Supporting Knowledge-intensive Processes Through Integrated Task Lifecycle Support

Nicolas Mundbrod, Florian Beuter, Manfred Reichert
Institute of Databases and Information Systems
Ulm University, Germany
Email: {nicolas.mundbrod, florian.beuter, manfred.reichert}@uni-ulm.de

***Abstract*—The operational support of knowledge-intensive business processes constitutes a big challenge. In particular, these processes are driven by knowledge workers utilizing their skills, experiences, and expertise. Regarding coordination and synchronization, in turn, knowledge workers still rely on simple task lists (e.g., to-do lists or checklists) and established communication software (e.g., email). While these means are prevalent and intuitive, they are ineffective and error-prone as well. Neither tasks are made explicit, synchronized, personalized, nor are they independent from media breaks. Most important, a task management lifecycle is not provided, i.e., the efforts and knowledge invested by the knowledge workers in task management are not preserved for comparable future endeavors. This work introduces the proCollab approach proposing a systematic and lifecycle-based task management support for knowledge workers. To establish a sound task management lifecycle, in particular, we apply process mining to analyze knowledge workers' changes applied to task lists in order to derive optimizations task list templates. To demonstrate feasibility and benefits, a proof-of-concept prototype was developed and applied. Overall, the integrated, systematic and lifecycle-based task management support is prerequisite for the effective IT support of KiBPs.**

Keywords*—task management, knowledge–intensive business process, adaptive case management, knowledge workers, process mining, to-do lists, checklists

I. Introduction

A structural shift from an industrial towards a knowledge-based society has been taking place in highly developed countries [1]: knowledge-intensive business processes (KiBPs) residing in sensitive key business areas (e.g., research, development, or service) have become predominant in many companies. Driving KiBPs, knowledge workers (e.g., engineers, or physicians) leverage their distinguished skills, experiences and expertise to cope with novel and sophisticated tasks. Thus, the systematic support of knowledge-intensive business processes and therein involved knowledge workers have become a crucial prerequisite for overall business success these days.

As a result, Adaptive Case Management (ACM) has been established as a new trend in BPM research focusing on the IT support of KiBPs [2]. However, this support is evidently challenging due to the characteristics of KiBPs (cf. Figure 1) which are *non-predictable*, *emergent*, *goal-oriented*, and *knowledge-creating* [3]. Hence, KiBPs have not been supported by any kind of process-aware information systems at the operative level so far. In turn, knowledge workers, who aim at achieving common goals, still rely on communication tools (e.g., email) and simple task lists (e.g., to-do lists, checklists) to coordinate each other [4]. These means are intuitive and, in consequence, prevalent on one side, but unfortunately highly error-prone and ineffective on the other [5]. In particular, tasks are often managed paper-based, not explicitly represented as coordination artifacts, and spread over different places. Thus, knowledge workers suffer from media breaks as well as a lack of synchronized and lifecycle-based task management. Especially the latter prevents knowledge workers to leverage coordination artifacts (e.g., task lists) in comparable contexts (i.e. KiBPs) to increase their productivity.

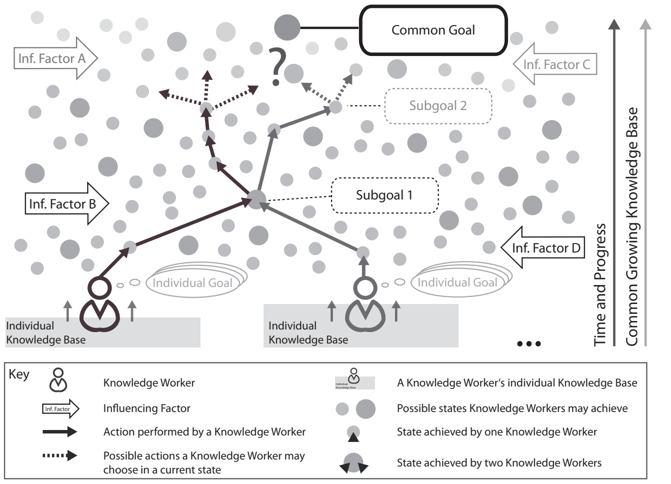
Common Goal
Inf. Factor A
Inf. Factor C
?
Subgoal 2
Subgoal 1
Inf. Factor B
Inf. Factor D
Individual Goal
Individual Goal
Individual Knowledge Base
Individual Knowledge Base
...
Time and Progress
Common Growing Knowledge Base
Key
Knowledge Worker
Influencing Factor
Action performed by a Knowledge Worker
Possible actions a Knowledge Worker may choose in a current state
A Knowledge Worker's individual Knowledge Base
Possible states Knowledge Workers may achieve
State achieved by one Knowledge Worker
State achieved by two Knowledge Workers

Fig. 1. Characteristics of KiBPs

In this work, we present the *proCollab*[1] approach aiming at the systematic support of KiBPs and therein involved knowledge workers. As tasks constitute the key objects for knowledge workers when it comes to coordination in KiBPs, proCollab particularly provides the foundation of process-aware and lifecycle-based task management empowering collaborating knowledge workers to coordinate each other more effectively. To leverage best practices and knowledge gained in comparable KiBPs, proCollab enables the process-aware provision of *task list templates* to let knowledge workers instantiate these templates on demand. To successfully establish a lifecycle-based support for KiBPs, we further introduce an approach based on process mining to continuously analyze knowledge workers' actual usage of instantiated task list templates. Thereby, we are able to derive optimizations for the task list templates over time and to evolve them accordingly. Finally, the feasibility of establishing an integrated task management lifecycle is demonstrated by a proof-of-concept prototype.

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

The remainder of this paper is organized as follows: Section II presents the applied methodology and required basics. Section III introduces the proCollab approach presenting its core components and their interplay. In turn, Section IV presents the approach of analyzing the usage of instantiated task list templates in order to improve and evolve the templates. Section V describes the proCollab proof-of-concept prototype. Section VI discusses related work and Section VII concludes the paper with a summary and outlook.

II. BACKGROUNDS

A. Methodology

This work is part of a long-term project targeting at establishing a systematic support of KiBPs. Therefore, the *design science* research methodology [6] is applied by us to assure high research quality. Regarding the *design science research process* [7] in particular, we previously addressed the *problem identification and motivation* phase [3] as well as the phase of identifying the *objectives of a solution* in the shape of challenges and requirements [8], [5]. Drawing upon, this paper may be categorized as a *design- and development-centered approach* and presents the following key contributions:

1) The proCollab meta-model, which has been designed to establish an integrated, enables process-aware task management support for KiBPs. proCollab comprises templates and instances for its interconnected components to provide the necessary foundation for a task management lifecycle.
2) An integrated task management lifecycle approach based on process mining is introduced to continuously analyze knowledge workers' usage of instantiated task list templates in order to evolve these templates realizing a sustainable support of KiBPs.

B. Knowledge-intensive Business Processes

While [9] provides a detailed discussion of different KiBP notions and definitions, this paper uses the notion of *knowledge-intensive business processes* (KiBPs) based on [10]:

"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."

In [3], we proposed the *KiBP Lifecycle* (cf. Figure 2) as an essential foundation for every approach supporting knowledge workers involved in KiBPs. As the proCollab approach presented in Section III directly relies on this lifecycle, the phases are discussed in the following:

Orientation: In this first lifecycle phase, information about the KiBP is systematically collected and analyzed. Based on interviews, analysis, and existing literature a description for the KiBP is compiled.

Template Design: Subsequently, a *collaboration template* (CT) is defined for the respective KiBP. A CT comprises the coordination artifacts likely used by the knowledge workers at run time. Specified CTs are then offered to knowledge workers at the collaboration run time.

Collaboration Run Time: To actively start a guidance, knowledge workers instantiate a CT to create a *collaboration instance* (CI). A CI determines the supportive guidance offered by an approach to the knowledge workers involved in a concrete instance of the KiBP. In the context, the CI may be continuously adjusted by the knowledge workers.

Records Evaluation: On one side, knowledge workers involved in a CI can make use of insights from *comparable* collaboration records (i.e., archived CIs). On the other, ab analysis of collaboration records is performed to learn from them and to improve, i.e. evolve existing CTs.

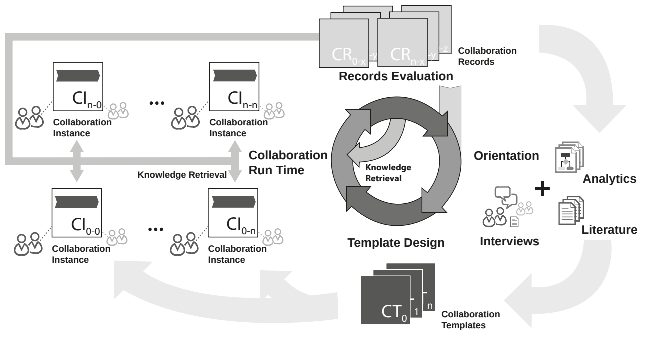

Fig. 2. KiBP Lifecycle

C. Application Scenario

To systematically examine how knowledge workers collaborate and work in KiBPs, we have studied various representative application scenarios to compile the key challenges and requirements for the support of KiBPs in detail [8]. We now recap one representative application scenario to draw attention to the challenging requirements of an integrated, lifecycle-based task management support for KiBPs as well as to facilitate the understanding of the ongoing sections.

In development projects for electrical and electronic (E/E) car components (cf. Fig. 3), the common goal for the involved knowledge workers is to develop an E/E car component until a fixed release date [11]. Hundreds of professionals (e.g., engineers) from different disciplines and partners (i.e., OEM and suppliers) are involved in long-running E/E development projects up to several months or years. While there are initially predefined project roles, many professionals or even entire organizations (e.g., suppliers) are invited to the project on demand. Hence, the knowledge workers must follow a development methodology with sub-goals (called *quality gates* and *milestones*) to ensure effective E/E development. Obviously, the projects' development phases comprise various sub-phases as well as parallel development processes that have to be managed properly. Therefore, involved knowledge workers need to frequently communicate and especially synchronize with each other (e.g., in meetings).

To foster the quality of development processes, to ensure compliance with regulations, and to track the development progress, a large, central project *checklist* with hundreds of *check items* is initially created and continuously managed by a dedicated quality assurance officer. Usually, the officer regularly discusses the currently relevant check items with the project members in the scope of an interview. Additionally, *to-do lists* and *task sheets* are dynamically used by the knowledge workers to manage personal tasks as well as to coordinate each other in smaller, specialized teams. In summary, checklists and their items are used for quality assurance (*retrospective work*) whereas to-do lists are used to dynamically plan and coordinate work in future (*prospective work*). In prior work [5], we additionally observed that checklists (e.g., spreadsheets) are not supposed to be changed much (for the sake of quality assurance), whereas to-do lists or task sheets strongly require frequent updates and, in particular, the insertion of new tasks. Nonetheless, we observed in all inspected application scenarios that neither checklists nor to-do lists are systematically managed in an integrated, synchronized and lifecycle-based manner.

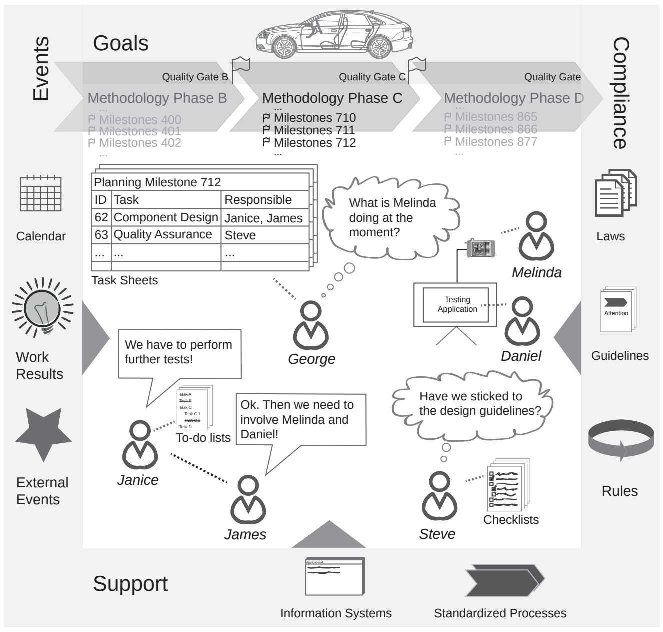

Fig. 3. Overview of an E/E Development Project

III. THE PROCOLLAB APPROACH

Drawing upon our prior work [3], [5], [8], we have developed the lifecycle-based proCollab approach enabling an integrated and systematic task management support of KiBPs. Thereby, we especially considered the fact that knowledge workers frequently switch between planning and performing work due to the challenging characteristics of KiBPs [12]. Consequently, knowledge workers strongly rely on managing and communicating tasks in relation to common goals. Further, we have focused on the proper, but lightweight representation of the KiBP Lifecycle (cf. Section II-B) as well as established artifacts, i.e. checklists and to-do lists (cf. Section II-C). Thereby, we aim at developing and providing an integrated approach for knowledge workers involved in KiBPs.

Figure 4 provides an overview of the proCollab key components, i.e. *processes*, *task trees*, and *tasks* (cf. Sections III-A-III-C). As the KiBP Lifecycle is directly incorporated, each proCollab key component can be further distinguished in corresponding templates, instances, and records (cf. Sections III-D-III-F).

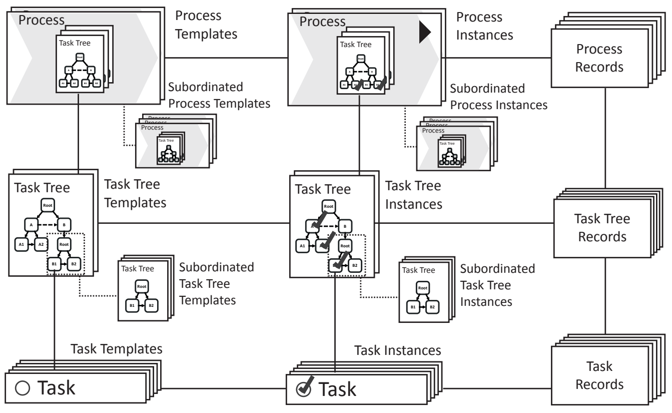

Fig. 4. Overview of the proCollab Components

To establish a generic support for knowledge workers, proCollab relies on these generic data structures and still allows for the type-, domain-, and purpose-specific specialization of the generic proCollab components. For instance, a proCollab process may be specialized to various medical cases regarding patient treatment (cf. Figure 5) whereas a proCollab task tree may be detailed into a quality assurance checklist (cf. Section II-C).

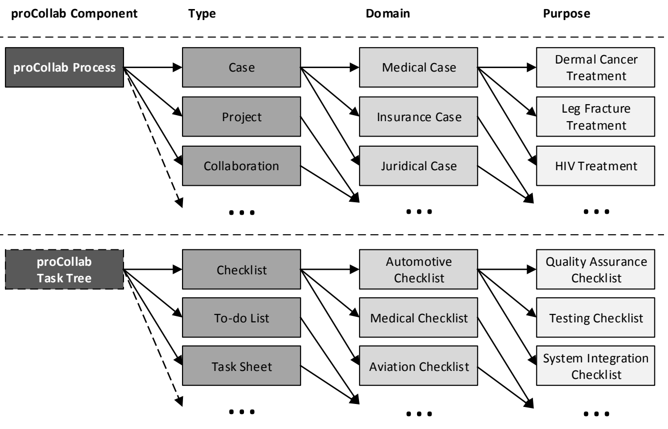

Fig. 5. Examples for the Specialization of proCollab Components

A. Processes

In practice, knowledge workers collaborate in the scope of *projects*, *cases*, or just *temporary endeavors* [3]. As these are all organizational and temporal frames representing KiBPs, the notion of *process* is used in proCollab to generalize from these. Naturally, every process may be arbitrarily nested (e.g., sub-projects). Further, a process always exposes a goal the knowledge workers want to achieve at the end, relevant conditions (e.g., due dates, available resources), linked resources (e.g., documents) and organizational assignments (e.g., project roles and corresponding rights). Depending on the specialization of a process (e.g., a project), knowledge workers may further add specific conditions, constraints, and organizational assignments. Finally, every process links to task trees enabling knowledge workers to coordinate each other in order to successfully achieve the process goal(s).

B. Task Trees

proCollab provides the generic data structure of a task tree enabling the definition and usage of established task lists, i.e. to-do lists and checklists (cf. Figure 6). The latter are heavily used by the knowledge workers to coordinate tasks among each other as well as to establish the important *work awareness* of who is doing what in the current context, i.e. KiBP, [13]. As opposed to ordinary tree structures, an recommended order is specified in which tasks may be processed by the knowledge workers. However, knowledge workers may always deviate from the recommended order to address the current situation in a KiBP. Through task lists knowledge workers iteratively specialize (coarse-grained) tasks by defining more detailed sub-tasks. Thus, a certain task may be connected to an arbitrary set of subordinated tasks that are supposed to be performed first in order to complete the actual task itself.

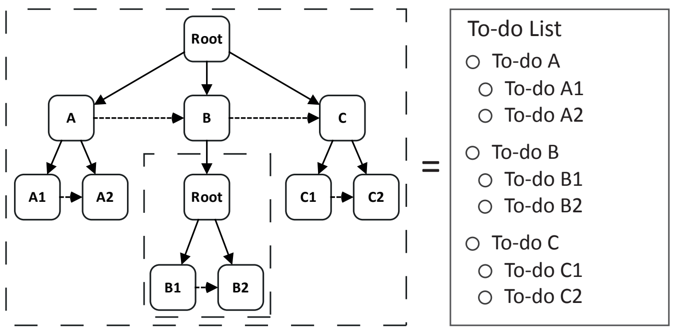

Fig. 6. Example of Nested Task Tree and To-do List Specialization

In proCollab, every task tree exposes one root node with a number of ordered child nodes (cf. Figure 6). The latter may have, in turn, further child nodes that are ordered as well. Apart from the root node, every node in a task tree is either a task or an embedded task tree. The root node is not shown as a task, but may be leveraged to store the purpose of the task list. As task trees can be nested, loops are not permitted to be designed. The ordering of the tasks constitutes a recommendation for knowledge workers. As it is not prescribing, the proCollab task trees offer a highly flexible, executable data structure to manage tasks in the shape of to-do lists or checklists in the context of KiBPs.

C. Tasks

In proCollab, a task always comprises a work description (label and an optional, detailed explanation), a current state (e.g., "in progress") and an assignment (e.g., users or roles) (cf. Section III-G) as well as optional conditions for attention (e.g., required inputs, due dates, or priorities). As tasks may either induce prospective or retrospective work (cf. Section II-C), their properties are set accordingly, e.g.: the label of a task is either to be formulated as a question (in checklists) or as a prompt (in to-do lists). To support both to-do lists and checklists, tasks may be specialized as to-dos or items in checklists. Finally, any task of course may reference necessary resources (e.g., documents) or even foreknowledge required to accomplish the task.

D. Templates

According to the KiBPs lifecycle (cf. Section II-B), templates shall enable knowledge workers to accelerate their planning and coordination through utilizing and working with best practices and standards. Based on the goals the knowledge workers want to achieve, they may choose a template and instantiate it on demand. Through instantiation, a template is transformed into a corresponding instance (cf. Section III-E) the knowledge workers can work with. In particular, there are *process*, *task*, and *task tree templates* (cf. Figure 7).

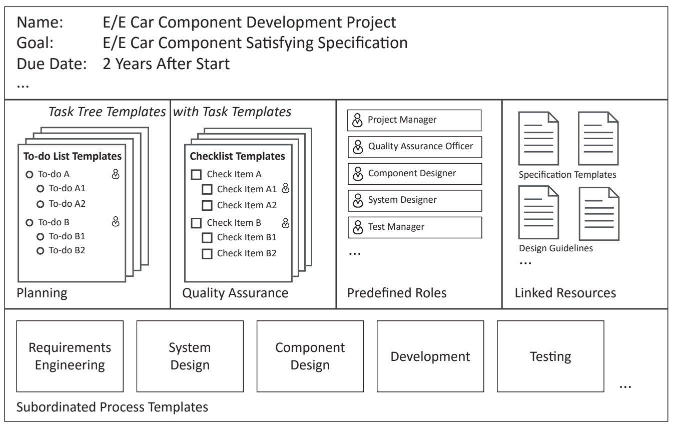

Fig. 7. Process Template Example with Task Tree and Task Templates

1) Process Templates: At the beginning of a KiBPs (e.g., a case), knowledge workers may look for a process template fitting to their goals best. A process template comprises predefined roles with corresponding rights, several specified conditions (e.g., a relative due date), linked resources, and, most important, linked task tree templates. Thus, the initial setup regarding planning is eased as knowledge workers can directly start to use (instantiated) task tree templates.

2) Task Tree Templates: A task tree template consists of task templates (cf. Section III-D3) and subordinated task tree templates. A task tree template constitutes best practices for planning (to-do list) or quality assurance (checklist) in one or several KiBPs. Thereby, a task tree template targets one or several goals in the scope of process template. As soon as a process template is instantiated, its linked task tree templates are instantiated as well. Alternatively, knowledge workers may utilize a task tree template by dynamically selecting and instantiating it in the context of an existing process instance.

3) Task Templates: A task template denotes a task that occurs in one or several task tree templates (cf. Section III-D2). A task template, in turn, may comprise several predefined conditions (e.g., duration), assignments (based on roles), and connected resources (e.g., documents). If a task template is linked in several task tree templates, an update of the task template may be performed centrally to update linked task tree templates, too.

E. Instances

At collaboration run time (cf. Section II-B), knowledge workers collaborate based on the instances of proCollab components. In particular, proCollab provides *process instances*, *task tree instances*, and *task instances* (cf. Figure 8).

1) Process Instances: A process instance represents a running project, a case, or a loose collaboration. Therefore, a process instance may contain several subordinated process instances (cf. Figure 8) enabling knowledge workers to focus on specialized sub-goals. In general, knowledge workers may create a process instance based on a process template (instantiation) or without (blank process instance). If a process template gets instantiated, the linked task tree templates are automatically instantiated and linked to the process instance. In any case, a process instance also comprises a start date, a desired duration or end date, assigned goals, and linked resources (e.g., documents).

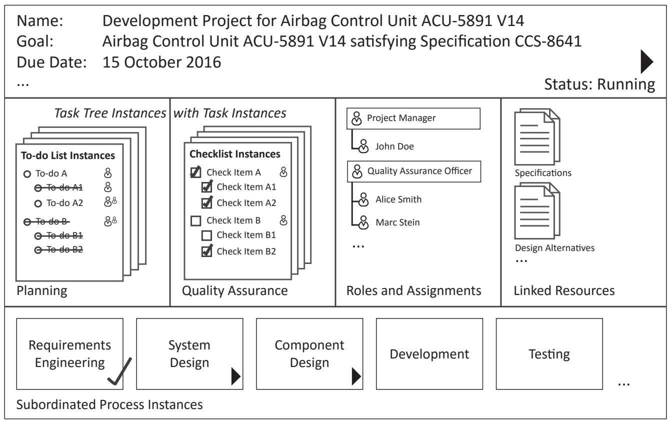

Fig. 8. Process Instance Example with Task Tree and Task Instances

2) Task Tree Instances: A task tree instance represents a task tree, e.g., a to-do list or checklist, in use. In general, knowledge workers may create a task tree instance either based on pre-specified task tree templates (instantiation) or without any predefined tasks. Task tree instances may be arbitrarily nested and comprise subordinated task tree instances as well as task instances. This means that knowledge workers may add a new task tree instance to the process instance or, as a sub-tree, to an existing task tree instance. Naturally, knowledge workers may add, update, and remove task instances and embedded task tree instances on demand to coordinate each other effectively as well as to increase work awareness in a process instance. Finally, every task tree instance is either directly linked to process instances or embedded in another one.

3) Task Instances: Task instances may be added to task tree instances (cf. Section III-E2) either based on a task template (instantiation) or without. Further, knowledge workers may update and remove task instances in a task tree instance as required. Additionally, they may perform advanced operations based on these operations, e.g., moving, splitting, and merging task instances. Additionally, knowledge workers may assign tasks to to themselves or other knowledge workers participating in the process instance to coordinate each other. Finally, task instances typically expose a state (cf. Section III-G) that can be changed by the knowledge workers.

F. Records

Based on the idea of collaboration records (cf. Section II-B), process, task, and task tree records generally consist of completed instances (i.e. instances exposing the state *completed*) as well as corresponding *change* and *execution logs*. In particular, *change logs* contain the history of applied changes (i.e., insertion, updates, and removals) whereas *execution logs* comprise the history of state changes (cf. Section III-G).

1) Process Records: A process record contains a completed process instance as well as corresponding change and execution logs. In particular, the change log of a process record contains the change history regarding the assignments of knowledge workers to the process instance, the history of instantiations or removals of task tree instances as well as the history of changes considering the conditions of the process instance (e.g., update of a due date). Finally, a process record links to all task tree records linked to the completed process instance as captured by the process record.

2) Task Tree Records: A task tree record contains a completed task tree instance as well as corresponding change and execution logs. The change log of a task tree record is of special interest as the optimization of task templates is directly based on the knowledge workers' usage of task instances (cf. Section IV). The change log first contains a history of updates applied to the task tree instance itself, e.g., updates regarding the description, assignments, and conditions. Most importantly, the change log also reflects the insertions as well as removals of task instances in the task tree. Finally, a task tree record naturally links to all task records linked to the completed task tree instance as captured by the task tree record.

3) Task Records: A task record comprises a completed task instance as well as corresponding change and execution logs. The change log starts with the time of instantiation of the task instance. Further, it comprises the history of updates of the task instance regarding its description, its assignments as well as its conditions (e.g., priority).

G. State and Organizational Models

Due to lack of space, we do not go into details regarding all proCollab component state models (indicating the operational semantics) and the entire proCollab organizational model. However, we exemplarily present a simplified state model of a task instance and an excerpt of the organisational model (cf. Figure 9) in order to foster the understanding and the presentation of the integrated proCollab task management lifecycle support (cf. Section IV).

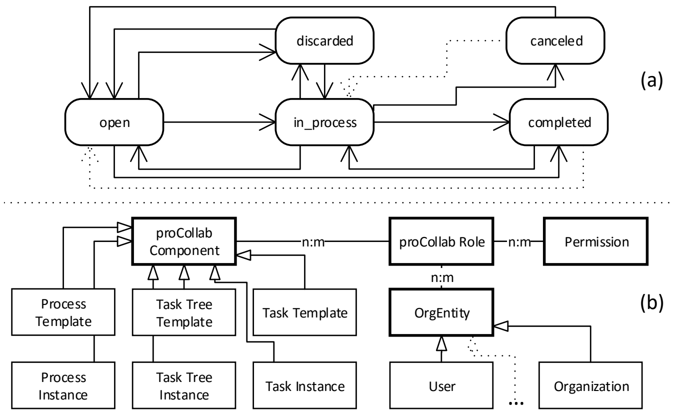

Fig. 9. State Model of Task Instances (a) and Excerpt of the proCollab Organizational Model (b)

Every proCollab component and the task instance, in particular, references a state model. The task instance exposes its current state based on the state model (cf. Figure 9 (a)). For this work, we posit a simplified model based on the states *open*, *in progress*, *discarded*, *canceled*, and *completed*. Additionally, proCollab enables the assignment of knowledge workers to tasks, task trees and process based on a powerful organizational model (cf. Figure 9 (b)). In particular, the organizational model allows for the flexible definition of *proCollab Roles* based on a set of corresponding pre-specified permissions. The latter allows, for example, accessing functions to add a new task instance or to change the state of a proCollab component.

IV. Task Management Lifecycle

Based on the presented proCollab components, an integrated task management lifecycle can be established. The latter is a prerequisite for the durable, systematic support of KiBPs and therein involved knowledge workers (cf. Section II-B). In particular, an integrated task management lifecycle necessitates the continuous optimization of provided task tree templates on the basis of the knowledge workers' usage of task tree instances created from these templates. Thereby, knowledge and efforts applied by the knowledge workers for planning and coordination are systematically preserved for comparable KiBPs. Thus, if knowledge workers are involved in such comparable KiBPs (e.g., sharing the same goal), they will directly benefit from the provision of these optimized task tree templates.

Figure 10 gives an overview of the systematic optimization of task tree templates and its different phases *preparation*, *analysis*, and *optimization*.

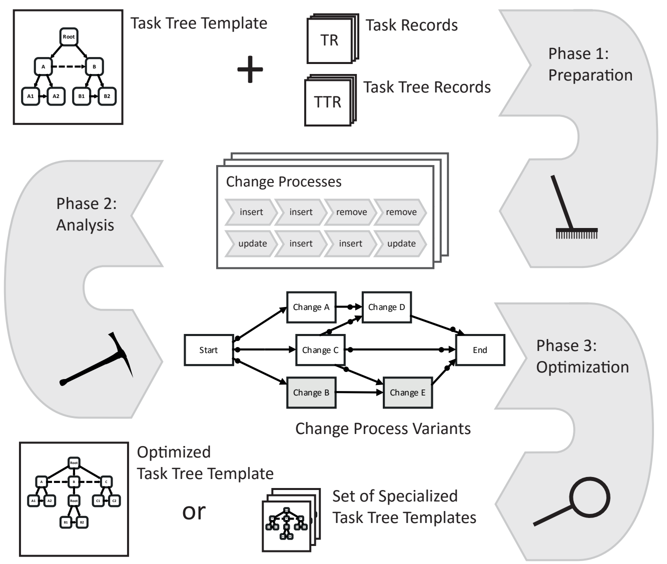

Fig. 10. Optimization of Task Tree Templates

In the *preparation phase*, the input of the optimization approach is selected and prepared. In particular, we first select the task tree template that shall be optimized (i.e., evolved) as well as the set of corresponding task tree records providing the necessary logs. Thereby, the change log of every task tree record constitutes a *change process*: a sequence of single changes applied to one particular task tree instance and its task instances by involved knowledge workers. In the ensuing *analysis phase*, we employ a process mining algorithm to identify the existing variants of the change processes in order to finally determine the most frequent changes performed on the task tree instances. Regarding the latter, we especially identify sequences of frequent changes in correlation with identified change process variants. Finally, we are able to improve the given task tree template in the *optimization phase*: a task tree template may be improved in general as well as we may also derive specialized task tree templates for certain KiBPs (i.e. process templates).

Instead of evaluating the change logs of completed task tree instances, a task tree template might be improved by the analysis and comparison of derived, completed task tree instances as well. However, through the evaluation of the change logs and the application of change mining, we are especially able to consider exactly the changes applied by the knowledge workers on the task tree instances at run time. Finally, the comparison of completed task tree instances becomes even more challenging if the latter are arbitrarily nested and highly diverse (e.g., to-do lists; cf. Section II-C).

A. Phase 1: Preparation

To optimally improve a certain task tree template ttt_x, several *preparations* have to be accomplished. Initially, the set of task tree records, which comprise the completed task tree instances originally derived from ttt_x as well as their corresponding logs, needs to be selected. The selection of task tree records directly depends on the optimization goal. To optimize ttt_x in general, all available task tree records comprising task tree instances derived from ttt_x will be leveraged for an analysis. Note that various process templates may comprise the task tree template ttt_x and, hence, an optimization of ttt_x then affects them, as well. By contrast, a task tree template may be also specialized for a better usage in one particular process template pt_y. Consequently, the task tree records linked by the process records $pr_1^{pt_y}, \ldots, pr_n^{pt_y}$ will then be solely selected for the analysis to improve ttt_x.

After the selection of a set of task tree records, the time span to be considered for the analysis is specified as well. Especially, the focus of the analysis may be set on changes applied at the very beginning of the knowledge workers' usage of the instances. Depending on the usage of the task tree instances (checklist vs. to-do list), the time span may be also modified according to the expected number of changes on these task tree instances. For example, the analysis of the task tree records may be limited to the time span starting at the instantiation time of a task tree instance until two weeks later.

For optimizing a task tree template, we consider the following changes applied to task tree instances and task instances in particular (cf. Figure 11).

1) *insertion* of a task instance into a task tree instance
2) *update* of an existing task instance in a task tree instance
3) *removal* of an existing task instance from a task tree instance

We therefore propose to map higher level operations (e.g., copying task instances) on these three basic operations. A move operation, for example, can be mapped by the sequence of insertion and removal operations. In this context note that the order of the applied changes has to be considered for an analysis since the changes are not commutative in general.

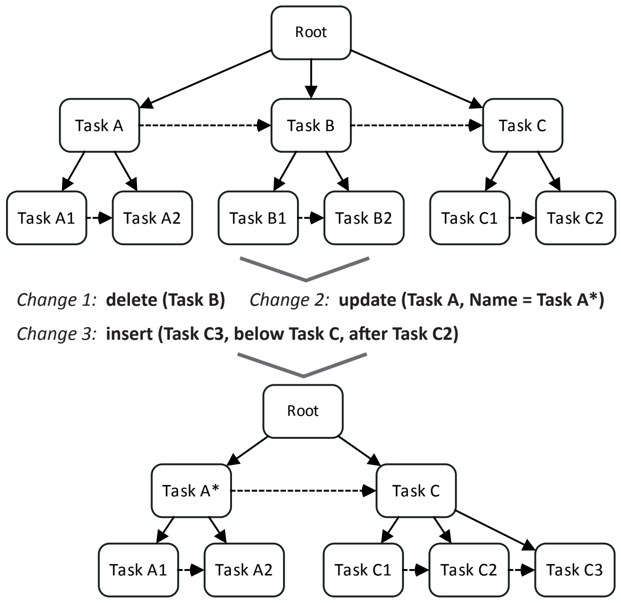
Root
Task A
Task B
Task C
Task A1
Task A2
Task B1
Task B2
Task C1
Task C2
Change 1: delete (Task B)
Change 2: update (Task A, Name = Task A*)
Change 3: insert (Task C3, below Task C, after Task C2)
Root
Task A*
Task C
Task A1
Task A2
Task C1
Task C2
Task C3

Fig. 11. Example of Changes Applied to a Task Tree

Overall, the mentioned changes on the task tree instances have to be pre-processed to make them comparable. Therefore, both the task tree records (insertions, removals) and the linked task records (updates) are combined into analyzable change processes first. As every change operation references an existing task instance of the analyzed task tree instance, update and removal operations can be well compared across change processes. However, to increase the comparability, we propose to split update operations addressing the change of several parameters of a task instance into several update operations that only change one of the parameters each. Further, we propose to remove all changes that have been undone in a change process. For instance, a knowledge worker may accidentally insert a task instance and remove it shortly after.

The comparison of inserted task instances is especially challenging: the descriptions and the meanings of the inserted task instances have to be compared somehow. If the insertion operations were not properly compared regarding their similarity, an ensuing analysis would not provide any valuable recommendations for optimizing of a task tree template. As task instances are ordered in a task tree instance, the position, where a task instance is inserted in a task tree instance, is generally determined by the parental task instance which the inserted task instance belongs to (hierarchy determination) as well as the preceding and succeeding task instances (ordering determination). Since the ordering of task instances is not prescriptive and hampers the comparison of task instance insertions in an analysis, we only regard the parental task instance as the position of an insertion for the comparison. Note that this is not a limitation (cf. Section IV-C).

In addition, other information may be leveraged to increase comparability of the insertion operations of two task instances:

- The task templates the task instances are potentially derived from,
- the descriptions (e.g., name), assignments (e.g., roles), and conditions (e.g., due dates),
- the positions the task instances are inserted at in the task tree instances,
- the process instances and the process templates the task instances indirectly belong to,
- the knowledge worker(s) (name, role, etc.) who performed the changes, and
- other changes performed before or after the inspected changes.

Based on these considerations, we posit the availability of a similarity function that particularly calculates the similarity of two comparable insertions using the above-mentioned parameters. As a result, mappings are generated for insertions showing high similarity, e.g., based on a threshold of 80%. Utilizing these mappings and the threshold, we harmonize the insertions of task instances. However, the information is preserved externally during the transformation of the change logs into the XES standard [14]. The latter is required for the process mining algorithm processing the change logs in the analysis phase.

B. Phase 2: Analysis

The objective of the *analysis* phase is to retrieve the *change process variants* applied to task tree instances. Especially, we want to detect the most frequently applied changes within the identified change process variants. Therefore, recorded change operations of the given change logs are leveraged as the input for the multi-phase mining algorithm [15]. As an output, the latter provides a causal net (CNET) [16] model comprises the applied changes as nodes. Further, it creates the most important causal relationships between the nodes. The latter are expressed by edges between the nodes as well as input and output bindings deposited at each node (cf. Figure 12).

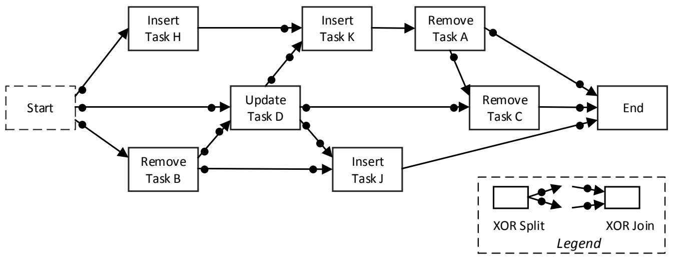

Fig. 12. CNET Example with Change Operations as Nodes

The multi-phase mining algorithm generalizes the change processes through *underfitting* the CNET model (see [17]). In particular, the CNET model does not explicitly capture interdependencies between choices at exclusive branches. Hence, the CNET contains traces not observable in the change logs. To deal with this issue, we adapted the multi-phase mining algorithm: for every node in the CNET model, a list of existing, partial traces (i.e. sequences of preceding nodes) are stored in a map. These traces have been observed by the mining algorithm before reaching this node in the change log. Thus, they can be leveraged to identify the valid change process variants in the CNET model.

To identify valid change process variants as well as filter out those ones generated by *underfitting*, the nodes of the CNET model and, especially, their input and output bindings are inspected stepwise. For each output binding of a node in the CNET model, a new variant is created and the nodes referenced by the output bindings are subsequently analyzed for each of the created variants separately. In turn, the input bindings of these referenced nodes are then checked first to verify the causal relationship in the variant. Further, we check the existence of a partial trace to this node. If there is no partial trace, the variant will be discarded immediately. In the positive case, the output bindings of the current node are again evaluated to further add nodes to be considered for a variant. Through following this approach to the end of the CNET, the valid change process variants are identified successfully.

Additionally, we modified the multi-phase mining algorithm to capture the number of change processes (i.e. change logs) represented by every change process variant. This *variant frequency* is determined to identify the most frequent changes applied within and across change process variants. Naturally, a node (i.e. a change) of the CNET model may be part of several change process variants (cf. Figure 13). Hence, we first calculate the number of variants every node in the CNET is part of. The relative *node frequency* (e.g., 10%) is then calculated for every node through multiplying this number with the variants frequency, and then dividing it with the number of change processes. The node frequency denotes how frequent a certain change was applied to the examined task tree instances derived from ttt_x.

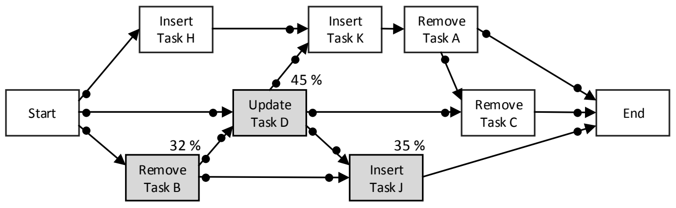

Fig. 13. CNET Example showing a Variant (grey) and Node Frequencies

Finally, sequences of frequently applied changes within and across change process variants may be identified utilizing the generated intermediate results. Therefore, different predefined thresholds denoting relative frequencies (e.g., 30 %) for insertion, update, and removal operations are used to select the most frequent changes (i.e. nodes) of the CNET model. The thresholds may be stepwise lowered until a certain number of nodes (e.g., 20 nodes) is selected in case the thresholds were set to ambitious.

Subsequently, we look for variants comprising most of identified frequent changes. For every variant found, we determine the order (sequence) of the frequently applied changes based on the causal relationships expressed in the CNET model for the variant. Thereby, we ensure that the sequences of frequently applied changes are applicable to ttt_x. We then calculate the average frequencies for every sequence of frequently applied changes—the *sequence frequencies*. Finally, the list of sequences of frequently applied changes can be ordered based on the sequence frequencies.

C. Phase 3: Optimization

Based on the results of the analysis phase, a given task tree template may be improved in general as well as specialized task tree templates may be derived for certain KiBPs (i.e. process templates). If the identified frequent changes are applied to a task tree template, we may additionally optimize the order of the existing task templates through leveraging the execution logs of the task tree records.

1) Application of Changes: Based on the identified sequences of frequently applied changes and the sequence frequencies in particular, a task tree template ttt_x can be optimized accordingly. As checklists often comprise many predefined items and changes are not supposed to be applied too often (for the sake of quality assurance), there is strong correlation or even accordance between the set of frequently identified changes and the sequence with the highest sequence frequency. In this special case, the sequence with the highest sequence frequency may be applied to ttt_x automatically.

By contrast, to-do lists are supposed to be highly modified by knowledge workers at run time. Hence, the underlying task tree templates are supposed to be rather coarse-grained allowing for the required flexibility. As a result, there is a considerably high number of changes applied to the derived task instances, and these changes are likely diverse as well. Thereby, various variants are detected by the analysis phase and, in turn, several sequences of frequently applied changes may be identified as comparable possibilities (similar sequence frequencies) to optimize ttt_x. In this case, a knowledge worker may review the recommended sequences of frequently applied changes and decides to specialize the task tree template ttt_x for certain purposes, i.e. KiBPs. In particular, the task tree templates is therefore forked and the sequences of frequently applied changes are applied separately.

Finally, the application of insertion operations on the task tree template requires the creation of new task templates. Therefore, the information deposited in the inserted task instances can be leveraged to a certain degree. However, for instance, assignments are not easily derived and, hence, a knowledge worker should likely review the inserted task templates to the end.

2) Order Optimization: After the selection of applicable changes operations on the task tree template, the ordering of the task templates in the task tree template has to be optimized as well. This is particularly required since we simplified the insertion operations: for the analysis, we only regarded the parental task instance to increase the comparability of insertion operations (cf. Section IV-A).

To successfully establish the orderings of the task templates in a task tree template, both the completed task tree instances and the executions logs as a part of the task tree records may be leveraged (cf. Figure 14). In particular, the completed task tree instances can be analyzed to determine the average position of task instances derived from the task templates to be reordered. This approach can be combined or even replaced by another, advanced approach: Based on the execution logs of the task instances and, especially, the state changes of task instances, an optimized order can be derived by the comparison of the start and the completion of task instances (cf. Section III-G) to rearrange the task templates in a task tree template.

V. Validation

The proCollab approach with its integrated task management lifecycle explicitly addresses generations of a considerable number of KiBP instances, which even often take place in sensitive businesses. Hence, the thorough elaboration of concepts and, especially, a highly mature implementation addressing further issues (e.g., a powerful, but intuitive user and role management) will be required to finally conduct valuable empirical studies to successfully validate the concepts presented in this work.

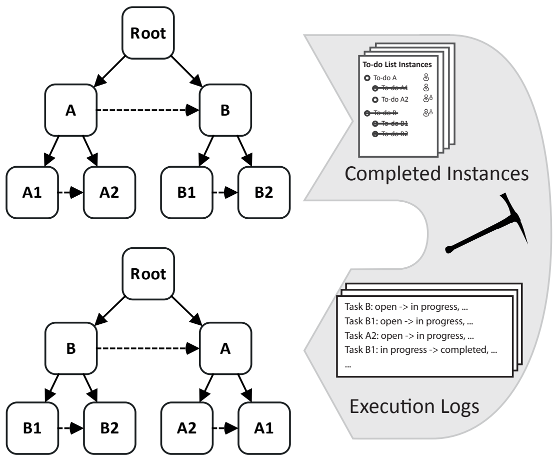

Fig. 14. Order Optimization Example for a Task Tree Template

To prepare such studies, we developed a sophisticated proof-of-concept prototype that is realized with Java Enterprise Edition 7 and further relies on a multi-layer architecture (cf. Figure 15) based on the Model-View-Controller design pattern. In particular, the application logic layer represents the core of the prototype realizing the key management services of the proCollab approach, i.e. creation, update, linkage, and removal of the proCollab key components. A REST-based interface in the communication layer enables existing web and mobile applications to communicate with the services to manage the proCollab components. Hence, knowledge workers may then use both channels to particularly manage their projects or cases (i.e. proCollab processes) including task trees in the shape of to-do lists and checklists. To validate the integrated task management lifecycle, we additionally implemented a basic similarity function (cf. Section IV-A), the modified multi-phase process mining algorithm (cf. Section IV-B), and the necessary functionality to determine the sequence of frequently identified changes.

To address the technical feasibility and scalability of our approach, we first created a reference task tree template including 20 different task templates to generate different sets of corresponding task tree instances comprising between 250 and 50,000 instances (in steps of 250) as well. Then, we simulated the usage of the task tree instances to receive usable change and execution logs in considerable time. For every set of task tree instances, a set of five change operations, i.e. insertions, updates, and removals of task instances, were particularly enforced with high probability whereas other change operations were randomly accomplished. Based on these preparations, we performed the analysis and optimization phases for every set of task tree instances and thoroughly recorded the elapsed execution time. The tests were accomplished on a laptop with an Intel dual-core CPU Intel Core i7 2640M with 2.8 GHz, 8 GB RAM, 1 TB hard disk (SATA3), and a Windows 8.1 64-bit operating system. By this procedure, we were able to show that the modified multi-phase process mining algorithm scales well with the number of provided task tree instances (cf. Figure 16). Further, we perceived the expected recommendations regarding the optimization of the task tree template as well as additional valuable insights. For instance, we could observe that the insertions of task instances subordinated to existing task instances are recognized well, whereas further insertions of more detailed task instances are less frequently identified (due to the similarity function). Hence, valuable recommendations are likely connected to a powerful similarity function and the proper selection of thresholds regarding frequently identified change operations.

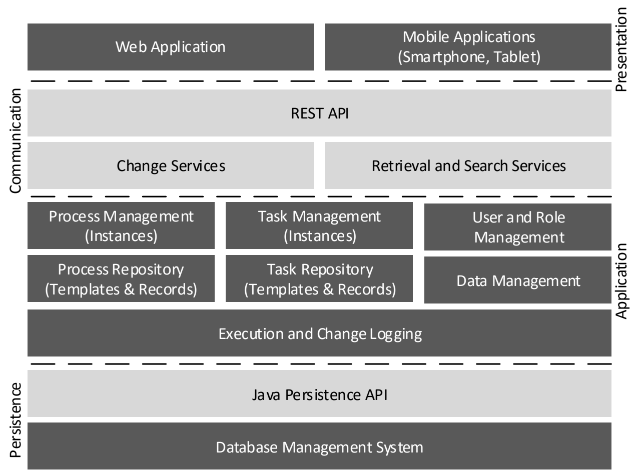
Web Application
Mobile Applications (Smartphone, Tablet)
Presentation
Communication
REST API
Change Services
Retrieval and Search Services
Process Management (Instances)
Task Management (Instances)
User and Role Management
Process Repository (Templates & Records)
Task Repository (Templates & Records)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

Fig. 15. Architecture of the proCollab Prototype

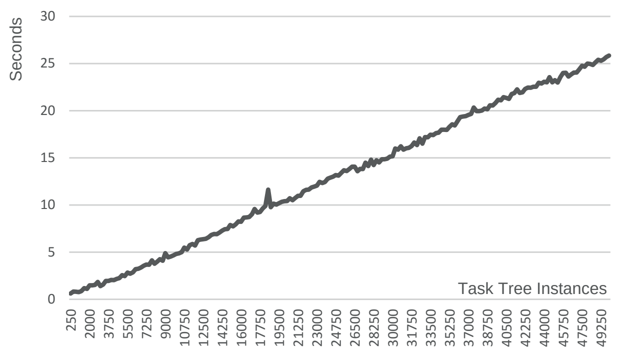

Fig. 16. Scalability of the proCollab Task Management Lifecycle Approach

VI. Related Work

In general, the roots of IT support of collaborative workers can be found in the interdisciplinary research field of *Computer Supported Cooperative Work* (CSCW) [13] and groupware in particular [18]. However, the work closely related to the proCollab approach and the optimization of task tree templates can be found in the two more recent research fields of *Adaptive Case Management* (ACM) and *Process Mining*. Originated from the Business Process Management (BPM) research, ACM can be regarded as a recently established research field [19] targeting at the systematic support of KiBPs based on the principles of *case management* and *cases* [20]. Consequently, the modeling notation CMMN[2] was developed to create, deploy, and interchange case-based specifications for the support of KiBPs [21]. As CMMN does not provide a dedicated representation for task trees and relies on various specialized case elements, proCollab does not implement CMMN. However, the proCollab components process and task can be generally compared with the CMMN elements case and task.

Another comparable approach to proCollab is presented in [22] and is called *Cognoscenti*. The latter allows for modeling and using projects including goal lists and corresponding goals. Thereby, goals are comparable to tasks whereas the approach does not provide an integrated analysis of the usage of project templates and therein comprise goal lists. In comparison, the work presented in [23] proposes a system based on CMMN and the improvement of case templates based on the analysis of case instances. In particular, the execution logs of the case instances (comparable to process instances) are analyzed based on process mining to find insights regarding the usage of CMMN-based tasks and their interdependencies. As we employed process mining to detect the most frequent changes applied to task tree instances derived from a task template, our work should be put into context of process mining in general [24] and change mining in particular [25]. Considering the latter, there is profound literature about mining frequently applied changes on standardized business process.

VII. Conclusion

Altogether, this paper presents the lifecycle-based proCollab approach enabling knowledge workers to systematically accomplish the needed task management in the scope of KiBPs. Therefore, we introduced the central proCollab components as well as the fundamental lifecycle entities. Based on the proCollab approach, we discussed the integrated task management lifecycle enabled by the systematic analysis of task tree records through process mining. Not limited to proCollab, the optimization of task tree templates based on change mining may even be leveraged in comparable approaches dealing with any kind of task trees, i.e. checklists or to-do lists.

In future research, we aim at conducting studies to further investigate the possibilities of optimizing task tree templates. In particular, we want to investigate the number of task tree records required to adequately optimize task tree templates. At the moment, we expect that the amount of task tree records depends on whether checklists or to-do lists are regarded. Based on user studies, we further want to examine whether proposed task tree optimization have been tried and trusted. Finally, another aspect we want to investigate is how often the optimization of task tree templates shall be performed to achieve the most adequate results.

References

[1] M. Pfiffner and P. Stadelmann, *Wissen wirksam machen*. Bern: Haupt Verlag, 1998.

[2] K. D. Swenson, "The Nature of Knowledge Work," in *Mastering the Unpredictable*. Meghan-Kiffer-Press, 2010, pp. 5–28.

[2]Case Management Model and Notation

[3] N. Mundbrod, J. Kolb, and M. Reichert, "Towards a System Support of Collaborative Knowledge Work," in *Business Process Management Workshops*, ser. LNBIP, vol. 132. Springer, 2013.

[4] V. Bellotti, B. Dalal, N. Good, P. Flynn, D. G. Bobrow, and N. Ducheneaut, "What a To-Do: Studies of Task Management Towards the Design of a Personal Task List Manager," in *Proc. CHI '04*, 2004, pp. 735–742.

[5] R. Pryss, N. Mundbrod, D. Langer, and M. Reichert, "Supporting medical ward rounds through mobile task and process management," *Information Systems and e-Business Management*, vol. 13, no. 1, pp. 107–146, February 2015.

[6] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[7] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007.

[8] N. Mundbrod and M. Reichert, "Process-aware task management support for knowledge-intensive business processes: Findings, challenges, requirements," in *Proc. EDOCW'14*, Sept 2014, pp. 116–125.

[9] C. Di Ciccio, A. Marrella, and A. Russo, "Knowledge-Intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches," *Journal on Data Semantics*, 2014.

[10] R. Vaculin, R. Hull, T. Heath, C. Cochran, A. Nigam, and P. Sukaviriya, "Declarative business artifact centric modeling of decision and knowledge intensive business processes," in *Proc. EDOC'11*, 2011, pp. 151–160.

[11] J. Tiedeken, M. Reichert, and J. Herbst, "On the Integration of Electrical/Electronic Product Data in the Automotive Domain," *Datenbank Spektrum*, vol. 13, no. 3, pp. 189–199, 2013.

[12] G. Hube, "Beitrag zur Beschreibung und Analyse von Wissensarbeit," Ph.D. dissertation, University of Stuttgart, Stuttgart, 2005.

[13] C. Gutwin and S. Greenberg, "A Descriptive Framework of Workspace Awareness for Real-Time Groupware," *Computer Supported Cooperative Work (CSCW)*, vol. 11, no. 3, pp. 411–446, 2002.

[14] C. W. Günther and E. Verbeek. (2014) XES Standard Definition Version 2.0. [Online]. Available: http://www.xes-standard.org/_media/xes/xesstandarddefinition-2.0.pdf

[15] van Dongen, Boudewijn F and van Der Aalst, Wil MP, "Multi-phase Process Mining: Aggregating Instance Graphs into EPCs and Petri Nets," in *PNCWB 2005 workshop*, 2005, pp. 35–58.

[16] W. van der Aalst, A. Adriansyah, and B. van Dongen, "Causal Nets: A Modeling Language Tailored towards Process Discovery," in *CONCUR 2011*, pp. 28–42.

[17] W. M. van der Aalst, V. Rubin, H. M. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther, "Process mining: a two-step approach to balance between underfitting and overfitting," *Software & Systems Modeling*, vol. 9, no. 1, pp. 87–111, 2010.

[18] C. A. Ellis, S. J. Gibbs, and G. Rein, "Groupware: some issues and experiences," *Communications of the ACM*, vol. 34, no. 1, pp. 39–58, 1991.

[19] M. Hauder, S. Pigat, and F. Matthes, "Research challenges in adaptive case management: A literature review," in *Proc. EDOCW'14*, 2014, pp. 98–107.

[20] M. J. Pucher, "The Elements of Adaptive Case Management," in *Mastering the Unpredictable*. Meghan-Kiffer-Press, 2010, pp. 89–134.

[21] OMG. (2014) Case Management Model and Notation. [Online]. Available: http://www.omg.org/spec/CMMN/1.0/PDF

[22] K. D. Swenson, "Demo: Cognoscenti Open Source Software for Experimentation on Adaptive Case Management Approaches," in *Proc. EDOCW'14*, 2014, pp. 402–405.

[23] S. Schönig, M. Zeising, and S. Jablonski, "Supporting collaborative work by learning process models and patterns from cases," in *Proc. Collaboratecom'13*, 2013, pp. 60–69.

[24] W. M. van der Aalst, *Process Mining*. New York: Springer Berlin Heidelberg, 2011.

[25] C. W. Günther, S. Rinderle, M. Reichert, and W. van der Aalst, "Change Mining in Adaptive Process Management Systems," in *Proc. OTM'06*. Springer, 2006, pp. 309–326.