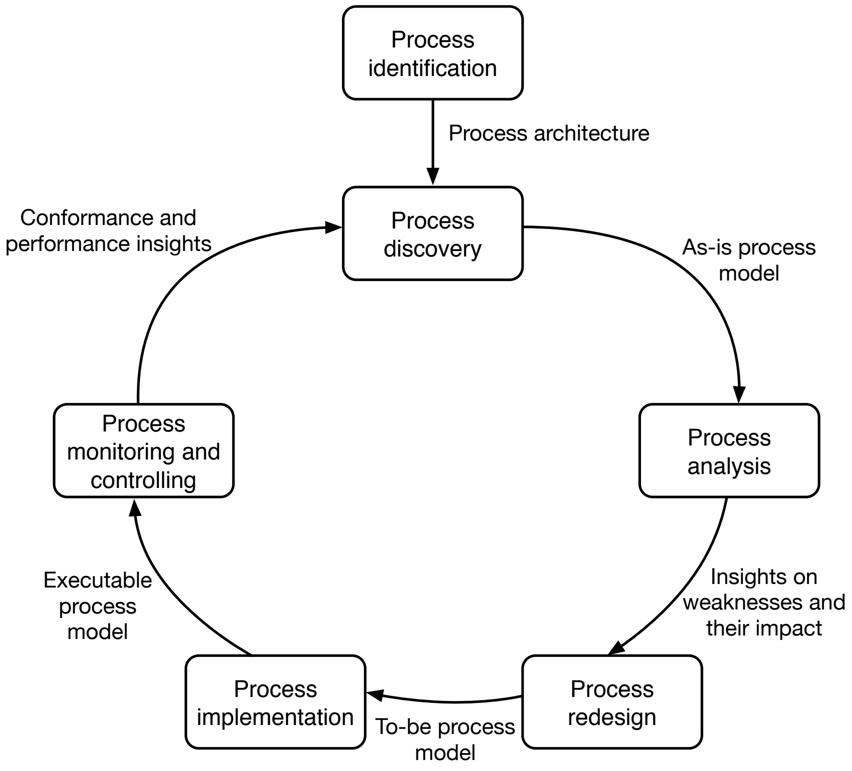

Figure 1.1: BPM Lifecycle

While the approach of BPM works well for supporting and automating predictable, routine processes by standardization, there is no comparable support for KiBPs that have become the prevalent type of work in highly developed countries [1]. Due to their challenging characteristics it is substantially difficult to provide a comparable, adequate support for this special kind of processes. However, to increase the productivity of knowledge workers it is necessary to provide them with a dedicated support, taking the characteristics of KiBPs into account.

1.1 Problem Statement

In particular, KiBPs are *emergent*, *non-predictable*, *knowledge-creating*, and *goal-oriented* [14]. Especially the former three attributes contrast with routine business processes and underline the challenges for an information system that is able to support KiBPs. The traditional approach of BPM proposing the creation of process templates at design time and instantiating the templates at run time, is not feasible for KiBPs [15]. Instead, the alternating planning and work phases involved in KiBPs have to be adequately taken into account to support knowledge workers effectively.
In general, knowledge workers use their skills and expertise to accomplish their work and collaborate to achieve their common goals. However, they often still rely on paper-based task lists to manage the coordination as well as to plan their own work. In particular, these task lists are either used for prospective planning of work (i.e. to-do lists) or for retrospective evaluations of accomplished work (i.e. checklists). Naturally, the availability of paper-based artifacts is limited and it thus hinders the coordination among knowledge workers involved in KiBPs. Once knowledge workers completed their work, task lists are usually disposed of and, hence, there is no analysis of task lists taking place. However, if completed task lists were analyzed, they could be optimized for future uses and, thereby, the productivity of knowledge workers could be increased by relieving them from redundant work. Furthermore, it is difficult for multiple knowledge workers to rely on the same task list as only one person can have the task list at a time, unless they use multiple copies of the same task list. The former hampers the progress made by knowledge workers, due to the limited access to the task list, while the latter imposes synchronization problems between knowledge workers. Additionally, paper-based task lists do not allow for the enforcement of access restrictions to task lists or parts thereof in contrast to digital task lists. Therefore, it is desirable to have a dedicated information

system in place taking care of a systematic task management and providing knowledge workers with the most recent state of the task lists they are involved in.

1.2 Contribution

As a result, a powerful lifecycle approach is necessary to provide support to knowledge workers in a sustainable and systematic manner. In the context of the proCollab[1] research project [18], aiming at the provision of a holistic support for KiBPs based on task list management, this thesis deals with the design and implementation of lifecycle concepts. Especially, the lifecycle approach has to integrate the concept of alternating planning and work phases by allowing for flexible changes on task lists at run time. proCollab already features support to design templates for both KiBPs and integrated task lists and to instantiate them at run time. However, the records of completed KiBPs must be made available to let them be evaluated by both knowledge workers as well as proCollab itself. The main goal is to reduce knowledge workers time spent for planning and, thereby, increasing their productivity.
Due to the *emergent* nature of KiBPs resulting from constant *uncertainty*, it is particularly difficult to provide knowledge workers with appropriate process templates and the corresponding task list templates. Therefore, this thesis proposes concepts for the analysis of task list records as knowledge workers typically use task lists (e.g. checklist, to-do lists) to organize their work. To reliably provide knowledge workers with the most suitable task list templates, proCollab is to be enhanced to automatically optimize and even create task list templates based on task list records. In particular, the lifecycle approach to achieve this goal is twofold: On one side, proCollab should derive a task list template based on a set of comparable task list records. This way, knowledge workers may use the generated task list template in future and don't have to start planning work from scratch over and over again. On the other side, KiBPs and their associated task lists are also subject to change over time. Thus, knowledge workers may still rely on a certain task list template, but they apply changes to the derived task list instances for various reasons. This consequently raises the question whether there are frequently applied

[1]**Pro**cess-Aware Support for **Collab**orative Knowledge Workers

changes to the task list instances indicating that the template itself should be adjusted accordingly. Therefore, proCollab should continuously analyze task list records and allow for a controlled evolution of task list templates by incorporating the most frequently made changes into the templates. Again, this reduces knowledge workers time spent for planning, since they do not need to adapt task list instances in the same way every time. In addition, knowledge workers also should be provided with system support at run time. The alternating planning and work phases are a characteristic of KiBPs implying that users should be provided with recommendations regarding relevant future tasks. This is another aspect where records of successfully completed task list instances constitute a valuable basis to improve future instances based on the same task list template. By offering valuable task recommendations at run time, when knowledge workers plan upcoming tasks, the planning phase can be further reduced enabling knowledge workers to focus on their actual work.

1.3 Outline

The remainder of this thesis is structured as follows: Chapter 2 introduces the fundamental concepts this thesis is based on. Subsequently, Chapter 3 presents an approach to optimize task list templates through the application of a change mining algorithm. Chapter 4 describes an approach to derive a task list template from a set of task list records using a cluster mining technique. Chapter 5 elaborates on the problem of providing appropriate task recommendations to knowledge workers at run time and proposes a basic concept to solve particular aspects of this issue. Chapter 6 presents selected excerpts of the implementation of the approaches described in Chapters 3 and 4. Finally, Chapter 7 concludes this thesis with a summary of the provided task management lifecycle concepts and an outlook on future work.

2 Fundamentals

This chapter presents the essential fundamentals this thesis is based on and explains the frequently used terminology. Section 2.1 introduces business processes whereas Section 2.2 discusses KiBPs to highlight the differences between these process types. Subsequently, the proCollab research project is described in Section 2.3, followed by the discussion of change operations in Section 2.4. Finally, Section 2.5 concludes this chapter with the introduction of process mining.

2.1 Business Processes

While there are many definitions for the term *business process*, this thesis relies on the one provided in [33] defining a business process as follows:

Definition 2.1. *A* ***business process*** *consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations.*

Furthermore, the term *business process management system* (BPMS) is defined in [33] as follows:

Definition 2.2. *A* ***business process management system*** *is a generic software system that is driven by explicit process representations to coordinate the enactment of business processes.*

In order to improve the management and the support of the various business processes in a company, a Business Process Management System (BPMS) can help by making typical business processes available as process templates stored in a process repository. Initially, a process must be analyzed to define an appropriate *process template*, which is a composition of activities, events, and decisions. Once a process template is specified, future occurences of this business process can be assisted by the BPMS generating corresponding *process instances* based on the template. Since the BPMS is now aware of the activities to be executed, their order and the assigned users, it is able to support the business processes accordingly. Thereby, a BPMS may reduce time delays, costs, and error rates.
Figure 2.1 shows an example for a business process using the Business Process Model and Notation (BPMN) notation [6]. The process model depicts how a typical order process in a company may be defined and executed. Triggered by the receipt of an order, it is checked whether all ordered goods are in stock. If the listed goods are not available in total, the missing goods have to be reordered before the process can proceed. For the other case that all goods are in stock, no further action is required. Both cases are then joined and followed by the activities for sending the invoice and shipping the goods, which may be executed in parallel. Finally, the process is concluded after the receipt of the payment.

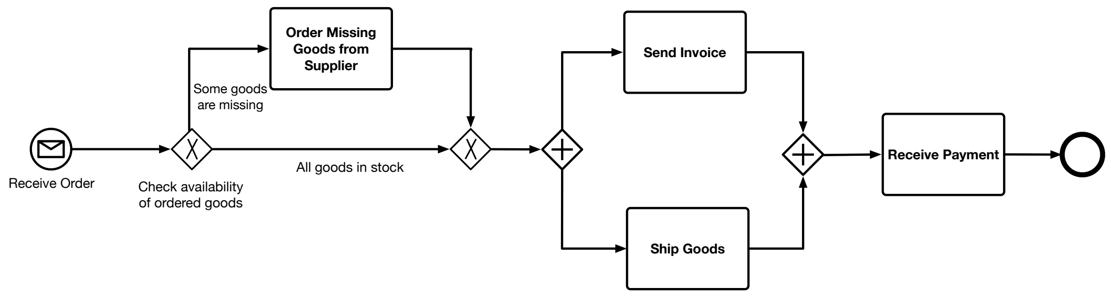

Figure 2.1: An Order Process in BPMN

2.2 Knowledge-intensive Business Processes

This thesis uses the following definition of the term *knowledge-intensive business process* (KiBP) provided in [24].

Definition 2.3. *Knowledge-intensive processes (KiBPs) are processes whose conduct and execution are heavily dependent on knowledge workers performing various interconnected knowledge intensive decision making tasks. KiBPs are genuinely knowledge, information and data centric and require substantial flexibility at design and run time.*

While extensive research has been done to provide support for business processes through BPMSs, there is still a lack of comparable assistance for KiBPs. This is due to the different characteristics of standard business processes and KiBPs [14]. In particular, KiBPs are characterized as *non-predictable*, *emergent*, *goal-oriented*, and *knowledge-creating*. Traditional business processes comprise mainly routine work (e.g. production processes) that can be standardized in a process template by a domain expert. Therefore, it is usually well predictable which activities have to be done as well as their order. By contrast, KiBPs are conducted by knowledge workers jointly utilizing their skills and expertise. In general, a high degree of uncertainty is involved in these processes. This fact generates a need for continuous adjustment of the process, due to the alternating planning and work phases. Standard business processes also require flexibility at run time. However, this is rather the case for exceptional situations requiring deviations from the prescribed process model. Furthermore, KiBPs can not be characterized through a fixed outcome, but rather through a common goal. This common goal is typically further divided into smaller, more manageable sub-goals (also called milestones) by the involved knowledge workers, ultimately leading to the fulfillment of the main goal. To achieve their next goals, knowledge workers have to define the next tasks to be performed. This means that KiBPs are emergent, because knowledge workers have to continuously evaluate the current state of the process to choose those activities bringing them closer to their common (sub-)goal. Another important aspect of KiBPs is collaboration between knowledge workers. Usually there are multiple knowledge workers, potentially from different fields, participating in a KiBP. They work together and communicate with each other to reach their common goal effectively. Another contrast to standard business processes is that KiBPs are knowledge creating, as the involved knowledge workers gain more experience during the course of the process. Examples for KiBPs are conducting research, patient treatment processes in hospitals, and criminal investigations.

2.3 proCollab

In the following, the proCollab project and its prototype are introduced briefly in Section 2.3.1. Subsequently, the KiBP lifecycle model employed in the proCollab project is explained in Section 2.3.2 and Section 2.3.3 concludes with the main entities of proCollab.

2.3.1 proCollab Project

The proCollab research project at Ulm University aims at developing a holistical support for collaborative knowledge workers and their KiBPs. Therefore, the proCollab approach proposes the systematic task management support in the scope of KiBPs. Task lists are artifacts frequently used by knowledge workers to keep track of their progress. Common examples of task lists are to-do lists, which are used to plan work that needs to be done in future, and checklists, which are used in a retrospective manner to evaluate work results. Obviously, paper based task lists lead to media breaks and their availability is

limited. In particular, only one person may access a task list at one time. Further, if there are multiple versions for different people working on the same task list, there are consistency and coordination problems. To address this problem, proCollab supports digital task lists as an integral part of a KiBP and asserts that all involved knowledge workers may access the current state of their task lists at any time. Adapting to the goal-oriented nature of KiBPs, proCollab enables users to specify the context of the KiBP and the associated task lists to achieve this goal.
The proCollab approach has been implemented in a prototype – the proCollab prototype. At the core of the prototype is the server component managing and storing the system's data (e.g. processes, task lists, and users). Additionally, the proCollab prototype features mobile clients for smartphones and tablets providing a maximum of flexibility and a web application, which administrators can manage user accounts and task list templates with.

2.3.2 proCollab Lifecycle

The proCollab approach employs a lifecycle model for systematically supporting KiBPs that consists of four different phases as shown in Figure 2.2 [14]. The single phases of this lifecycle are described in the following.

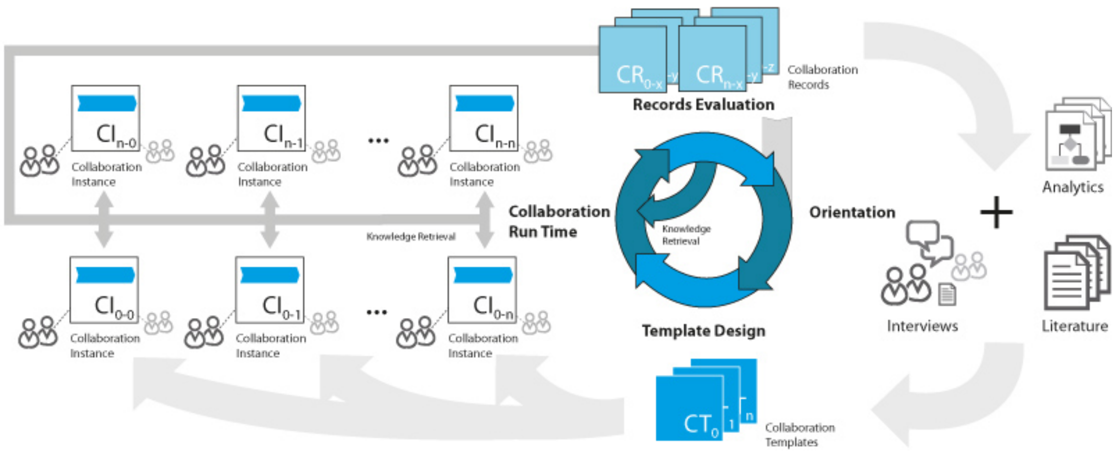

Figure 2.2: The KiBP Lifecycle Model employed by proCollab [14]

Orientation Phase

Starting with the *Orientation Phase*, the context of the collaboration is defined, including the identification of required resources, both human and informational as well as the type of collaboration. The result of this phase is based on interviews with the involved knowledge workers, literature on the subject and past collaborations of the same kind.

Template Design Phase

In the *Template Design Phase*, the findings of the previous phase are used to define *collaboration templates*. Each of these templates may contain a group of tasks and feature a common goal for the participating knowledge workers. Since the main purpose for creating templates is to enable their reuse, their design should be generic to ensure their suitability for similar collaborations.

Collaboration Run Time Phase

In the *Collaboration Run Time Phase* a template is instantiated and, if necessary, adjusted for the specific project or case. While working on their tasks to achieve their common goal, knowledge workers may also access the information of past collaborations based on the same template – so called collaboration records. This way, they can benefit from the experience stored in the records of finalized instances.

Records Evaluation Phase

The lifecycle model's final phase – the *Records Evaluation Phase* – comprises the analysis of collaboration records. To provide knowledge workers with better templates in future, existing collaboration templates should be optimized in respect to the records. For example, an improved template should not contain any tasks that remained unused or that were deleted. Instead, an improved template should incorporate tasks that were not included in the template, but have been added frequently at collaboration run time. Hence, the Records Evaluation Phase provides a feedback loop to the Collaboration Run Time Phase through controlled evolution of existing templates for future collaborations.

2.3.3 proCollab Entities

To directly support the KiBP lifecycle, the proCollab meta model consists of *processes*, *task trees*, and *tasks* (from top to bottom). For every entity in the meta model, there are

templates enabling the reuse of existing components, instances allowing for the usage at run time, and records archiving completed instances.
The entities are shown in Figure 2.3 and defined in the following.

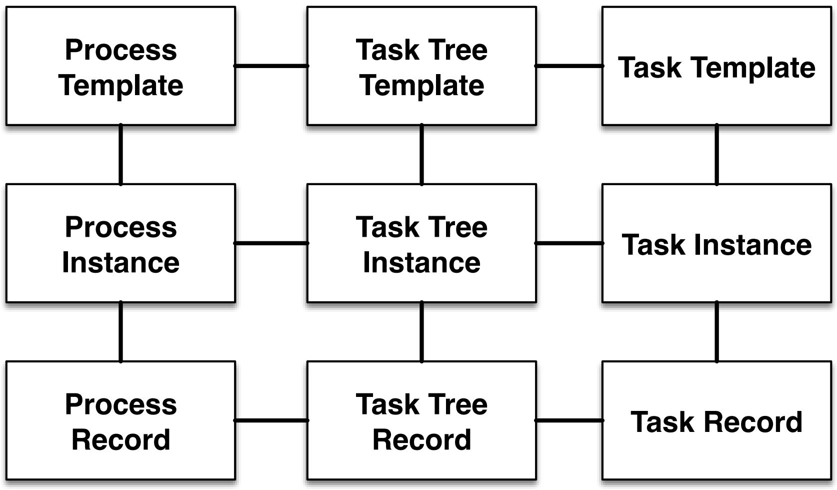

Figure 2.3: Overview of the proCollab Entities

In the following, the templates of processes, task trees and tasks are defined.

Definition 2.4. *A **Process Template (PT)** contains the context information for a certain type of collaboration. It includes the goal of the collaboration, the roles of involved knowledge workers, and corresponding task tree templates. Once a PT has been developed, knowledge workers, who are planning to start a project or a case, can search for this template and instantiate it to get started quickly.*

Definition 2.5. *A **Task Tree Template (TTT)** contains a set of task templates and subordinated TTTs. It can be used for prospective planning as to-do list (i.e. tasks that have to be done) or retrospective evaluations as checklist (i.e. to evaluate work results). TTTs contained in a PT are automatically instantiated as soon as the PT is instantiated. Alternatively, a TTT can be instantiated in the context of an existing Process Instance (PI).*

Definition 2.6. *A **Task Template (TT)** contains information about the task's name, a detailed description, the role of the knowledge worker performing it, and the expected* *duration. A TT also may be part of multiple TTTs leading to the fact that an update on a TT will automatically affect all linked TTTs.*

Subsequently, the instances of processes, task trees and tasks are defined.

Definition 2.7. *A* ***Process Instance (PI)*** *usually refers to a certain project or case and is either based on a PT or created as an individual instance. It contains a list of all involved knowledge workers and their roles as well as the associated Task Tree Instances (TTIs).*

Definition 2.8. *A* ***Task Tree Instance (TTI)*** *can be created individually or instantiated from an existing TTT. Every TTI consists of a set of entries that are either subordinated TTIs or Task Instances (TIs). The TTI also contains additional information about its creator, the involved knowledge workers and the PI it belongs to. For every TTI, an execution log and a change log are maintained. While the execution log covers all state transitions that occur until the work on this instance is finished, the change log contains all change operations applied to this instance.*

Definition 2.9. *A* ***Task Instance (TI)*** *belongs to a (parent) TTI and may either be instantiated from a TT or be individually defined from scratch. It also has an execution log documenting all state transitions and a change log covering all applied change operations.*

Finally, the records of processes, task trees and tasks are defined.

Definition 2.10. *A* ***Process Record (PR)*** *contains a completed PI. It contains its own execution and change log and the logs of all task tree records belonging to this process.*

Definition 2.11. *A* ***Task Tree Record (TTR)*** *contains a completed TTI along with the respective execution and change logs.*

Definition 2.12. *A* ***Task Record (TR)*** *refers to a completed TI and includes the execution and change log of this instance. The change log in particular lists all updates (e.g. change of the description or asssignment to another person) performed on the respective completed TI.*

Since a task tree is a data structure to represent task lists like checklists and to-do lists, Figure 2.4 shows two equivalent notations of the same task list. While the task tree representation highlights the structure of the underlying list, the more intuitive task list representation, which is common for knowledge workers in practice, will be used to illustrate the examples in the following.

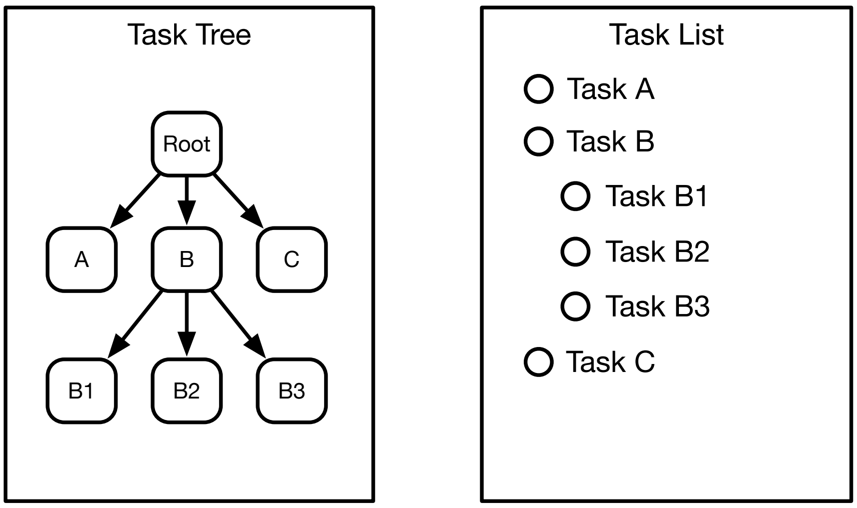

Figure 2.4: Equivalent Task Tree and Task List Examples

2.4 Change Operations

Process models prescribe the order of activities, as they are expected to be executed, without taking exceptional situations into account. Therefore, a BPMS must allow for flexible changes to process instances at run time. Naturally, KiBPs need even more flexibility at run time than standard business processes, due to the alternating planning and work phases in KiBPs. All of the applied changes should be recorded in the instance's change log, as a valuable basis for a future analysis and optimization. There are different types of change operations, ranging from simple changes requiring just one action to more complex high-level changes [32]. The latter can be composed of multiple simple change operations, however. The fundamental change operations for TTIs, which will be taken into account closely, are listed in the following:

- Insert a TI or TTI at a given position
- Delete a certain TI or TTI
- Update a certain TI (e.g. modify its description)

In contrast to the former two change operations, it is important to emphasize that the update operation does not change the structure of the TTI.
To increase the usability and the effectiveness of the approach, knowledge workers should be enabled to apply more complex high-level change operations to TTIs. However, it is notable that high-level changes can be expressed as a sequence of basic change operations leading to the same result. The following list contains relevant high-level change operations for TTIs:

- Move a TI or TTI to a different position
- Replace a TI or TTI with another TI or TTI
- Swap the positions of two TIs or TTIs

The example shown in Figure 2.5 illustrates an explicit (a) and an implicit move (b) operation leading to the same result. On the left side, the move operation is directly applied to place *"Task D"* right below *"Task A"*. On the right side, we first delete *"Task D"* from the TTI and then insert it below *"Task A"* with another change operation. As depicted at the bottom of Figure 2.5, the resulting TTI looks exactly the same for both variants.

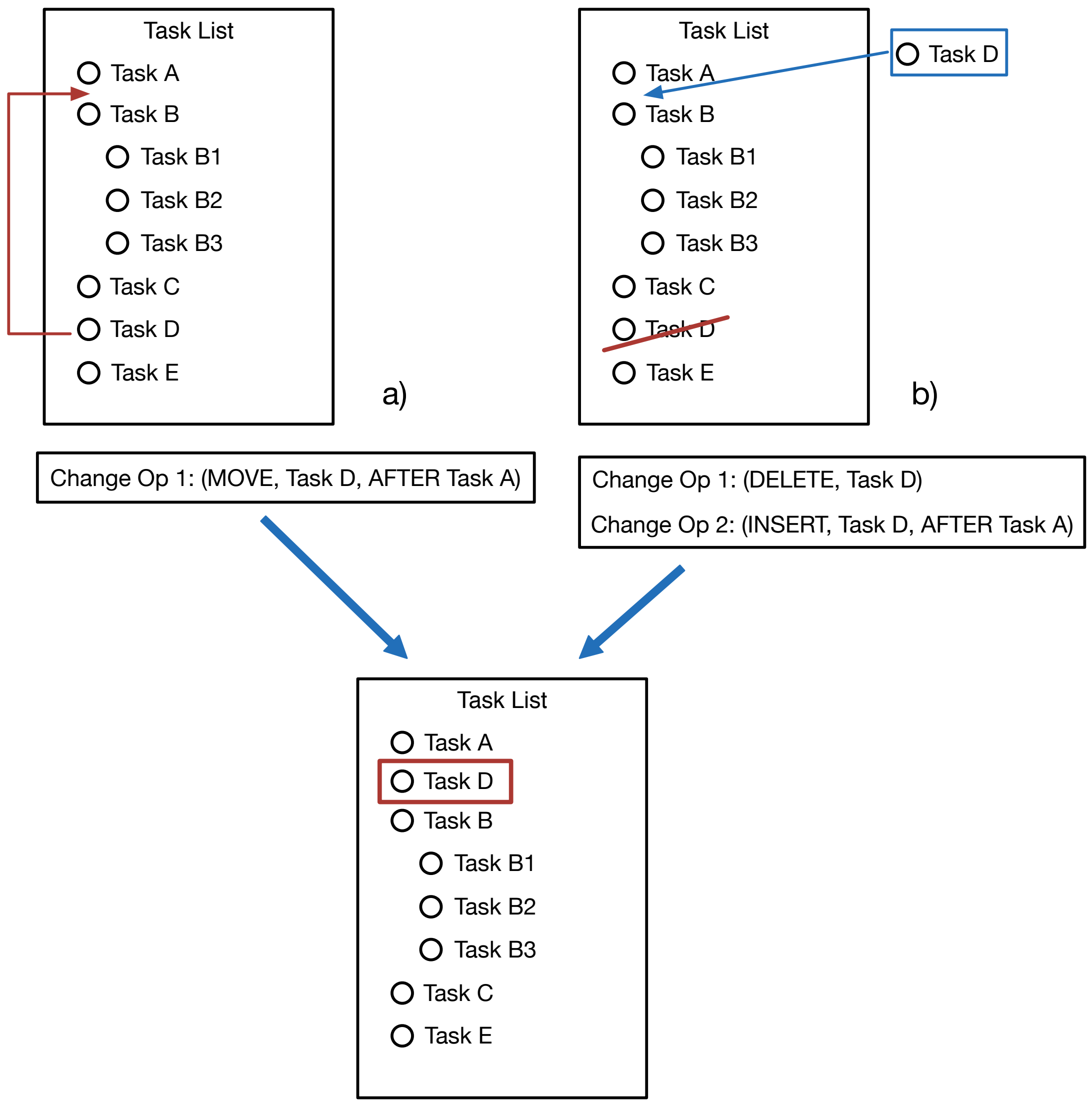

Figure 2.5: Task List Example with Change Log

2.5 Process Mining

Process mining comprises various techniques to gain insights from event logs [26]. Most information systems, including BPMSs, create such log files at run time that may serve as a foundation to apply process mining algorithms. The minimum information, which an event log entry must provide, is a reference to the respective process instance, the name of the executed activity, and a timestamp. Additionally, it is useful to store the originator (i.e. the person who triggered this event) of the event and the state of the activity. Based on this information in an event log, it is possible to reconstruct the order of executed activities. Therefore, events are grouped together for every process instance and sorted by their timestamps. Furthermore, the duration of each referenced activity can be computed by calculating the difference between an activity's begin and end timestamp.
Different process mining approaches can be categorized by the goal they pursue. One of the different goals of process mining is called *process discovery*. Its focus is to mine a process model from a given event log. In cases where a process model can not be defined easily, this technique can be leveraged to automatically derive one. Another goal is conformance checking, which detects deviations between an existing process model and the recorded behavior in the event log. Basically, this is a comparison between the process as-is (event log) and the process how it is supposed to be running (process model). Apart from this, there are several other aspects that can be investigated by applying process mining techniques, e.g., identifying bottlenecks in a process. By fixing such bottlenecks the average duration of a process instance can be significantly decreased.

2.5.1 Change Mining

While process mining extracts information from execution logs, change mining utilizes process mining algorithms to optimize process models based on change logs [8]. Just like execution logs, change logs record events that are related to certain process instances. Change log information includes the type of change operation (e.g. insert or delete), the

subject of the change operation (i.e. a process activity) and, optionally, the location of the change operation in the process instance. The latter is optional as some operations, like "delete”, do not require this necessarily. Additionally, the originator of the change operation and the timestamp are included in the change log. The latter is used to reconstruct the order in which the different change operations were applied to the process instance.
In the same way as with process discovery, a change process can be mined using similar process mining algorithms. The change process illustrates the applied change operations, their order, and their frequency. Based on this change process, different change process variants are discovered. By taking into account the frequency of change operations, it is possible to determine sequences of frequently applied change operations in the mined change process. To optimize a process model, the most frequent changes are applied to the analyzed process model in the same order as they have been observed in the change process. Furthermore, a process model can be adjusted with identified changes to generate an additional process model (i.e. a variant of the analyzed process model) that is more suitable for a certain set of process instances.

2.5.2 Causal Nets

There is a huge variety of process model notations that can be used to represent the process mining results like classic Petri nets [17], Workflow nets [25], Event-driven Process Chains (EPCs) [22] or BPMN [16]. A particularly interesting notation to capture observed behavior in an analyzed event log are *Causal nets (C-nets)* [27]. They were specifically designed to be well suited for the needs of process mining techniques. Like many other process model notations, a C-net is a graph structure where each node stands for an activity and edges are used to connect them with each other. Instead of token-based semantics, the control flow is determined by the input and output bindings of each node. Each binding contains an arbitrary amount of node sets and every such set is an alternative path (i.e. the XOR semantics). A node is activated as soon as one of its input bindings is fulfilled. This means that all of the bound nodes have been executed successfully before this node. Following the execution of a node, an output binding is chosen and all of its bound nodes have to be visited in the later process. More precisely, every node listed in a selected output binding must be visited before executing the end node.

The example C-net in Figure 2.6 illustrates the semantics. For every node, the input bindings are marked with dots at the incoming edges and bindings spanning multiple nodes are connected through an additional arc. Likewise, the output bindings are shown at the outgoing edges of each node. Initially, only the start node is activated and succeeded by either A and B, B and C or A, B and C – this is represented by the three corresponding output bindings. Activity B (highlighted in red) has only one input binding (blue dot) referring to the start node. That means, B can be executed right after the start node and is then followed according to its three output bindings by either just D (yellow dot), just E (red dot), or D and E (green connected dots). Note that this is an example of a C-net employing the OR semantics. The output bindings represent the logical expression of D OR E, in contrast to D XOR E which would exclude the alternative of choosing both elements together. The fact that the C-net is able to use the OR semantics is a very useful aspect for process mining applications as it significantly increases the expressiveness of this notation.

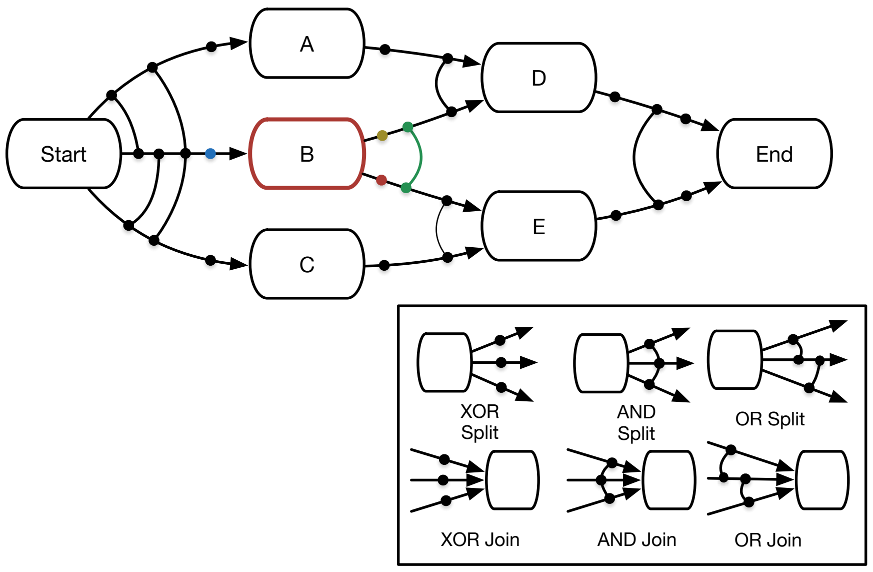

Figure 2.6: A C-net example

3 Optimizing Task Tree Templates with Change Mining

3.1 Problem Statement

Following the KiBP lifecyle (cf. Section 2.3.2), a certain TTT is usually created as the result of the orientation and template design phases. Then, this TTT is instantiated multiple times in the shape of TTIs for appropriate KiBPs during the collaboration run time phase. At run time, change operations (cf. Section 2.4) may be applied to a TTI for various reasons. Among these reasons are that a TTT is too generic (i.e. it does not contain tasks, needed for the TTI at run time), the tasks are not ordered in the way knowledge workers require them, or the TTT contains tasks that are not necessary and, therefore, need to be removed. This problem is difficult to avoid since the characteristics of KiBPs (i.e. *uncertainty*, *emergence*) make it hard to model an appropriate TTT in

advance. But if knowledge workers have to perform redundant tasks by always adjusting a TTI shortly after the instantiation in similar ways, they could be significantly relieved if the most frequent changes were incorporated into the existing TTT. Ideally, such an optimization of TTTs should be performed automatically and continuously to assure that knowledge workers are always provided with the most adequate templates. This also takes the *knowledge-creating* trait of KiBPs into account, in the sense that the knowledge gained through TTRs is reused to improve the corresponding TTT (i.e. the adjustments made to the template don't have to be done over and over again).
In order to provide such automatic optimizations, proCollab provides entities supporting the KiBP lifecycle. For both task trees and tasks, proCollab must be able to instantiate templates as well as to archive completed instances as records. Furthermore, execution and change logs need to be maintained as they can be leveraged for the analysis and optimization of TTTs.

3.2 Illustrative Example

To illustrate the optimization approach proposed in the remainder of this chapter, an example is used and shown in Figure 3.1. The TTT to be optimized is depicted on the left side of the figure and contains a total of ten tasks enumerated from A to J. Based on this TTT are a total of 100 TTIs and the change logs belonging to TTRs that have been applied to these TTIs are shown on the right side of the figure. While 40 TTIs remained unchanged, the other 60 TTIs have been adjusted according to the given change logs. In this example each of the four change logs is contained in 15 TTIs. Note that in practical scenarios change logs usually will be more diverse. However, changes with low frequency values won't be part of the optimized TTT and, hence, their exclusion in this example does not limit the applicability of the proposed optimization approach in general.

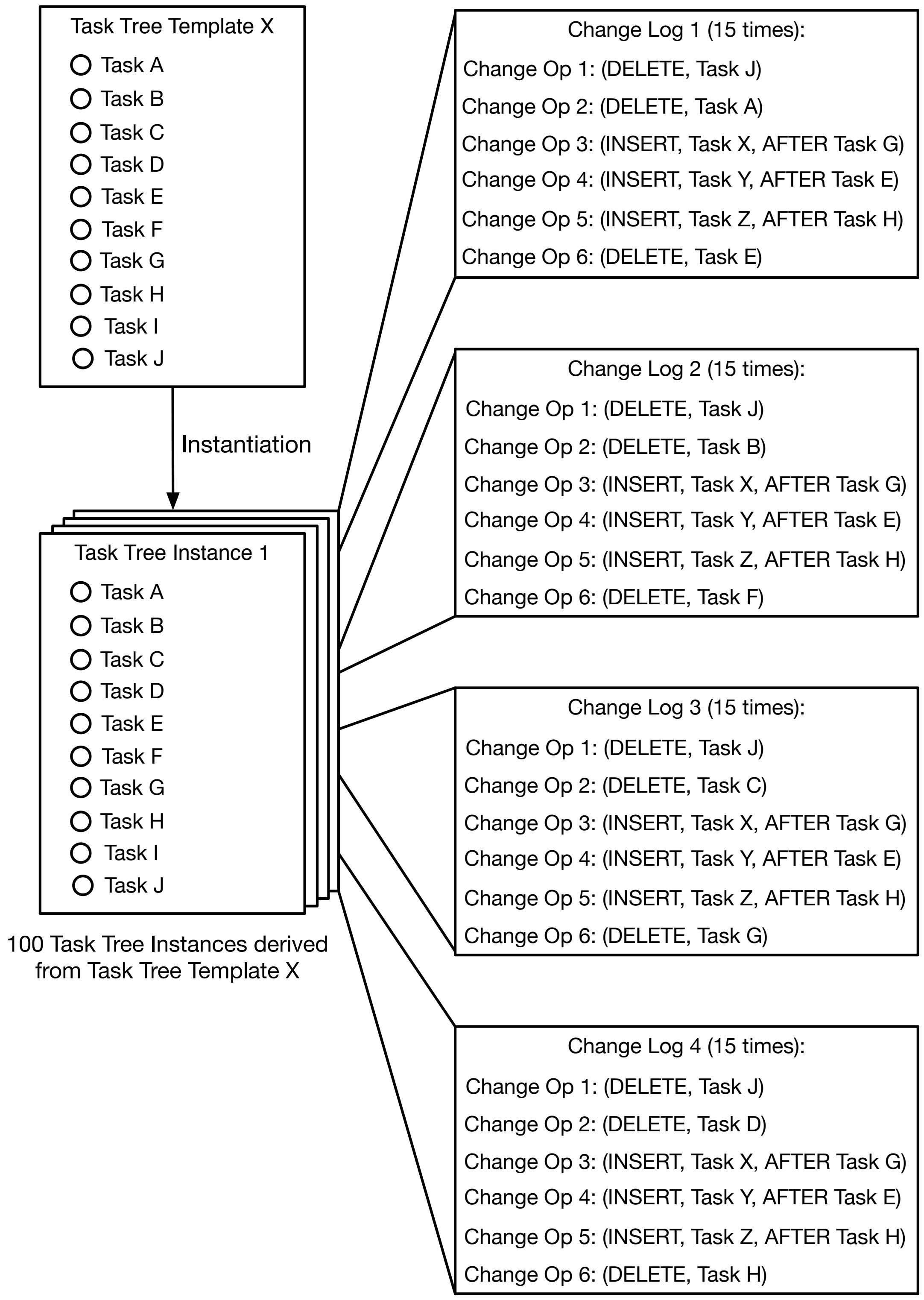
Task Tree Template X
Task A
Task B
Task C
Task D
Task E
Task F
Task G
Task H
Task I
Task J
Instantiation
Task Tree Instance 1
Task A
Task B
Task C
Task D
Task E
Task F
Task G
Task H
Task I
Task J
100 Task Tree Instances derived from Task Tree Template X
Change Log 1 (15 times):
Change Op 1: (DELETE, Task J)
Change Op 2: (DELETE, Task A)
Change Op 3: (INSERT, Task X, AFTER Task G)
Change Op 4: (INSERT, Task Y, AFTER Task E)
Change Op 5: (INSERT, Task Z, AFTER Task H)
Change Op 6: (DELETE, Task E)
Change Log 2 (15 times):
Change Op 1: (DELETE, Task J)
Change Op 2: (DELETE, Task B)
Change Op 3: (INSERT, Task X, AFTER Task G)
Change Op 4: (INSERT, Task Y, AFTER Task E)
Change Op 5: (INSERT, Task Z, AFTER Task H)
Change Op 6: (DELETE, Task F)
Change Log 3 (15 times):
Change Op 1: (DELETE, Task J)
Change Op 2: (DELETE, Task C)
Change Op 3: (INSERT, Task X, AFTER Task G)
Change Op 4: (INSERT, Task Y, AFTER Task E)
Change Op 5: (INSERT, Task Z, AFTER Task H)
Change Op 6: (DELETE, Task G)
Change Log 4 (15 times):
Change Op 1: (DELETE, Task J)
Change Op 2: (DELETE, Task D)
Change Op 3: (INSERT, Task X, AFTER Task G)
Change Op 4: (INSERT, Task Y, AFTER Task E)
Change Op 5: (INSERT, Task Z, AFTER Task H)
Change Op 6: (DELETE, Task H)

Figure 3.1: Illustrative Example

3.3 Optimization Approach

The presented optimization approach consists of three different phases and, thereby, automatically optimizes the given TTT. First, the preparation phase begins with preprocessing change logs so that a mining algorithm can be applied to them later on. The analysis phase utilizes a process mining algorithm to mine the *change process* from the given logs. Based on the change process, the different variants are identified and then leveraged to determine sequences of the most frequent change operations. Finally, the optimization phase adjusts the TTT by incorporating these changes into it.
An alternative to the proposed change mining approach could be the optimization of TTTs based on the direct comparison of completed TTIs. However, this approach would not allow for a controlled evolution by optimizing TTTs as the evaluation of the change logs does. By utilizing change mining, the change operations applied by knowledge workers and the order of these change operations are explicitly taken into account, while the comparison of TTIs is solely based on the final structure of these TTIs. Hence, intermediate results cannot be considered by an approach based on the comparison of TTIs.

3.3.1 Preparations

The first step before the actual analysis can be started is to select the TTRs based on the TTT that should be optimized. For a general optimization of the template, all available TTRs that are related to this TTT can be used. However, the input for the analysis can be limited to a subset of these TTRs to specifically improve the TTT for a certain kind of KiBPs.
Another decision regarding the selection of input data must be made regarding the amount of change log entries to consider. While it is possible to simply perform the analysis on the entire change logs, it may be more reasonable to use only the beginning of the change logs up to a specified time span instead. This would limit the optimization to changes applied shortly after the instantiation of the TTIs, which most likely affect parts of the respective TTT that should be adjusted accordingly.

Obviously, change logs may contain changes that have been done by mistake and were then undone again. To avoid the inclusion of such changes in the analysis, change logs should be cleaned up by removing these changes from the logs.

In this context, note that only insert, delete, and update operations are considered in the following. This does not limit the applicability of the proposed approach as outlined in Section 2.4. Any high-level change operation can be expressed as a sequence of these basic operations. Note that update operations correspond to the respective TT of a TI and not to the TTT containing the TT. Consequently, the optimization of TTs is achieved through the evaluation of change logs of TRs. To increase the comparability of update operations, changes adjusting multiple attributes of a TI at once, should be decomposed into updates of single attributes of a TI.

As a prerequisite for the analysis it is further required to establish a matching between the recorded change operations in the change logs of different TTRs. While removal and update operations can be well compared, the comparison of inserted tasks is particularly challenging. Since insert operations add new TIs to a TTI, a similarity function is necessary to determine which of them ultimately represent the same task. A basic approach towards this problem could use techniques to compare the text descriptions of tasks and match the ones with the highest similarity score. In this context, activity label matching approaches [11] used for business process model matching can be leveraged. For instance, the string edit distance is a simple measure to determine the likeness of two strings. However, the string edit distance solely compares the whole text descriptions syntactically and, thereby, limits the quality of the matching. The comparison can be significantly improved by decomposing the text description into a *bag-of-words* [11] first. Instead of comparing the whole strings, the matching score should be calculated as the number of successfully matched words divided by the total number of words contained in the larger *bag-of-words*. This way, descriptions using the same terms in a different order get more appropriate similarity scores.

To further enhance the quality of the matching more advanced text analysis techniques can be utilized. For example synonyms in text descriptions could be taken into account by querying a lexical database like WordNet [13]. Additionally, other information than the text description could be included in the comparison. Particularly the TTs that TIs were

derived from can be compared, if this information is given. Other comparable information includes the position of the inserted task in the TTIs, the knowledge worker performing the change operation and other changes that have been applied before and after this change.
Regarding the logged change operations, it is notable that change operations are not commutative in general. Therefore, the order in which changes have been applied must be considered when analyzing change logs and, subsequently, optimizing the TTT.

3.3.2 Analysis

During the analysis phase, the change logs are utilized to determine the most frequent changes. This is accomplished based on the following steps, which are explained more detailed in the remainder of this section.

1. Use **change mining** to determine the change process of the given logs
2. Determine the **different variants** in the change process
3. **Identify** the **most frequent changes** in these variants

To illustrate the analysis phase, Figure 3.2 shows the single steps, which are part of the analysis.

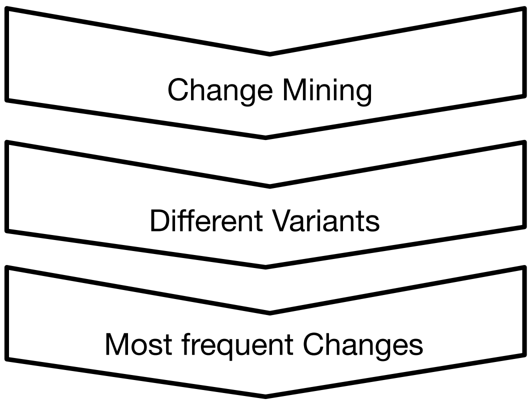

Figure 3.2: Steps of the Analysis Phase

Change Mining

Based on the change mining approach proposed in [8], this thesis uses the multi-phase mining algorithm ([30], [31]) to mine change processes from log files containing the history of change operations for the selected TTRs. The multi-phase mining algorithm takes the log file as input and creates a C-net representing the mined change process. This C-net is built in a two step approach by first creating C-nets for every single trace in the change log (i.e. for every TTR) and then aggregating all of them into one C-net.

Every node in the final C-net represents a change operation found in the log and is annotated with its frequency. In addition to creating the C-net, the miner also fills a map structure that contains the sequences of preceding changes for every change operation. This is needed later to filter out invalid variants as the change process generalizes the behavior observed in the log [28] (i.e. it can allow for traces that were not included in the log).

The resulting C-net for the illustrative example introduced in Section 4.2 is shown in Figure 3.3. The four different change logs are each valid paths through the mined C-net and every change operation is annotated with its respective frequency value. In this context note that there also were 40 TTIs with no applied changes (cf. Section 3.2) that are not depicted in Figure 3.3. However, they are considered when the absolute frequencies are converted to relative frequency values.

Furthermore, the final C-net is an example for *underfitting* as it allows for a total of 16 valid traces, while the change logs only contain four different traces. Therefore, the map of the preceding partial traces created during the mining process allows to identify the valid change process variants later. Since the C-net provides no means to remember the decisions made on previous XOR splits, every choice after *"Insert Task Z"* could be made although only one of them is valid depending on the choice made after *"Delete Task J"*.

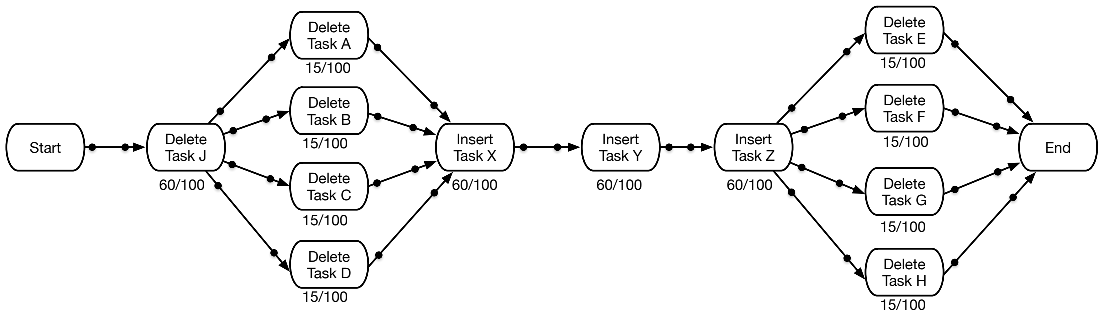

Figure 3.3: Change Process of the Illustrative Example

Determining Variants

The set of change process variants can be determined by traversing the mined C-net in forward direction (i.e. from the start node to the end node). All output bindings of a C-net node represent alternative paths (i.e. exclusive choices) and, therefore, must be considered as branches to additional variants. Beginning with the start node, a new variant is created and stored in a queue for every output binding. The nodes bound by these output bindings have to be inspected in the following. During the iteration over all variants in the queue, a node with fulfilled input bindings is determined for every variant (i.e. the variant's active node). Then, the output bindings of this active node are considered, leading to the consideration of additional variants for every output binding. To avoid the consideration of invalid variants, a check is performed for all active nodes based on the map to verify that there is a corresponding preceding partial trace contained in the change logs. If no such trace exists, the variant is discarded. This process continues until all variants in the queue successfully reached the end node of the C-net.

Figure 3.4 shows the four variants determined by traversing the C-net depicted in Figure 3.3. As mentioned before, this C-net allows for a total of 16 different paths from the start to the end node, but the invalid ones were filtered out using the preceding partial traces map. Since the variants are sequences of change operations, the input and output bindings are not shown in Figure 3.4.

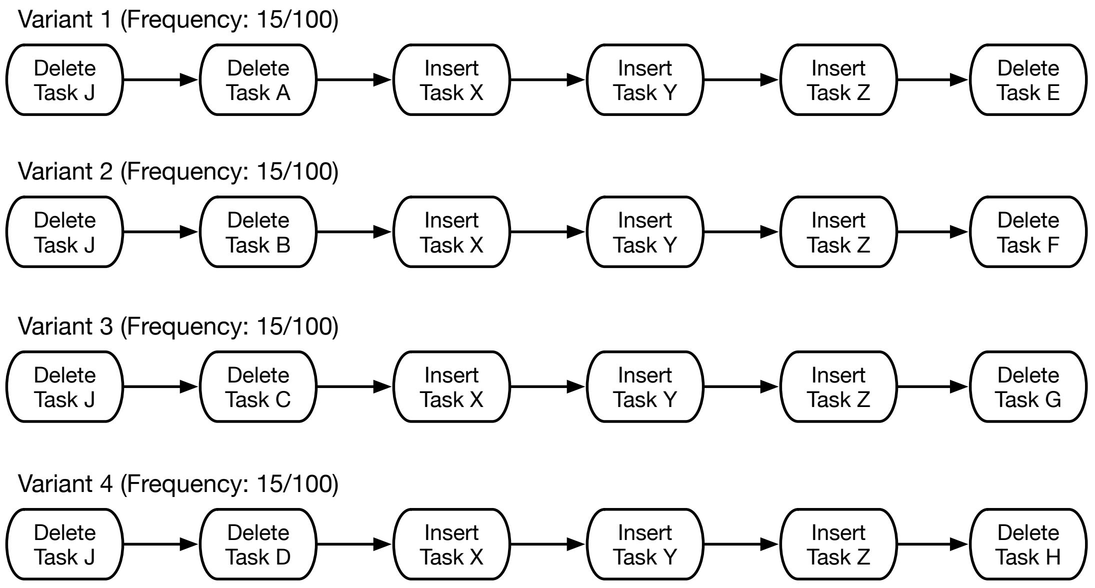

Figure 3.4: Valid Change Process Variants of the Illustrative Example

Identifying Change Blocks

Once the change process variants have been determined, the analysis continues with the search for change blocks, i.e. sequences of frequent changes. As a prerequisite, a map of the most frequent change operations is created, containing all changes with a frequency value higher than a given threshold (e.g. 50%). To generate this map, the frequency values of all change process variants containing a particular change operation are summed up, accordingly.

In a similar way, the change blocks are identified by traversing all variants. For every encountered change operation, a check is performed to verify whether the change operation is one of the most frequent changes or not. Every time such a change operation is found, the algorithm builds a change block of maximum length, beginning at the position of the currently examined change operation. This is accomplished by including the following change operations if their frequency values are high enough. Once the end of a change block has been reached, the change block is stored or its frequency is updated in case the same change block was already found in other variants before.

Figure 3.5 shows the identified change blocks for the variants of Figure 3.4. Assuming a minimum occurence of 50% as a threshold for change blocks, *"Delete Task J"* is the first change block as all of the four variants start with this change operation. Since all variants have a different change operation following on *"Delete Task J"*, this change block cannot be extended. The second change block consists of the three insert operations contained in all four change process variants in the same order.

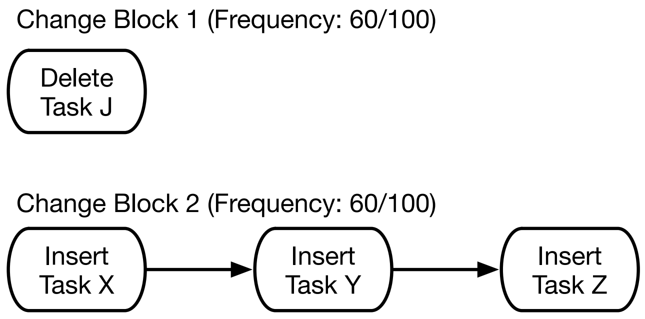
Change Block 1 (Frequency: 60/100)
Delete Task J
Change Block 2 (Frequency: 60/100)
Insert Task X
Insert Task Y
Insert Task Z

Figure 3.5: Identified Change Blocks of the Illustrative Example

3.3.3 Task Tree Template Optimization

In the TTT optimization phase, the results of the analysis phase are used to improve a given TTT. Depending on the goal of the analysis and the provided TTRs, the TTT is either optimized to replace its current version or to introduce one or several specialized TTTs for certain KiBPs. Another aspect influencing the application of change blocks is the type of task list a TTT represents. Since checklists are very detailed and highly structured, fewer changes might be applied to them. By contrast TTTs for to-do lists are rather coarse-grained as it is a normal process that more detailed tasks are added during the course of the action. Therefore, more change operations may be recommended and applied to a TTT for to-do lists. Instead of automatically modifying and introducing TTTs, proCollab also may provide recommendations based on the identified change blocks and let a knowledge worker decide finally.
Note that the application of insert operations to a TTT requires the introduction of new TTs for the inserted TIs. While the necessary information to create an appropriate TT

may be partly inferred from the TIs, this likely needs to be checked by a knowledge worker as well.

Figure 3.6 shows the optimization of the example TTT depicted on the left side. The previously identified change blocks have a frequency value of 60%. Hence, the original TTT is adjusted by applying the change blocks in the same order as they have been applied to the variants. On the right side of the figure, the adjusted TTT is depicted. Accordingly, *"Task J"* was removed from the TTT and the three inserted tasks are highlighted in red.

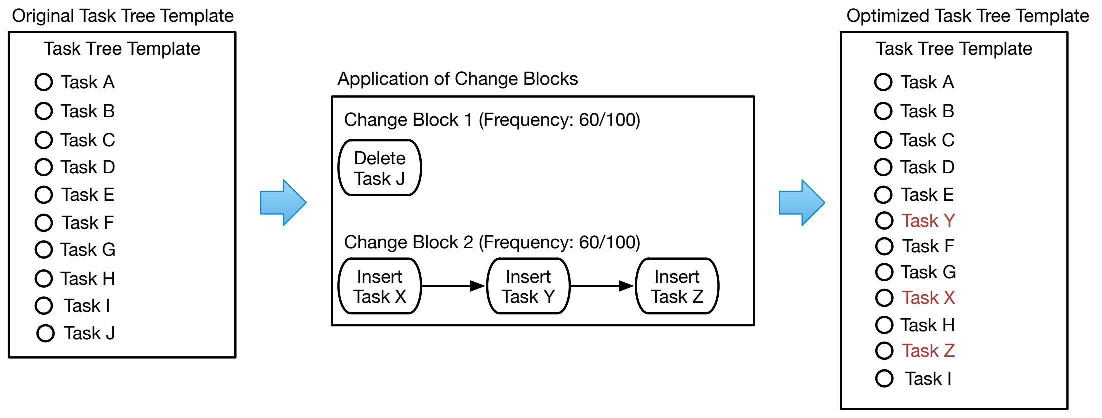

Figure 3.6: Task Tree Template Optimization

3.3.4 Limitations

The presented change mining approach solely relies on change logs of TTRs to optimize TTTs. Naturally, there are other aspects that should also be taken into account when optimizing TTTs.

For instance, tasks should be ordered in the way they are actually being worked on. If this is not the case for a TTT, then tasks should be reordered accordingly. For this purpose, the execution logs of TTRs can be leveraged as they record the beginning of work for all contained tasks. With this information at hand, the average position of every TI can be calculated and used to reorder the respective TTT. Since a complete reordering of all tasks in a TTT might confuse knowledge workers more than it actually helps, threshold values can be used to restrict the number of repositioned tasks to an absolute number. Alternatively, an approach that determines whether tasks were executed earlier or later than their current position in the TTT suggests can be employed. Tasks occuring earlier than their current position in the TTT could be moved one position up in the TTT and vice versa for later occuring tasks.

3.4 Summary

This chapter presents an approach based on change mining to optimize TTTs in three phases. The preparation phase comprises the selection of appropriate TTRs for the analysis and preparing the change logs for the analysis. During the analysis phase, the change logs are leveraged to determine the change process through the application of a multi-phase mining algorithm. Based on this change process, the different change process variants are identified and, in turn, used to determine sequences of the most frequent change operations. Finally, the optimization phase incorporates these sequences of change operations into the TTT in the same order as they have been applied to the change process variants. When knowledge workers instantiate the optimized TTT in future, they don't have to apply these most frequent changes again as the TTT has been automatically adjusted accordingly.

4 Deriving Task Tree Templates through Applying Cluster Mining

4.1 Problem Statement

While Chapter 3 is showing how existing TTTs can be optimized, this chapter presents an approach to automatically derive a common TTT for a group of comparable completed TTIs. A TTT may be manually created by a knowledge worker to enable the reuse of existing knowledge, but the characteristics of KiBPs make it rather difficult to specify an appropriate TTT beforehand. Alternatively, a TTT can be derived automatically from comparable TTRs by exploiting the *knowledge-creating* nature of KiBPs. Since TTRs represent successfully completed TTIs, the inferred template is a promising candidate to be used in KiBPs of the same kind in future. Additionally, deriving TTTs automatically has the advantage that TTTs are created based on facts (i.e. the successfully completed TTIs comprised in TTRs) instead of knowledge workers' expectations.

4.2 Preparations

Since the given input is a set of TTRs the first necessary step is to mine the TTI variants represented by these TTRs.
Some of the techniques introduced in Chapter 3 can be reused to achieve this. Before the application of a mining algorithm to determine the TTI variants is possible, a matching between the TIs in the TTIs must be established. This is necessary as the TTIs have their own TIs and, hence, it is unknown which of these TIs correspond to each other. Otherwise, it would be impossible to determine the frequencies of the same tasks across all TTIs. The matching of TIs is simple if TIs in different TTIs have been based on the same TTs. However, this information cannot be taken for granted and, therefore, a technique to establish a matching between the TIs of different TTIs is required. For example, a mapping can be accomplished using activity label matching approaches as stated in Section 3.3.1. Furthermore, other known information (e.g., the role of the assigned person of a task, the position of the task, etc.) of TIs can be leveraged to create an appropriate matching.
Once the TIs have been compared and matched successfully, the multi-phase mining algorithm can be applied to the execution logs of the TTRs. Then, the TTI variants can be determined by traversing the resulting C-net as shown in Section 3.3.2.

4.3 Illustrative Example

The example TTI variants depicted in Figure 4.1 are used to illustrate the cluster mining approach in the following. There are a total of four different TTI variants annotated with their frequency values. The red box below the TTI variants lists the constraints common to all variants and can be used to verify the result later on. For example, *"Task A"* precedes *"Task D"* in all four TTI variants and, therefore, a common TTT for these variants should also place *"Task A"* as a predecessor of *"Task D"*. The illustrative example also contains hierarchical constraints that are not explicitly listed as they are rather obvious (e.g. *"Task D3"* is a subtask of *"Task D"* in all variants).

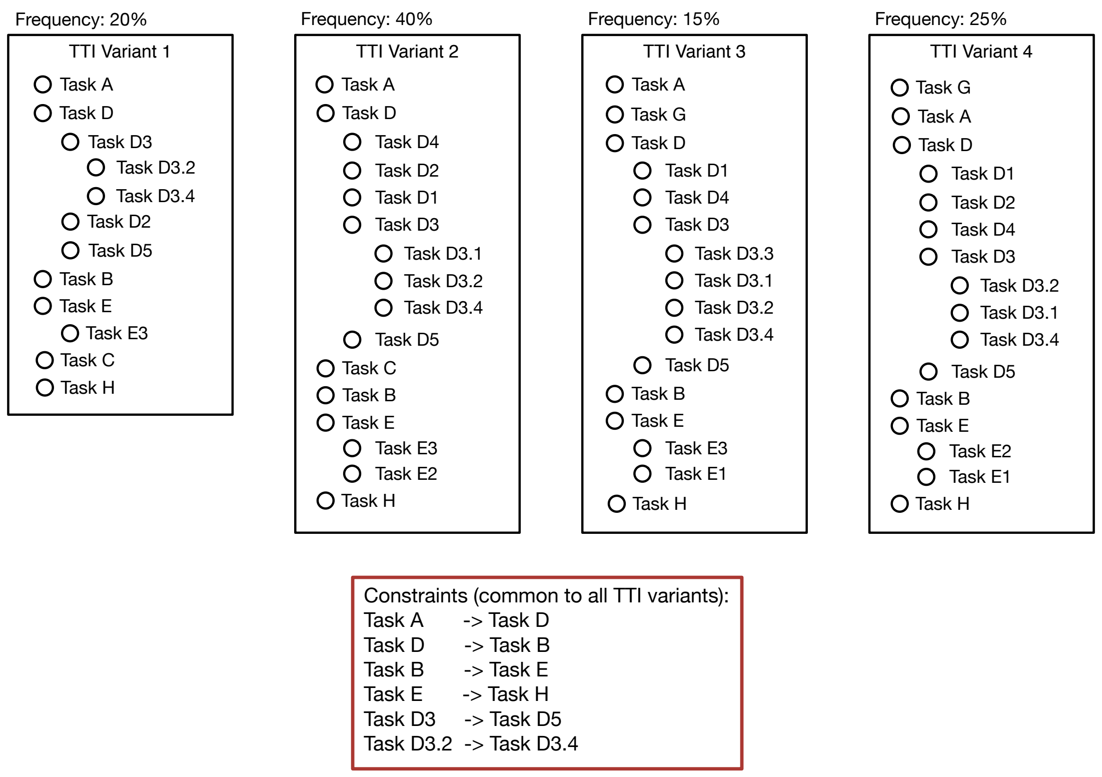

Figure 4.1: Example Variants of Task Tree Instances

4.4 Cluster Mining

The cluster mining algorithm described in the following is based on the MinADEPT algorithm [2] for business process variants and has been adapted to support the different characteristics of TTI variants. Basically the cluster mining approach relies on determining the order relations between all TIs in all TTI variants and iteratively building blocks of tasks with similar relations to the other tasks. This process continues until all TIs have been clustered into one block, which is the resulting TTT.

4.4.1 Analyzing Task Frequencies

The first step towards the creation of a common TTT is the selection of a set of relevant tasks that are considered in the ensuing clustering process. For this purpose, a threshold value is defined and only TIs with a frequency value above this threshold will be part of the TTT.
In the illustrative example this threshold is set to 50%. Figure 4.2 shows that *"Task D3.3"*, *"Task G"*, and *"Task E1"* have frequency values below 50%. Therefore, these TIs are excluded from the cluster mining and the TTT will be limited to the remaining 16 TIs fulfilling the threshold value.

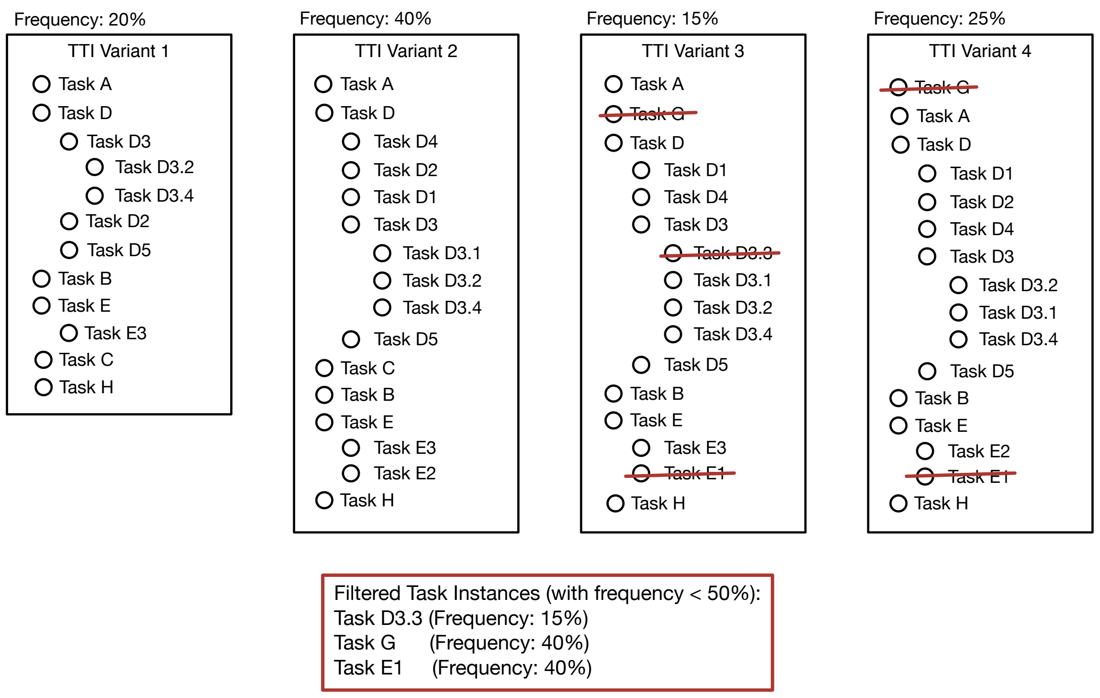

Figure 4.2: Filtering of infrequent Tasks

4.4.2 Converting Variants to Aggregated Order Matrix

The cluster mining algorithm works on an aggregated order matrix and, hence, the variants have to be converted to this representation first. Initially, every TTI variant is represented by its own order matrix. This order matrix contains the transitive relations between every possible pair of TIs in a TTI variant. For two TIs A and B, the relation might be a predecessor/successor relation, i.e. either A is followed by B or vice versa. Further, two TIs can also be part of a hierarchical relation, i.e. either A is a subtask of B or vice versa. If none of these relations holds for two TIs, this is explicitly noted as "no relation". Note that this does not indicate that there is no relation at all between such TIs, but it definitely is none of the previously stated ones. Table 4.1 summarizes the relevant relation types and their encodings in the order matrix.

Relation Code	Relation Type
0	No relation between A and B
1	A precedes B
2	A succeeds B
3	A is a parent task of B
4	A is a subtask of B

Table 4.1: Relations used in Order Matrix

To encode a TTI variant in an order matrix, the relations for every TI to all other TIs have to be analyzed. For the hierarchical relations, this can be done by marking all (transitive) subordinated TIs of a given TI in the order matrix, accordingly. Similarly, the superordinated TIs are identified by moving up in the hierarchy of a TI's parent tasks until the top-level is reached. The predecessor/successor relations only have to be analyzed for TIs with the same superordinated TI. The relations to all remaining TIs can be set to "0" then.

An example of an order matrix is shown in Table 4.2 for TTI variant 1 in Figure 4.1. This order matrix captures the relations of all TIs in this TTI variant to each other. For example, *"Task E3"* is the only subtask of *"Task E"* and, therefore, it has no relations to other TIs. Likewise, all top-level tasks (e.g. *"Task H"*) except *"Task D"* have no relations to the subtasks of *"Task D"*.

	A	B	C	D	D2	D3	D3.2	D3.4	D5	E	E3	H
A	-	1	1	1	0	0	0	0	0	1	0	1
B	2	-	1	2	0	0	0	0	0	1	0	1
C	2	2	-	2	0	0	0	0	0	2	0	1
D	2	1	1	-	3	3	3	3	3	1	0	1
D2	0	0	0	4	-	2	0	0	1	0	0	0
D3	0	0	0	4	1	-	3	3	1	0	0	0
D3.2	0	0	0	4	0	4	-	1	0	0	0	0
D3.4	0	0	0	4	0	4	2	-	0	0	0	0
D5	0	0	0	4	2	2	0	0	-	0	0	0
E	2	2	1	2	0	0	0	0	0	-	3	1
E3	0	0	0	0	0	0	0	0	0	4	-	0
H	2	2	2	2	0	0	0	0	0	2	0	-

Table 4.2: Order Matrix for Task Tree Instance Variant 1

After all TTI variants have been converted to order matrices, the latter are aggregated in one combined order matrix representing all variants. This aggregated order matrix contains in every matrix cell a 5-dimensional vector denoting the relative frequency of each relation for every pair of TIs in all TTI variants.
To create the aggregated order matrix based on the order matrices of the different TTI variants, the relations between all pairs of TIs have to be evaluated in terms of their relative frequencies. For this purpose, the relation for a certain pair of TIs in each order matrix is weighted with the relative frequency of the respective TTI variant. The weights of all order relations are then summed up across all TTI variants and the resulting distribution of the different relations for this pair of TIs is then stored in the aggregated order matrix.
Table 4.3 shows the order relations between *"Task B"* and *"Task C"* for all TTI variants as they are contained in their corresponding order matrices.

TTI Variant	1	2	3	4
Variant Frequency	0.2	0.4	0.15	0.25
Order Relation	1	2	0	0

Table 4.3: Order Relations between Tasks B and C in all TTI Variants

The matrix cell for *"Task B"* and *"Task C"* in Table 4.4 illustrates the aggregated order matrix. Since TTI variants 3 and 4 with a combined frequency of 40% do not contain *"Task C"*, the value for no relation between the two TIs is set to 0.4. In TTI variant 1 (frequency of 20%) *"Task B"* precedes *"Task C"* leading to a value of 0.2 for *relation type 1*. Finally, TTI variant 2 contains *"Task B"* as successor of *"Task C"* setting the value of the successor relation to 0.4 according to the frequency of TTI variant 2.
If these two TIs were clustered together, their order relation would be set to *"Task C"* followed by *"Task B"* as this relation has the highest value with 40%. The *relation type 0* has the same value. However, this relation type must be ignored as it is not possible to cluster TIs together without an order relation between them.

Relation Type	0	1	2	3	4
Frequency	0.4	0.2	0.4	0	0

Table 4.4: Single Matrix Cell of an Aggregated Order Matrix

4.4.3 Determining the Cluster Block

In the next step, the TIs (or blocks of TIs respectively) that should be clustered have to be selected. The clustering approach for task trees builds cluster blocks from the bottom to the top, due to the structure of task trees. This way, leaves of the task tree are clustered first and the top-level tasks are clustered late. The reason for this approach lies in the fact that if tasks from the top-level were clustered first, it would be impossible to determine the correct position for a child task of this cluster block. For example, two TIs A

and B are clustered together first as a cluster block containing *"Task A"* followed by *"Task B"*. Then, in the next step, *"Task C"* should be added as a subtask of this cluster block. Since it is unknown whether *"Task C"* should be a subtask of *"Task A"* or a subtask of *"Task B"*, the correct position for *"Task C"* in the task tree cannot be determined. In contrast, it is no problem to add the cluster block consisting of *"Task A"* and *"Task B"* as a subtask of *"Task C"*, since multiple TIs can have the same parent task in a task tree, but not vice versa. Therefore, TIs are clustered on the lowest level of the task tree first, making the *average block level* the main selection criteria for blocks to be clustered.
Initially, the average block level of every TI is determined during the computation of the aggregated order matrix. The average block level of a TI is the weighted average of the hierarchical position of this TI in all TTI variants.
In the example shown in Figure 4.1, *"Task D"* is on the top-level of all TTI variants and, hence, the average block level of *"Task D"* is equal to 0. Similarly, the average block levels of *"Task D3"* and *"Task D3.2"* are equal to 1 and 2, respectively. If two TIs are clustered together, the average block level of the new cluster block will be set to the average of the current average block levels of the cluster block's constituting parts.
In case there are multiple candidates for clustering a *separation value* is used as a secondary criteria to determine the best candidate. The idea behind the separation value is to select blocks for clustering which have similar relations to the remaining blocks.
To determine the two blocks to be clustered, the average block level of all block pairs is computed first. If only one such pair has the highest average block level, this pair is chosen as cluster block. For multiple such pairs the separation values are computed and the pair with the highest separation value is selected for clustering.
In the example shown in Figure 4.1 a total of 16 tasks have to be considered, but only the three children of "Task D3" each have a block level of 2, leading to the highest average block level. Because "Task D3.2" (transitively) precedes "Task D3.4" is the only task pair contained in all variants, it should form the first cluster block.

4.4.4 Determining the Order Relations of the Cluster Block

Once a cluster block has been determined, a decision about the order relation between the two parts of this new block must be made. Therefore, the relation with the highest value in the order matrix is selected to connect the two existing blocks. If the *relation type 0* has the maximum frequency value of all order relations, the next best relation would be chosen alternatively as two blocks cannot be clustered without an order relation between them.
As for the example in Section 4.4.3, the order relation *"Task D3.2"* precedes *"Task D3.4"* holds true for all TTI variants (i.e. the relative frequency of this relation type is 1.0). Consequently, the first cluster block contains *"Task D3.2"* followed by *"Task D3.4"*.

4.4.5 Recomputing Aggregated Order Matrix

Following on the clustering of two cluster blocks, the aggregated order matrix needs to be recomputed. Since the two selected cluster blocks are now represented as one cluster block, the matrix shrinks by one dimension. The order relations of the cluster block to all other cluster blocks have to be updated accordingly. For a change block consisting of two cluster blocks A and B, the order relations to all other cluster blocks X are calculated as the average of the order relations between cluster blocks A and X and between cluster blocks B and X.

4.4.6 Mining Result

The previously described steps from Section 4.4.3 to 4.4.5 are repeated until all tasks have been clustered together into one cluster block. This final cluster block represents the common TTT for the given set of TTRs.
Figure 4.3 illustrates the application of the cluster mining algorithm to the TTI variants shown in Figure 4.2. The intermediate results are highlighted to emphasize the order in which TIs are grouped together. In the first three iterations, the subordinated TIs of *"Task D3"* have been clustered as these TIs had the highest average block level. In the

next four iterations, the TIs D1 to D5 are grouped together and the 8th iteration clustered the TIs E2 and E3. As supposed for a *bottom-up algorithm* the subordinated TIs of D and E were clustered during the 9th and 10th iteration before the TIs on the top-level. The latter were grouped during the final iterations and the resulting TTT is depicted on the right side of Figure 4.3.
In this case, the resulting TTT is heavily influenced by TTI variant 2, since this TTI variant has the highest frequency value with 40%. However, the mined TTT fulfills all constraints common to all TTI variants and shown at the bottom of Figure 4.1. Therefore, the automatically derived TTT should be more suitable for future TTIs of the same kind as the provided TTRs.

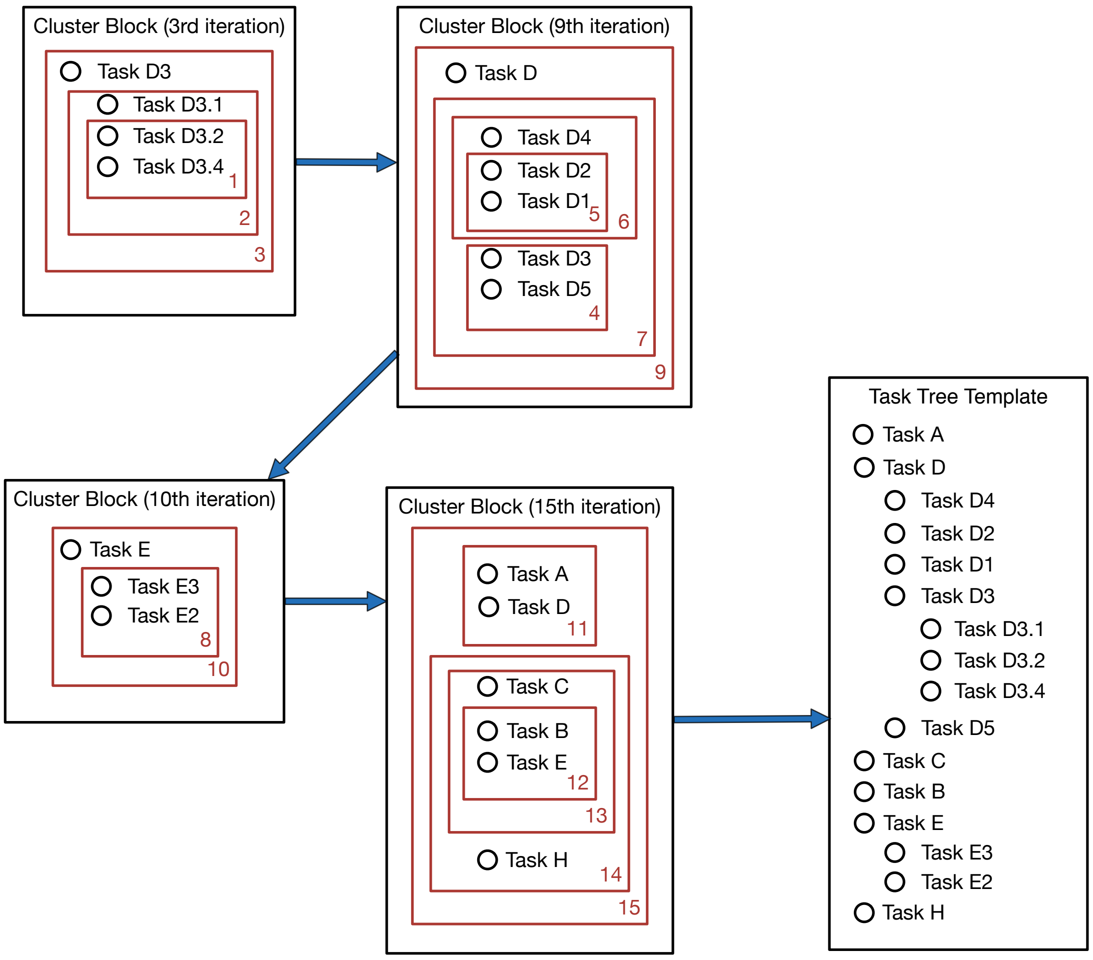

Figure 4.3: Example for Cluster Mining Approach

4.5 Summary

This chapter proposes a cluster mining approach to derive a common TTT for a set of related TTRs. First, a matching between the TIs of the different TTRs must be established. Once the TIs have been matched successfully, the set of TTI variants represented by the given TTRs must be determined. In the next step, the frequencies of TIs have to be analyzed to decide which of them are to be included in the TTT. The TTI variants have to be converted to order matrices first to generate an aggregated order matrix representing all TTI variants together. In an iterative process, the cluster mining algorithm builds cluster blocks of TIs with similar order relations to the remaining TIs. The algorithm terminates as soon as all TIs have been clustered into one cluster block, ultimately representing the common TTT for the given TTRs.

5 Providing Task Recommendations at Run Time

5.1 Problem Statement

Driven by KiBPs' *uncertainty* and their *emergent* nature contrasting to traditional business processes, knowledge workers often need to switch between the planning of upcoming tasks and the actual accomplishment of these tasks at run time. To adequately support knowledge workers, proCollab should be able to recommend tasks to them that are likely to be selected next. For this purpose, the current state of a TTI may be evaluated and compared with TTRs similar to the TTI. Then, recommendations can be derived by identifying open tasks (i.e. tasks that have not been accomplished yet) in the TTI that are directly following on the most recently completed task in TTRs. Such task recommendations, thereby may provide guidance to knowledge workers at run time and can help to decrease the necessary time for planning of upcoming tasks.

5.2 Approach

Basically, there are three major challenges induced by the problem statement: initially identifying a set of TTRs comparable to the given TTI, establishing a matching between the TIs in these TTRs and the TTI, and, finally, determining appropriate task recommendations. The following Sections 5.2.1–5.2.3 briefly elaborate on the former two issues and, subsequently, put the focus on an idea for a solution addressing the latter problem.

5.2.1 Selecting relevant Task Tree Records

Before task recommendations can be provided to support knowledge workers at run time, an appropriate set of TTRs must be selected first. If the TTI in use is based on a TTT, the analysis will utilize TTRs based on the same TTT. In this case, most tasks in the TTT and TTRs can be easily matched by taking into account the common TTs the TIs have been based on.
Otherwise, it is difficult to choose suitable TTRs as input for the analysis. A comparison with all available TTRs is hardly an option. Hence, other means are required to determine appropriate TTRs. For example, the information listed in the following can be leveraged to select such TTRs.

- TTRs associated with the same PT as the examined TTI
- TTRs involving the current user of the examined TTI

Since a PT usually contains multiple TTTs, the set of PTs in the process repository should be significantly smaller than the amount of TTTs in a task tree repository. Hence, the first step towards the identification of suitable TTRs should be the determination of a corresponding PT for the PI of the currently analyzed TTI. This can be done by leveraging meta information about PTs and the PI. Additionally, it would be useful to categorize PTs and PIs. Thereby, the effort required to determine an appropriate PT

for a certain PI could be further reduced. Once a PT has been determined, the TTTs contained in this PT can be compared with the examined TTI.
In particular, a matching between tasks in the TTTs and the TTI must be established. For this purpose, techniques from activity label matching for business process models can be utilized as stated in Section 3.3.1. Then, the TTRs of the TTT with the best *similarity score* should be selected to be compared with the TTI.

5.2.2 Scoring the Similarity of TTRs to a TTI

For each TTR found, a matching between the TIs of the TTI and then in the TTR has to be determined. Obviously, a promising TTR should have an execution log, in which TIs have been perfomed in a similar order as in the analyzed TTI. Therefore, the score to calculate the matching quality between a TTR and a TTI should be based on the comparison of their execution logs. In particular, the following should be considered for every TI in the examined TTI:

- Is there a corresponding TI in the TTR for each TI in the examined TTI?
- Do the direct predecessor and succesor of a TI in the TTI correspond to a **transitive** predecessor and successor, respectively, in the TTR?
- Do the direct predecessor and succesor of a TI in the TTI correspond to a **direct** predecessor and successor, respectively, in the TTR?

Taking all this information into account, a weighted overall score should be determined for every TTR that is compared to the TTI. Therefore, an appropriate weight should be assigned to each of these three components. First, the similarity score for each component is computed separately. Subsequently, the overall score for the similarity between the TTI and the TTR is determined by summing up every component's weighted similarity score. Once, the similarity score of every TTR has been calculated, the TTRs can be sorted according to their similarity score. The best fitting TTRs can be used to derive task recommendations.
Figure 5.1 illustrates an example how these criteria can be leveraged to determine a similarity score for the execution logs of a TTR and the TTI. In this example, it is

assumed that TIs have been matched before and that identical tasks have the same task name. As indicated by the red arrows, four of the five tasks in the execution log of the TTI correspond to tasks in the TTR. This leads to a similarity score of 0.8 for this criteria, calculated as the number of successfully matched tasks in the execution log of the TTI divided by the total number of tasks in the execution log of the TTI.

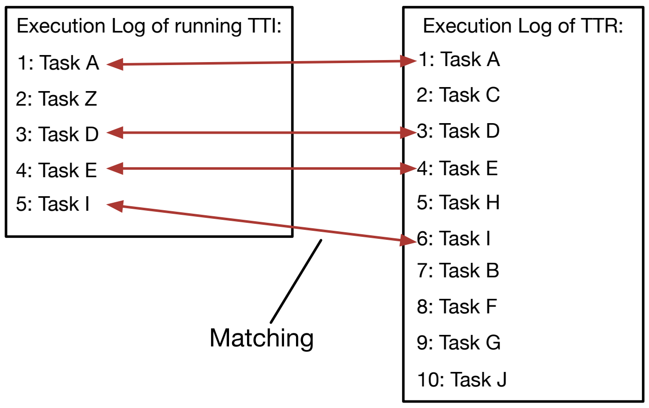

Figure 5.1: Execution Log Matching

Table 5.1 shows the predecessor/successor relationships between the matched TIs in Figure 5.1. For every TI in the TTI, the actual predecessor and successor in the TTR are listed in Table 5.1. The fact whether this corresponds to the predecessor and successor in the TTI is stated in brackets. Note, that *Task Z*, which could not be matched, is omitted here as it would significantly lower the value of the score for the relationships. This does not pose a problem, since TTRs, which all tasks are matched successfully, will get a higher similarity score for the respective component of the overall score. Therefore, the execution log of the TTI is now considered as the sequence of tasks *A*, *D*, *E*, and *I*.
The pair of *Task E* and *Task I* is an example for two tasks for which the transitive successor relation is fulfilled in the TTR. However, the direct successor relation is not fulfilled, i.e. *Task E* is succeeded by *Task H*, which is, in turn, followed by *Task I*.
To calculate the similarity score of the direct and transitive predecessor/successor relationships, the number of fulfilled order relations is divided by the number of all considered order relations. For the direct predecessor/successor relationships, this

leads to a similarity score of 0.33 (i.e. two matches divided by six relationships in total), while the transitive relations have a similarity score of 1.0 (i.e. six matches divided by six relationships in total).
Assuming weights of 50% for the score of corresponding tasks, 30% for correct transitive order relations, and 20% for correct direct order relations, the overall score can be computed accordingly. For this example, the weighted overall score is 0.766, calculated from the single components with their respective weights: 0.5 * 0.8 + 0.3 * 1.0 + 0.2 * 0.33.

Task Name	Direct Predecessor	Direct Successor	Transitive Predecessor	Transitive Successor
A	-	C (false)	-	D (true)
D	C (false)	E (true)	A (true)	E (true)
E	D (true)	H (false)	D (true)	I (true)
I	H (false)	-	E (true)	-

Table 5.1: Relationships between Tasks in Execution Logs

5.2.3 Deriving Task Recommendations

The identification of TTRs with similar execution logs to the given TTI enables the determination of tasks within these TTRs that are likely to be chosen next. The set of tasks to recommend is limited to the open tasks in the TTI. Obviously, already completed tasks can not be recommended as well as tasks exclusively contained in the TTR or the TTI (i.e. tasks that could not be matched). The open tasks, which have been successfully matched, can be classified in two disjunct sets. In the TTR, these TIs either have been executed before the most recently completed TI of the TTI or afterwards. Naturally, those TIs that have been performed earlier in the TTR and that are still open in the TTI are candidates to be selected next. Similarly, TIs directly following on the most recently completed TI are candidates that are likely to be chosen next.
To illustrate the determination of task recommendations, Figure 5.2 shows a TTI and a TTR sharing the same set of tasks. The execution log of the TTI contains four tasks that

have been completed so far and matched to their counterparts in the TTR's execution log as highlighted by red arrows. Furthermore, the blue boxes in the execution log of the TTR highlight tasks to be recommended to the user of the TTI. For example, *Task C* and *Task H* were both executed before *Task I*, which is, in turn, the most recently completed task in the TTI. Therefore, it seems likely that one of these tasks should be performed next. Similarly, *Task B* could be following on *Task I* as this is the behavior observed in the execution log of the TTR shown in Figure 5.2.

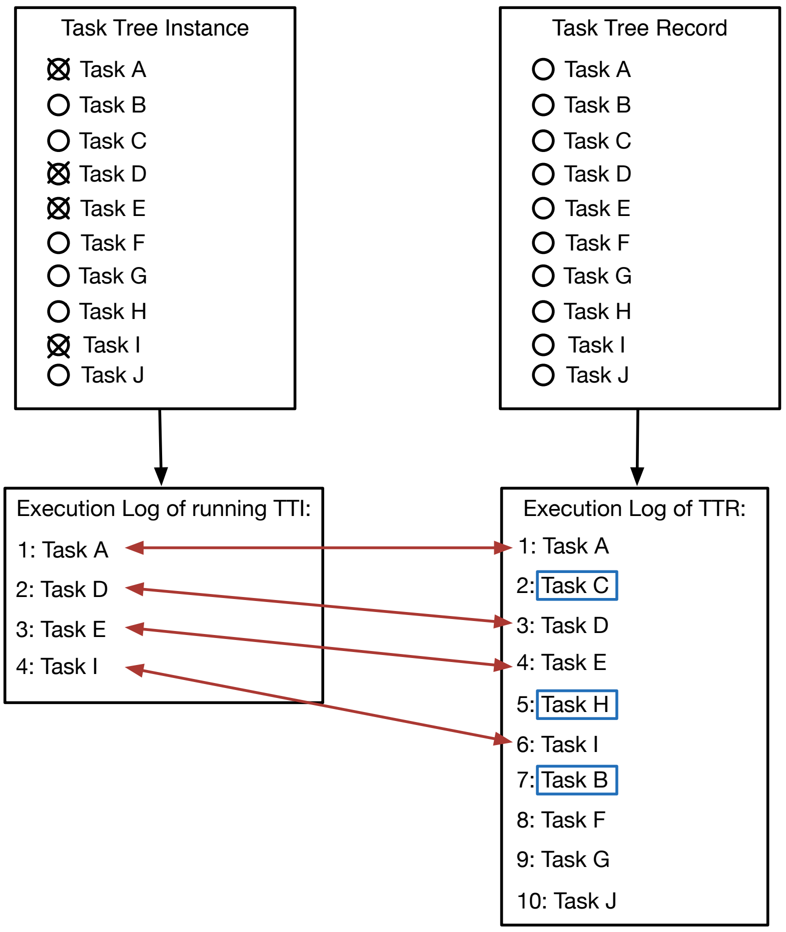

Figure 5.2: Providing Task Recommendations

6 Implementation

This chapter presents several aspects of the proof-of-concept implementation of the previously introduced optimization approaches. In Section 6.1, the architecture of the proCollab prototype is explained. Subsequently, Section 6.2 describes technologies employed by the proCollab protoype. Finally, Section 6.3 highlights selected excerpts of the implementation of the concepts proposed in Chapters 3 and 4.

6.1 Architecture of the proCollab prototype

The proCollab prototype is based on a multi-layer architecture as shown in Figure 6.1. To provide knowledge workers with high flexibility, proCollab can be used with web clients as well as mobile clients (i.e. tablets and smartphones) ([12], [19], [34], [23], [7]).
For the communication between client and server, a Representational State Transfer

(REST) interface is utilized. Clients use this REST interface to search for task trees and to retrieve them. Furthermore, they may call change operations to TTIs and TIs respectively.

The application layer contains the core components of the proCollab server. In particular, the server manages all instances (i.e. PIs, TTIs, and TIs) and the repositories for processes and tasks. The latter contains both templates and records of processes, task trees, and tasks. Additionally, the proCollab server also logs all change operations and state transitions of TTIs and TIs in the execution and change logs of the respective instance.

Finally, the persistence layer uses the Java Persistence API (JPA) to store templates, instances and records in the database.

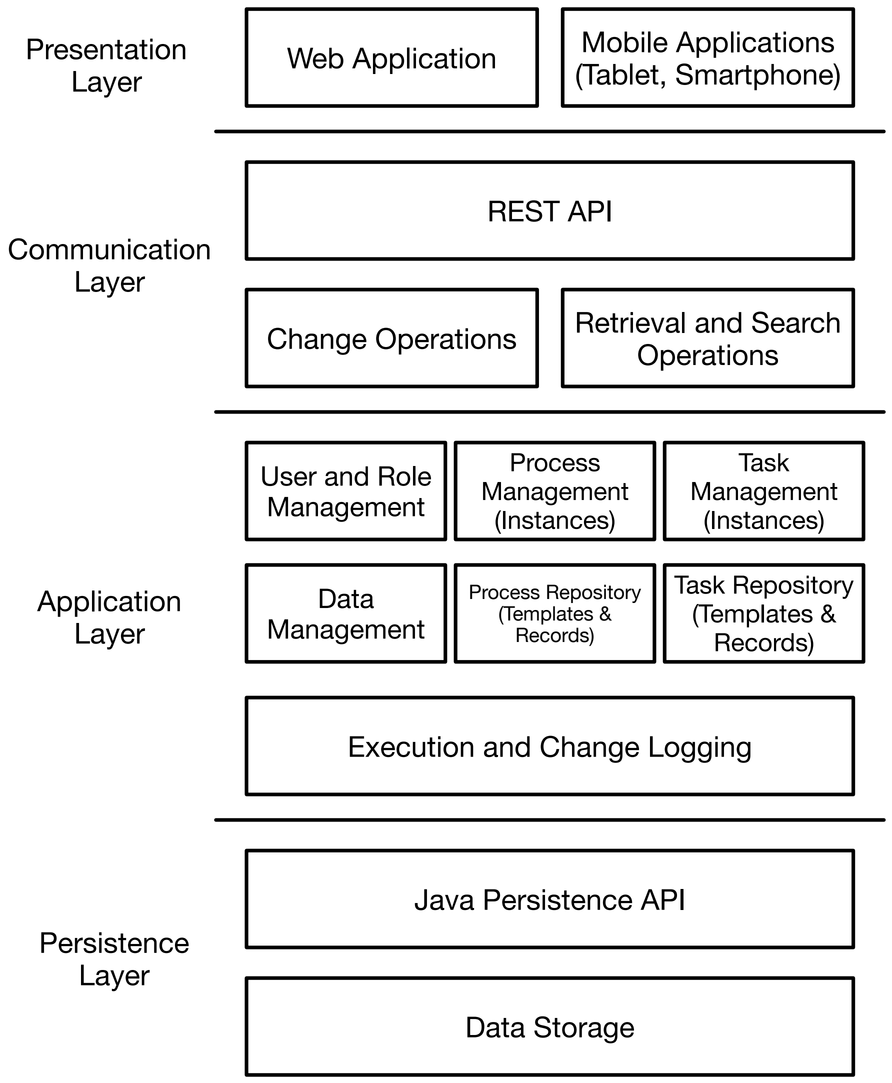
Presentation Layer
Web Application
Mobile Applications (Tablet, Smartphone)
Communication Layer
REST API
Change Operations
Retrieval and Search Operations
Application Layer
User and Role Management
Process Management (Instances)
Task Management (Instances)
Data Management
Process Repository (Templates & Records)
Task Repository (Templates & Records)
Execution and Change Logging
Persistence Layer
Java Persistence API
Data Storage

Figure 6.1: Architecture of the proCollab Prototype

6.2 Technologies

This section briefly introduces the technologies employed by the proCollab prototype.

6.2.1 Java Platform, Enterprise Edition

The Java Platform, Enterprise Edition (Java EE) [10] defines a software architecture for the component-based development of distributed web applications. Furthermore, Java EE supports multi-tier applications and provides transaction management. For the deployment of Java EE applications, an application server implementing the Java EE specification is required.

6.2.2 Representational State Transfer

REST is a programming paradigm used for the communication between client and server in distributed systems [5]. For this purpose, REST utilizes methods provided by the Hypertext Transfer Protocol (HTTP) (e.g. HTTP GET and PUT) to manipulate the state of resources used by the application. In particular, REST supports the CRUD operations to create, update, delete, and search for resources. A significant advantage of REST is that it is a lightweight solution compared with the competing standards Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL).

6.2.3 Java Persistence API

JPA specifies an object-relational mapping providing an interface between Java objects and relational database systems. This mapping is established between the attributes of a Java object and the columns of a table in the relational database system. Furthermore, the relations between these Java objects have to be defined either in separate configuration files based on Extensible Markup Language (XML) or with Java annotations. This is required to derive foreign keys and additional tables to especially represent the relations between tables in the relational database system properly. To query the relational database system, JPA allows usage of the Structured Query Language (SQL) as well as its own Java Persistence Query Language (JPQL).

6.3 Implementation Excerpts

This section presents some highlights of the implementation of the approaches described in Chapters 3 and 4.

6.3.1 Optimization of Task Tree Templates

Listing 6.1 shows the high-level algorithm used to optimize TTTs as it has been introduced in Chapter 3. The input of this algorithm consists of the TTT, which is to optimize, and the set of TTRs (i.e. completed TTIs) for the analysis. First, the change logs of the given TTRs have to be converted to the eXtensible Event Stream (XES) format [9] as stated in lines 3-4. Then, the multi-phase mining algorithm generates a *MiningResult* containing the C-net, which represents the change process, and the map of preceding partial traces for each node. For this purpose, the standard multi-phase mining algorithm provided by the ProM framework [29] has been extended accordingly. The next step is to determine the change blocks (i.e. sequences of the most frequent changes) based on the MiningResult as it can be seen in lines 7-8. Finally, the original TTT is optimized through the application of the identified change blocks and returned to the caller of this method (lines 9-11).

```
1 public CListType optimizeListType(CListType template,
2         List<CListInstance> instances) {
3     XLog changeLog =
4         XESConverter.convertChangeLogToXES(instances);
5     MiningResult miningResult =
6         MultiPhaseMiner.doMining(changelog);
7     List<ChangeBlock> changeBlocks =
8         analyzeCNet(miningResult, instances.size());
```

```
 9      CListType optimizedTemplate =
10          applyChangeBlocks(template, changeBlocks);
11      return optimizedTemplate;
12  }
```

Listing 6.1: Implementation of TTT Optimization

6.3.2 Analysis of Change Operations

The analysis of the most frequent change operations in the C-net is shown in Listing 6.2. Based on the *MiningResult*, which contains the C-net and the map of preceding partial traces, the set of valid change process variants is determined by traversing all paths in the C-net (lines 3-4). Subsequently, the relative frequencies of all change operations in the variants are calculated. This is necessary in order to create a list comprising only the most frequent change operations (i.e. those changes with a frequency value above a threshold, e.g. 60%). Since the multi-phase mining plugin does not support empty traces in change logs, the total number of analyzed TTRs (here named *"traceCount"*) is needed to ensure that the relative frequencies are computed properly (i.e. the unchanged TTRs are also taken into account). Finally, the change blocks are determined by traversing all change process variants and searching for sequences of the most frequent changes therein.

```
1   private List<ChangeBlock> analyzeCNet(
2           MiningResult miningResult, int traceCount) {
3       List<Variant> variants =
4           VariantSearcher.traverseCNet(miningResult);
5       List<ChangeOpFrequency> changeFrequencies =
6           identifyChangeFrequencies(variants, traceCount);
7       List<ChangeBlock> changeBlocks =
8           identifyChangeBlocks(variants, changeFrequencies);
```

```
    return changeBlocks;
}
```

Listing 6.2: Implementation of the Analysis of Change Operations

6.3.3 Determination of Change Process Variants

The algorithm used to extract the valid change process variants from the C-net is shown in Listing 6.3. Initially, the C-net and the map of preceding partial traces have to be retrieved from the given *MiningResult* and two lists of variants have to be prepared to store both the temporary and final results (lines 2-7). Subsequently, a new variant is created for each output binding of the C-net's start node and added to the queue (lines 9-16).
Then, every variant in the queue is inspected iteratively until the queue is empty (lines 19-21). Variants are removed from the queue as soon as they turn out to be invalid or the end node is reached. In the latter case, the variants are added to the result list containing all valid variants.
For every variant in the queue, an active node has to be determined first. To be selected as an active node of a variant, a C-net node may not have been visited before and it must have a fulfilled input binding. Thereby, it is ensured that the nodes bound by one input binding of the potential active node have been visited before this node, which is a precondition to enable a C-net node. If all C-net nodes have been visited, the respective variant has reached the end node and is moved to the result list (lines 22-26). On the other hand, if there are still C-net nodes left that have not been visited, but none of them has a fulfilled input binding, the respective variant is invalid and, therefore, has to be discarded (lines 29-33).
Further, the selected active node is checked against the map of preceding partial traces to ensure that the variant will remain valid, if it was extended with the currently examined node (lines 35-40). This step is necessary due to the underfitting of the C-net allowing for traces not observed in the change logs (cf. Section 3.3.2). In particular, the C-net provides no means to remember the choices made on previous XOR splits. Hence, all combinations of XOR alternatives constitute valid paths through the C-net leading

to the introduction of many additional variants. To bypass this issue, only variants corresponding to traces found in the change log will be added to the result list. Subsequently, the active node is marked as visited (line 41) and its output bindings have to be considered next. Therefore, the currently active variant is extended with the first output binding of the active node (i.e. the nodes bound by this binding are added to its active nodes). Since every additional output binding represents an alternative path through the C-net, a new variant has to be introduced and added to the queue to explore these paths separately (lines 51-58).

```
1  private List<Variant> traverseCNet(MiningResult miningResult) {
2      CustomCNet cNet = miningResult.getCNet();
3      Map<CustomCNetNode, List<List<CustomCNetNode>>
4          precedingPartialTraces =
5          miningResult.getPrecedingPartialTraces();
6      List<Variant> resultList = new ArrayList<Variant>();
7      List<Variant> queue = new ArrayList<Variant>();
8      // Fill queue with initial set of variants
9      for (CustomCNetBinding startOutputBinding :
10         cNet.getOutputBindings(cNet.getStartNode())) {
11         Variant variant = new Variant();
12         variant.addOutputBinding(startOutputBinding,
13             cNet.getStartNode());
14         variant.getVisitedNodes().add(cNet.getStartNode());
15         queue.add(variant);
16     }
17     // Explore C-net paths until all variants in the queue
18     // reached the end node or were discarded
19     while (!queue.isEmpty()) {
20         for (int i = 0; i < queue.size(); i++)  {
21             Variant variant = queue.get(i);
22             if (variant.getActiveNodes().isEmpty()) {
23                 // Move this variant from queue to result list
```

```
        ...
        continue;
    }
    CustomCNetNode activeNode =
        variant.getFulfilledActiveNode(cNet);
    if (activeNode == null) {
        // Discard this invalid variant
        ...
        continue;
    }
    // Verify variant to avoid the underfitting of the C-net
    if (!variantPathIsValid(variant, activeNode,
        precedingPartialTraces)) {
        // Discard this invalid variant
        ...
        continue;
    }
    variant.setActiveNodeVisited(activeNode);
    Set<CustomCNetBinding> outputBindings =
        cNet.getOutputBindings(activeNode);
    if (!outputBindings.isEmpty()) {
        Iterator<CustomCNetBinding> iterator =
            outputBindings.iterator();
        CustomCNetBinding expansionBinding =
            iterator.next();
        // Create an additional variant
        // for each output binding
        while (iterator.hasNext()) {
            CustomCNetBinding outputBinding =
                iterator.next();
            Variant newVariant = new Variant(variant);
```

```
                    queue.add(newVariant);
                    newVariant.addOutputBinding(outputBinding,
                        activeNode);
                }
                // The current variant is expanded with the path
                // represented by the first output binding
                variant.addOutputBinding(expansionBinding,
                    activeNode);
            }
        }
    return resultList;
}
```

Listing 6.3: Implementation of the Identification of Change Process Variants

6.3.4 Application of Cluster Mining

The implementation of the cluster mining algorithm as introduced in Section 4.4 is shown in Listing 6.4. The input of the algorithm consists of a set of variants, their relative frequencies, and a threshold value denoting the minimum occurence required by a task to be included in the common TTT (cf. Section 4.4). Initially, every variant has to be converted to an order matrix reflecting the order relations between all task pairs in the respective variant (lines 3-11). Based on these order matrices, an aggregated order matrix is computed containing the distribution of the order relations for all task pairs across all variants (lines 14-15). During the cluster mining iterations, a pair of blocks, i.e. either a single task or a set of clustered tasks, with the highest average block level is determined first (lines 23-24). Then the matrix cell representing the order relations of this custer block is retrieved from the aggregated order matrix and the relation type with the highest relative frequency is chosen to cluster this block. However, an exception applies for the relation type "0", which has to be ignored. The last step of the iteration is the recalculation of the aggregated order matrix (lines 35-37). The two rows and columns representing the clustered blocks have to be replaced by a single row and column for the new cluster block. The iteration has finished as soon as one matrix cell solely remains in the aggregated order matrix. Finally, the last cluster block, which contains all tasks, is returned to the caller as the common TTT for the given set of variants.

```
public CListType clusterMining(Map<CListType, Double> variants,
        double threshold) {
    Map<OrderMatrix, Double> orderMatrices = new
        HashMap<OrderMatrix, Double>();
    // Convert variants to order matrices
    for (Map.Entry<CListType, Double> entry :
            variants.entrySet()) {
        OrderMatrix orderMatrix =
            new OrderMatrix(entry.getKey());
        orderMatrices.put(orderMatrix, entry.getValue());
    }
    // Compute the aggregated order matrix representing all
    // variants together in one order matrix
    AggregatedOrderMatrix orderMatrix = new
        AggregatedOrderMatrix(orderMatrices, threshold);
    int numberOfTasks = orderMatrix.getNumberOfBlocks();
    // Cluster tasks from bottom to top until all tasks
    // are contained in the same cluster block
    for (int i = 1; i <= (numberOfTasks - 1); i++) {
        ClusterIndex cluster = new ClusterIndex(0, 1);
        if (orderMatrix.getValidDimension() != 1) {
            // Determine the next tasks to cluster
            cluster =
                getClusterWithMaxBlockLevel(orderMatrix);
        }
        MatrixCell clusteredBlock =
            orderMatrix.getMatrixCellAt(cluster.getIdxA(),
            cluster.getIdxB());
            // Determine the order relation of the cluster block
            Cohesion bestRelation =
                computeCohesion(clusteredBlock);
            // Recompute the aggregated order matrix (i.e.
            // remove the parts constituting the cluster block
            // from the order matrix and add the cluster block)
            orderMatrix.recomputeOrderMatrix(
                cluster.getIdxA(), cluster.getIdxB(),
                bestRelation.getRelationType());
        }
        TreeElement result = orderMatrix.getResultType();
        return (CListType) result;
}
```

Listing 6.4: Implementation of the Cluster Mining Algorithm

7 Conclusion

7.1 Conclusion

Currently, knowledge workers suffer from a lack of appropriate system support for KiBPs. In many cases, knowledge workers still rely on paper-based task lists (e.g. checklists, to-do lists) to manage their work. The proCollab reserach project aims at providing a systematic support for knowledge workers and enables them to manage their task lists through an integrated system.
The contribution of this thesis is the provision of concepts to extend proCollab with a more powerful lifecycle support. Since KiBPs are *knowledge-creating*, TTRs (i.e. successfully completed TTIs) can be leveraged to improve future TTIs of the same kind. In particular, knowledge workers should be provided with suitable TTTs that help to reduce the time required for the planning of upcoming tasks. For this purpose, a cluster mining

approach has been proposed, which is able to automatically derive a common TTT for a set of TTRs. Furthermore, TTIs and, thereby, the TTTs they have been based on, are subjects to change. Therefore, TTRs should be analyzed continuously to optimize existing TTTs and to keep them up to date. To address this goal, an approach has been developed optimizing TTTs by incorporating the most frequently applied changes. This is achieved through the application of a change mining algorithm examining change logs contained in TTRs based on the same TTT.
Another problem, that knowledge workers are regularly facing, is the definition or selection of the next task in their TTIs. To adequately support knowledge workers in this context, proCollab should be able to provide them with appropriate task recommendations at run time. For this goal, a basic concept has been presented showing how TTRs could be utilized to derive task recommendations for comparable TTIs at run time.
Finally, the presented concepts to derive and optimize TTTs have been successfully realized in a proof-of-concept implementation for the proCollab prototype.

7.2 Future Work

In future work, the concepts presented in this thesis should be evaluated based on case studies or experiments. This way, knowledge workers can assess the quality of the automatically derived and adjusted TTTs. Additionally, it should be analyzed how threshold values can be optimally set to provide adequate results.
Furthermore, an efficient approach to establish a matching between TIs in different TTIs should be developed. This is particularly challenging as a potentially very large amount of comparisons between TIs is required to determine, which of them correspond to each other.
Another problem arising in the context of optimizing a TTT is called *schema evolution* [21]. When a TTT is changed, it shall be checked whether the currently active TTIs based on this TTT can be migrated to the new version of this TTT. If it was possible, all of the derived TTIs could be adjusted in the same way as the TTT.
Since it could happen that TTIs cannot be migrated to a new version of the respective

TTT, a sophisticated versioning concept is required to manage the different versions of the same TTT used in parallel by the knowledge workers.

List of Figures

Bibliography

[1] Ian Brinkley. Defining the Knowledge Economy. *London: The Work Foundation*, page 19, 2006.

[2] Chen Li. *Mining Process Model Variants: Challenges, Techniques, Examples*. Dissertation, University of Twente, The Netherlands, 2010.

[3] Peter F. Drucker. The new Productivity Challenge. *Quality in Higher Education*, 37, 1995.

[4] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer, Berlin, Heidelberg, 2013.

[5] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Dissertation, University of California, Irvine, 2000.

[6] Jakob Freund and Bernd Rücker. *Praxishandbuch BPMN 2.0*. Hanser, München, 2014.

[7] Sabrina Geiger. *Konzeption und Entwicklung einer auf Smartphones optimierten mobilen Anwendung für kollaboratives Checklisten-Management*. Bachelor's Thesis, Ulm University, Ulm, 2013.

[8] Christian W. Günther, Stefanie Rinderle-Ma, Manfred Reichert, Wil M. P. van der Aalst, and Jan Recker. Using Process Mining to Learn from Process Changes in Evolutionary Systems. *Int'l Journal of Business Process Integration and Management, Special Issue on Business Process Flexibility*, 3(1):61–78, 2008.

[9] Christian W. Günther and H. M. V. Verbeek. XES Standard Definition. http://www.xes-standard.org/_media/xes/xesstandarddefinition-2.0.pdf, 2014. last accessed on 2015/06/07.

[10] Java EE Expert Group. JSR 342: Java Platform, Enterprise Edition 7 Specification. http://download.oracle.com/otn-pub/jcp/java_ee-7-fr-eval-spec/JavaEE_Platform_Spec.pdf, 2013. last accessed on 2015/06/07.

[11] Christopher Klinkmüller, Ingo Weber, Jan Mendling, Henrik Leopold, and André Ludwig. Increasing Recall of Process Model Matching by Improved Activity Label Matching. In *Proceedings of the 11th International Conference on Business Process Management*, BPM'13, pages 211–218, Berlin, Heidelberg, 2013. Springer-Verlag.

[12] Andreas Köll. *Konzeption und Enwicklung einer auf Tablets optimierten mobilen Anwendung für kollaboratives Checklisten-Management*. Bachelor's Thesis, Ulm University, Ulm, 2013.

[13] George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[14] Nicolas Mundbrod, Jens Kolb, and Manfred Reichert. Towards a System Support of Collaborative Knowledge Work. In *1st Int'l Workshop on Adaptive Case Management (ACM'12), BPM'12 Workshops*, LNBIP, pages 31–42. Springer, 2012.

[15] Nicolas Mundbrod and Manfred Reichert. Process-Aware Task Management Support for Knowledge-Intensive Business Processes: Findings, Challenges, Requirements. In *IEEE 18th Int'l Distributed Object Computing Conference - Workshops and Demonstrations (EDOCW 2014)*, pages 116–125. IEEE Computer Society Press, 2014.

[16] Object Management Group. Business Process Model and Notation (BPMN) Version 2.0. http://www.omg.org/spec/BPMN/2.0/PDF, 2010. last accessed on 2015/06/07.

[17] Carl A. Petri. *Kommunikation mit Automaten*. Dissertation, Universität Bonn, Bonn, 1962.

[18] proCollab. `http://proCollab.de`. last accessed on 2015/06/07.

[19] Daniel Reich. *Konzeption und Entwicklung eines Cloud-basierten Persistenz-Systems für kollaboratives Checklisten-Management*. Bachelor's Thesis, Ulm University, Ulm, 2013.

[20] Manfred Reichert and Barbara Weber. *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer, Berlin, Heidelberg, 2012.

[21] Stefanie Rinderle. *Schema Evolution in Process Management Systems*. Dissertation, Ulm University, Ulm, 2004.

[22] August W. Scheer. Business Process Engineering, Reference Models for Industrial Enterprises. *Springer, Berlin*, 1994.

[23] Norman Thiel. *Konzeption und Entwicklung einer Web-Applikation für kollaboratives Checklisten-Management*. Bachelor's Thesis, Ulm University, Ulm, 2013.

[24] Roman Vaculin, Richard Hull, Terry Heath, Craig Cochran, Anil Nigam, and Piyawadee Sukaviriya. Declarative Business Artifact Centric Modeling of Decision and Knowledge Intensive Business Processes. In *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*, pages 151–160, 2011.

[25] Wil M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(01):21–66, 1998.

[26] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, Berlin, Heidelberg, 2011.

[27] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Causal Nets: A Modeling Language Tailored towards Process Discovery. *CONCUR*, pages 28–42, 2011.

[28] Wil M. P. van der Aalst, Vladimir Rubin, H. M. W. Verbeek, Boudewijn F. van Dongen, Ekkart Kindler, and Christian W. Günther. Process Mining: A two-step Approach to Balance between Underfitting and Overfitting. *Software & Systems Modeling*, 9(1):87–111, 2010.

[29] Boudewijn F. van Dongen, de Medeiros, Ana Karla A, H. M.W. Verbeek, AJMM Weijters, and Wil M. P. van der Aalst. The ProM Framework: A new Era in Process Mining Tool Support. In *Applications and Theory of Petri Nets*, pages 444–454. Springer, Berlin, Heidelberg, 2005.

[30] Boudewijn F. van Dongen and Wil M. P. van der Aalst. Multi-phase Process Mining: Building Instance Graphs. In *Conceptual Modeling – ER 2004*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer, Berlin, Heidelberg, 2004.

[31] Boudewijn F. van Dongen and Wil M. P. van der Aalst. Multi-phase Process Mining: Aggregating Instance Graphs into EPCs and Petri Nets. In *PNCWB 2005 workshop*, pages 35–58, 2005.

[32] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. Change Patterns and Change Support Features - Enhancing Flexibility in Process-aware Information Systems. *Data & Knowledge Engineering*, 66(3):438–466, 2008.

[33] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Computer Science. Springer, Berlin, 2012.

[34] Jule Ziegler. *Konzeption und Entwicklung eines Cloud-basierten Servers für kollaboratives Checklisten-Management*. Bachelor's Thesis, Ulm University, Ulm, 2013.

Name: Florian Beuter Matrikelnummer: 668730

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den ....

Florian Beuter