Configurable and Executable Task Structures Supporting Knowledge-intensive Processes

Nicolas Mundbrod and Manfred Reichert

Institute of Databases and Information Systems
Ulm University, Germany
{nicolas.mundbrod,manfred.reichert}@uni-ulm.de
http://www.uni-ulm.de/dbis

Abstract. The operational support of knowledge-intensive processes (KiPs) constitutes a big challenge. As KiPs tend to be unpredictable and emergent, KiP execution is driven by knowledge workers utilizing their skills, experiences, and expertise. For coordination and synchronization, knowledge workers rely on simple task lists (e.g., to-do lists or checklists). Though these means are intuitive and prevalent, their current implementations are ineffective as well as error-prone: tasks are neither made explicit nor synchronized nor personalized. Furthermore, media disruptions frequently occur and no task lifecycle support is provided. Consequently, the effort knowledge workers invest in task management is not preserved for future KiPs. This work presents the proCollab approach, focusing on the generic concept of task trees. The latter enable to constitute digital task lists of any kind and to establish a task management lifecycle in the context of KiPs. Further, a configuration approach for reusable task lists (i.e., templates) is included to support knowledge workers in configuring task lists at both design and run time. proCollab is implemented as a proof-of-concept prototype and validated along a real-world use case from the healthcare domain. Overall, proCollab improves coordination and synchronization among knowledge workers, prevents media disruptions, and enables the reuse valuable coordination knowledge.

Keywords: task management, knowledge-intensive processes, knowledge workers, task lists, to-do lists, checklists

1 Introduction

Residing in highly sensitive key business areas, such as research, engineering, or service management, knowledge-intensive processes (KiPs) have become the centerpiece for creating value in many companies in recent years [2, 8]. Driving KiPs, knowledge workers make use of their distinguished skills, experiences, and expertise to cope with emerging tasks. Thus, the systematic and sustainable support of KiPs constitutes a prerequisite for achieving business goals. At the same time, a more sophisticated KiP support still poses one of the biggest challenges companies face today [3].

KiPs can be characterized as *non-predictable*, *emergent*, *goal-oriented*, and *knowledge-creating* processes [8] whose elements (e.g., activities, artifacts, or resources) cannot be foreseen a priori. KiPs have not been fully supported by contemporary process-aware information systems at the operative level so far. Instead, knowledge workers, who aim to achieve common process goals, often rely on simple, paper-based task lists (e.g., to-do lists, checklists) to define and coordinate the various activities of a KiP (cf. Fig. 1) [1]. Though paper-based task lists are intuitive and prevalent on one hand, they are error-prone and ineffective on the other. Tasks are often managed based on paper, are not explicitly represented as coordination artifacts, and are spread over different localities [12]. Thus, knowledge workers suffer from media disruptions as well as the lack of a synchronized task lifecycle support. Due to this lack, knowledge workers cannot make use of existing artifacts (e.g., task lists) when facing comparable situations, i.e. in the context of other KiPs. If knowledge workers could reuse best practice task lists and combine them on demand, redundant efforts would be significantly reduced. Likely, in turn, work quality and productivity would be increased.

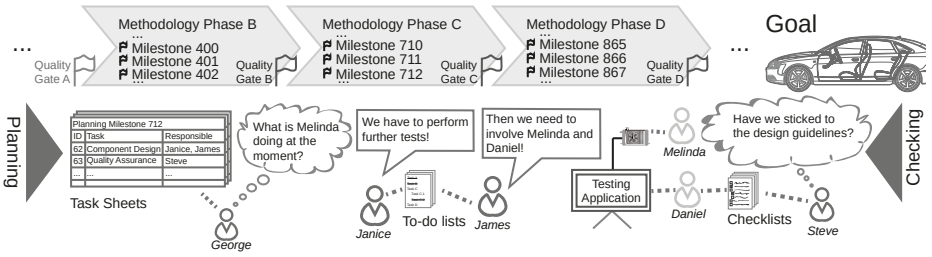

Fig. 1. Knowledge Workers Collaborating to Achieve a Goal (Automotive Domain)

In this work, we present fundamental aspects of the *proCollab*[1] approach, which aims at the systematic and sustainable support of KiPs. As tasks constitute the key entities for knowledge workers when it comes to coordination n the context of a particular KiP, but also across KiPs, proCollab provides the foundation for process- and lifecycle-based task management. In particular, it aims to empower knowledge workers to coordinate their activities among each other more effectively. To make use of best practices as well as knowledge gained in previous KiPs, proCollab encompasses the process-aware provision of *task list templates*, which knowledge workers may instantiate on demand. To foster the reuse of task list templates and to provide support for large sets of task list templates, a context-aware approach for configuring task list templates is included. This enables knowledge workers to easily configure task lists either at design or run time. Based on the proCollab approach, KiPs can be operationally supported through digital, synchronized and configurable task lists. Thereby, one can improve coordination and synchronization among knowledge workers, prevent media disruptions, and reuse valuable (process) knowledge. Finally, the feasibility of establishing an integrated task management lifecycle is demonstrated by a

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

proof-of-concept prototype. Further, the configuration approach is evaluated by applying it to a real-world healthcare scenario.

The remainder of this paper is organized as follows: Section 2 presents fundamentals and discusses key requirements. Section 3 then introduces the proCollab approach, whereas Section 4 deals with generic task lists enabling the modeling of templates and instances of different types of task lists, e.g., to-do lists or checklists. Section 4 further sketches key operations on task tree structures. Referring to these operations, Section 5 describes a flexible approach for configuring task lists, which allows knowledge workers to easily compose pre-specified task list templates. Section 6 evaluates the approach and Section 7 discusses related work. Finally, Section 8 concludes the paper and gives an outlook on future work.

2 Fundamentals and Requirements

To establish a common understanding of KiPs, this paper uses the notion of *knowledge-intensive processes* as introduced in [15]:
"*Knowledge-intensive processes are processes whose conduct and execution are heavily dependent on knowledge workers performing various interconnected knowledge-intensive decision making tasks. KiPs are genuinely knowledge, information and data-centric and require substantial flexibility at design- and run-time.*"

A detailed discussion of different KiP notions and definitions is provided in [2]. To draw attention on the challenges of a systematic KiP support and to facilitate the ensuing discussion of key requirements, we reuse an application scenario from prior work [9,14]:

> *Example 1.* In development projects for electrical and electronic (E/E) car components, the involved knowledge workers aim at developing an E/E car component before a fixed release date. Hundreds of professionals (e.g., engineers) are involved in these projects for up to several years. To ensure effective E/E development, the knowledge workers follow a development methodology with sub-goals, e.g., *quality gates* or *milestones*. Each development phase, in turn, may comprise sub-phases, as well as concurrent development processes. Hence, the knowledge workers need to frequently communicate and synchronize with each other. To ensure compliance with regulations (e.g., ISO 26262), to foster the quality of engineering processes, and to track the engineering progress, a central project *checklist* with hundreds of *check items* is initially set up and continuously managed by one or more quality assurance officers. Usually, the currently relevant check items are regularly discussed during interview with the project members. Additionally, pro-active task lists (e.g., *to-do lists* and *task sheets*) are dynamically used by the knowledge workers to manage personal tasks as well as to coordinate with each other in smaller, more specialized teams.

The presented scenario constitutes a typical example of how knowledge workers follow a methodology to cooperatively achieve a common goal as well

as to cope with the *emergent* and *unpredictable* nature of KiPs [8]. In general, respective methodologies, which are customized to a specific domain (e.g., the V model), can be abstracted by the Plan-Do-Study-Act (PDSA) cycle [8, 9] (cf. Fig. 2). We want to emphasize that collaborating knowledge workers, who follow a methodology designed for KiPs, iteratively stride through the stages of planning work, performing work, studying work results, and optimizing plans. In particular, the planning and studying stages are utilized by knowledge workers to establish efficient coordination as well as to assure KiP quality and effectiveness.

Fig. 2. PDSA-based Methodology present in Application Scenario

In the planning and studying stages, knowledge workers rely on different types of task lists as their key artifacts in use. In this context, *proactive task lists*, e.g., to-do lists, are used to dynamically plan and coordinate the various tasks emerging in the context of a KiP, whereas *retrospective task lists*, e.g., checklists, are used for quality assurance. Furthermore, both types of task lists increase *work awareness* [4], i.e, the awareness of who is doing what in the considered KiP. In prior work [12], we could observe that checklists, in practice, are not changed frequently for the sake of quality assurance, whereas to-do lists, task sheets, and similar artifacts require frequent updates, especially, the insertion of new tasks or entire sub-lists. However, in all considered application scenarios, neither checklists nor to-do lists have been supported by a KiP-aware system in an integrated, synchronized, and lifecycle-oriented manner.

To support KiPs, like the one presented in Example 1, various challenges and requirements need to be addressed. In order to design an approach that systematically supports KiPs, we conducted several case studies primarily in healthcare (e.g., ward rounds and patient treatment) and in the automotive domain (e.g., E/E engineering) [6, 8, 12, 14]. In these studies, we derived a set of key requirements [9]. In this paper, we focus on the *key requirements for enabling configurable and executable task lists* to properly support KiPs:

Meta Model (R1): A generic and expressive approach supporting KiPs must rely on a sound meta model that specifically allows for the representation of task lists of various types. Knowledge workers rely on task lists as key entities for planning, evaluating, and performing their work. Due to the emergent nature of KiPs, knowledge workers may continuously change task lists. For this use case, the meta model should provide change operations with a well-defined semantics that allow modifying a sound task list, ensuring soundness afterwards as well. To further increase the knowledge workers' efficiency and convenience, a set of high-level change operations (e.g., to swap tasks) relying on the low-level ones, are required. Finally, the trade-off between expressiveness and comprehensibility of

the meta model has to be well balanced to enable knowledge workers to seamlessly work with task lists.

Lifecycle Support (R2): In the context of a particular KiP, but also across KiPs, knowledge workers may want to use similar task lists when facing similar situations. For example, the engineering of an E/E car component requires checking functional safeness in a standardized way. To enable full lifecycle support of KiPs, therefore, the meta model needs to be enriched with an integrated and consistent support of *task list templates* and *instances* (cf. Fig. 3 a). Thereby, the introduction of task list templates allows establishing reusable artifacts of semantically connected tasks. As an example consider a checklist template with items for evaluating the functional safety of car components (cf. Example 1). During KiP execution, knowledge workers may choose a task list template, matching the given goal, needs and application context, and create a corresponding instance. To cope with the emergent nature of KiPs, in-progress task list instances may be further enhanced on demand by knowledge workers, e.g., by selecting and instantiating task list templates as subordinated task list instances. Finally, a lifecycle-based meta model relying on templates and instances provides the necessary foundation for evolving templates over time [7].

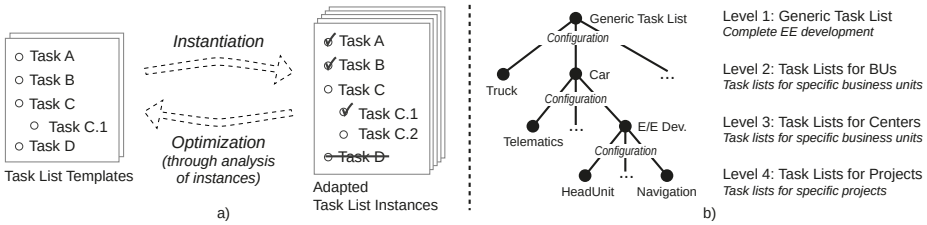

Fig. 3. Instantiation of Task List Templates and Multi-level Configuration

Configuration Support (R3): To facilitate the creation of task list templates, which may be reused in different contexts, as well as to decrease the efforts required to build up a task list, configuration support is needed. In particular, knowledge workers should be allows to configure a template in a way meeting the demands of the given application context. For example, the creation of new task list instances (e.g., checklists) may be performed by composing reusable task list templates. Additionally, configuration support necessitates the ability to remove and update existing tasks in a task list template before instantiating the latter. Generally, task list templates should be designed in a reusable and modular way to enable *multi-level configurations* (cf. Fig. 3 b). This includes the use of a generic template and the stepwise (i.e. level-based) integration of more fine-grained (i.e. specialized) task list templates to finally create the overall task list template matching the present requirements. Based on this principle, the efforts needed for creating a specific *task list variant* can be minimized significantly.

3 The proCollab Approach

The proCollab approach has been developed in the scope of a long-term research project to enable full lifecycle support for KiPs. In [8], we discussed the overall proCollab research vision, whereas [9] presented key challenges and requirements to be addressed by any KiP supporting approach. In turn, [7] introduced the key proCollab components focusing on an approach for optimizing and evolving task list templates based on the mining of existing task lists. This paper, in turn, focuses on the interplay of the key components of the proCollab meta model, its generic task trees and, in particular, an approach for configuring task lists.

To design the proCollab meta model, we specifically considered that knowledge workers repetitively perform the stages of planning work, performing work, studying work results, and optimizing plans (cf. Section 2). During these KiP stages, knowledge workers use widely established, task-based artifacts, e.g., checklists or to-do lists. Overall, proCollab relies on the key components of *processes*, *task trees*, and *tasks* to establish a framework with conceptual entities for representing KiPs as well as task-based artifacts used by knowledge workers during KiP execution (cf. Requirement R1). Moreover, to provide a lifecycle-based task management in the context of KiPs (cf. Requirement R2), processes and task trees are refined to *process templates* and *process instances* as well as *task tree templates* (with *task templates*) and *task tree instances* (with *task instances*) respectively (cf. Figure 4).

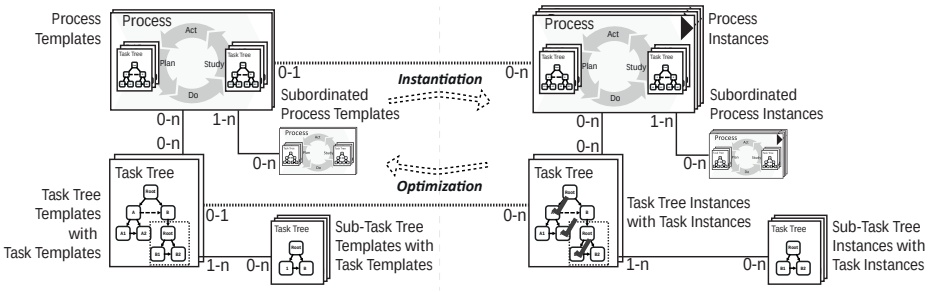

Fig. 4. Overview of the proCollab approach

Process templates and *task tree templates* shall enable knowledge workers to accelerate planning and coordination of their tasks based on best practices and standards. Before starting KiP execution, knowledge workers may retrieve a process template fitting best to their goals. Every process template may have an arbitrary number of subordinated process templates and feature various properties, conditions (e.g., a relative due date), and linked resources. Most importantly, every process template may be linked to an arbitrary number of task tree templates. A task tree template, in turn, contains *task templates* and, optionally, subordinated task tree templates. In particular, it reflects best practices for planning (to-do list) or quality assurance (checklist) in the context of KiPs. Hence, a task tree template refers to one or several goals addressed by the definition of a process template. For example, a standardized checklist for

ensuring functional safety based on ISO26262 can be well deposited as a task tree template in proCollab.

At run time, knowledge workers may collaborate in the context of specific *process instances*. A process instance may represent a running *project*, a *case*, or another *type of collaboration*. Moreover, it has properties like start date, duration, goals, and resources (e.g., documents). A process instance may further refer to subordinated process instances enabling knowledge workers to focus on specialized sub-goals. It is also noteworthy that every process instance may comprise multiple *task tree instances* (with corresponding *task instances*). In turn, a task tree instance constitutes the generic representation of common task-based artifacts in use (e.g., a to-do list). For example, an automotive E/E engineering project with to-do lists for planning and checklists for quality assurance can be properly supported by a corresponding proCollab process instance with its linked task tree instances (of type "to-do list" and "checklist"). In general, knowledge workers may create a process instance based on a pre-specified process template or may start even without any pre-specified template. If a process template gets instantiated, all linked task tree templates are automatically instantiated as well. The generated task tree instances are then linked to the process instance. Furthermore, knowledge workers may instantiate further task tree templates or add blank task tree instances to process instances on demand. Based on this flexible approach, the initial setup for the support of planning in a KiP becomes easier for knowledge workers. Finally, template concurrently promote best practice for coordination and existing process knowledge.

In practice, knowledge workers are collaborating in *projects* or *cases* as specific *types of KiPs* [8]. To support a wide range of application scenarios, proCollab incorporates type- and domain-specific specializations enabling domain- and KiP-specific customization of the generic proCollab components. For example, a proCollab process may be easily adapted to a specific automotive project regarding E/E engineering (cf. Fig. 5).

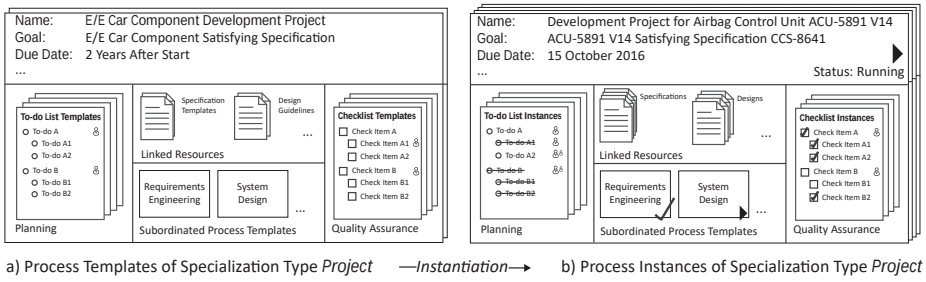

Fig. 5. Visualization of Process Templates and Instances from the Automotive Domain

Moreover, proCollab task trees may be used as a basis for supporting checklists or to-do lists at the operative level. Depending on the chosen specializations, *proCollab processes* (e.g., projects) and *task trees* (e.g., to-do lists) may feature additional properties, conditions, constraints, or assignments. To realize respective specializations, in turn, proCollab employs *specialization types* enhancing the

generic data structures of processes and task trees. For example, if a task tree template is linked to the specialization type "to-do list", it will be interpreted as a "to-do list template" with corresponding properties and an appropriate user interface representations (cf. Fig. 5). To ensure that certain specialization types are used coherently together, the specialization types can be interlinked. For example, the specialization types "to-do list" and "to-do item" may be interlinked and, hence, task trees of the type "to-do list" may only contain tasks of type "to-do item" (and none of the type "check item").

4 Task Trees

Enabling KiP support through process-related task lists and providing a solid meta model for representing the latter (cf. Requirement R1), proCollab employs the generic structure of *task trees*. In turn, a task tree includes *tasks* as well as *subordinated task trees* (cf. Fig. 6). The recommended order, in which tasks shall be processed, is specified through the *hierarchical* and *ordering edges* of a task tree. To be more precise, the pre-order traversal of any task tree directly provides its recommended sequence of tasks. To enable flexibility, however, knowledge workers may deviate from the recommended order, e.g., allowing them to deal with the current situation during KiP execution. Based on task lists relying on task trees, knowledge workers may iteratively refine coarse-grained tasks by defining more fine-grained sub-tasks. Thus, a particular task may refer to a set of subordinated tasks, which need to be completed to finish the task itself.

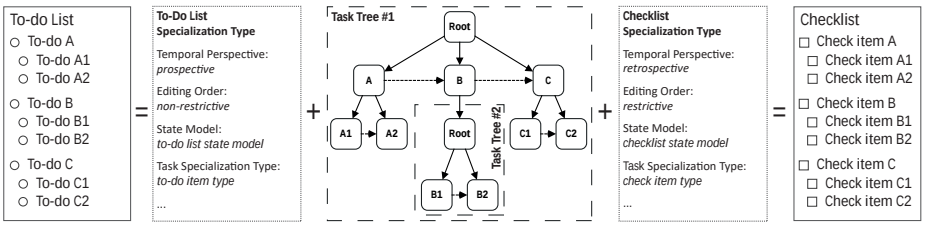

Fig. 6. Exemplary To-do List and Checklist and their Task Tree Representation

Every task tree exposes a root node with several ordered child nodes (cf. Fig. 6). The child nodes, in turn, themselves may comprise ordered child nodes. Except the root node, every task tree node either corresponds to a specific task or an embedded task tree (nesting). The root node does not correspond to a task, but may store task list properties (e.g., title, description, or purpose).

Using the conceptual model of a task tree yields several advantages. Task trees constitute an intuitive representation of common task lists. In particular, their generic and executable structure makes it possible to provide a powerful basis for both task list templates and instances as well as any concrete type of task lists, e.g., to-do lists or checklists. Furthermore, the data structure of a task tree provides a sound and common basis for defining required task tree operations (cf. Section 2). When using task lists, knowledge workers may add, update or

remove tasks and subordinated task trees on demand. Hence, a task tree is manipulable through a set of low-level operations including the *insertion*, *update*, and *removal* of task tree nodes as well as an operation to *filter* node attributes. Note that the filter operation is useful to limit the number of attributes displayed to knowledge workers. Moreover, if the filter is applied to a task tree node with child nodes, the filtering is hierarchically applied. Due to lack of space, we omit a formalization of the sketched operations. Fig. 7 illustrates the application of low-level operations to a task tree resulting in a new task tree version.

To add a task tree node to a task tree or to remove one, the respective parental node and the desired positions are required as parameters of the respective operations. As depicted in Fig 7, a particular task tree may be inserted several times, which allows for the reuse of task trees in different contexts. Note that this option is useful for task tree templates. For example, a particular task tree template with several tasks assuring quality may be embedded and reused at different spots of a parental task tree template. Consequently, a particular task tree may have several parental task trees due to its use in different application contexts. The interconnected task trees then constitute a graph of task tree nodes. Especially when inserting a subordinated task tree into an existing one, this must be carefully considered to avoid recursive nesting.

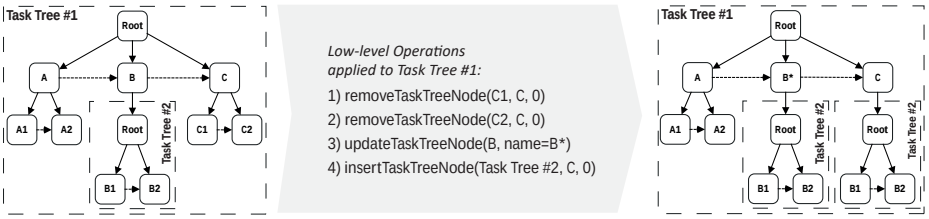

Fig. 7. Exemplary Low-level Operations Applied to a Task Tree

To ease the management of task structures, a set of high-level task tree operations is provided by proCollab. Knowledge workers may *move*, *copy*, *split* or *merge* task tree nodes. Further, they may *filter* out nodes that match certain properties. Thereby, the high-level operations are mapped to one or several low-level task tree operations. For example, splitting a task tree node involves the insertion of task tree nodes as well as the removal of the node to be split. Fig. 8 depicts the application of high-level operations on an exemplary task tree.

Relying on the conceptual model of task trees, all presented operations may be applied on both task tree templates and task tree instances no matter how they are refined by any specialization type. However, every task tree template solely consists of task templates and, optionally, subordinated task tree templates. Furthermore, every task tree template features additional template-specific properties, e.g., a specific state model. Analogously, task tree instances solely comprise task instances and subordinated task tree instances. Further, they may feature instance-specific properties and a dedicated state model, too. Based on this generic concept, proCollab supports the sound and integrated management of templates and instances of arbitrary task lists. In particular, knowledge workers

may compose, configure, and instantiate arbitrary task tree templates when starting and executing proCollab process instances.

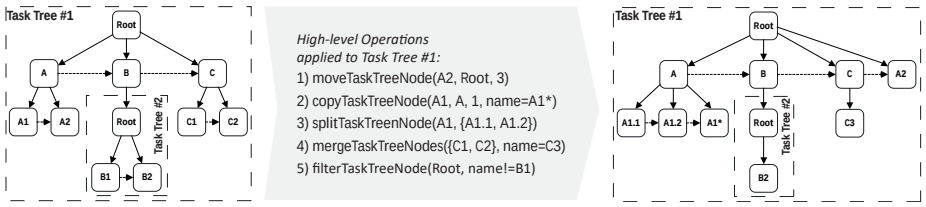

Fig. 8. Exemplary High-level Operations Applied to a Task Tree

5 Configurable Task Trees

To enable knowledge workers to efficiently configure task list templates in accordance to the given application context (cf. Requirement R3) or even to the given level of expertise involved knowledge workers expose, proCollab allows for the configuration of task tree templates. In this context, the sketched task tree operations provide the basis for a multi-level configuration of task tree templates. Furthermore, the operations enable both the combination of best practice task tree templates (e.g., inserting checklists for quality assurance) as well as the customization of task tree templates in accordance to knowledge workers' needs (e.g., filtering out non-relevant task templates).

To properly support the configuration of task tree templates, *contextual situations*, under which a task tree template might be instantiated, need to be explicitly defined. These contextual situations, in turn, may be utilized to define which operations shall be applied in which order to a task tree template during the configuration. To properly specify contextual situations, proCollab introduces *configuration parameters* each of which has a name, a pre-defined data type (*boolean*, *String*, etc.), and a value domain. Subsequently, *contextual situations* are defined by a name and a condition expressed in first-order logic relying on the set of pre-defined configuration parameters. Fig. 9 illustrates exemplary configuration parameters and contextual situations in the scope of the automotive use case (cf. Example 1) and, especially, functional safeness requirements (ISO26262) regarding E/E car component engineering.

Based on the defined contextual situations, one may provide one or more *configuration specifications* for a task tree template. A configuration specification contains a map data structure that allows assigning a sequence of *task tree operations* (applied on the respective task tree template) to every contextual situation. Fig. 10 depicts examples of task tree configuration specifications for task trees of the checklist specialization type. If a configured task tree template shall be instantiated, the currently *active* contextual situations need to be determined first. Accordingly, each defined configuration parameter obtains a value matching the defined data type.

The conditions of the contextual situations are then evaluated—a contextual situation will be considered as being *active* if its condition is fulfilled. Finally,

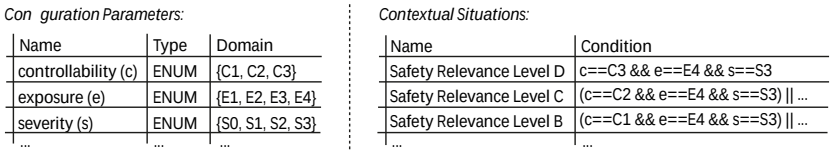

Fig. 9. Exemplary Configuration Parameters and Contextual Situations

the configuration specifications are processed in the pre-defined order. For every active contextual situation, the defined sequence of operations is applied to the task tree template. As soon as the configuration process is successfully completed, the task tree template is finally instantiated, i.e., a new task tree instance is created as the final result of the configuration.

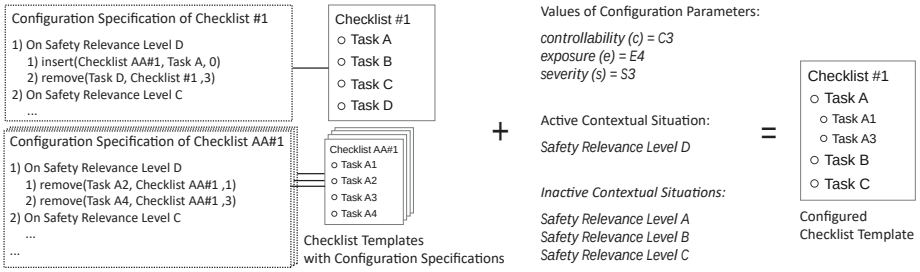

Fig. 10. Example of Multi-level Configuration Specifications for Checklists

Note that the application of task tree operations is not commutative. As a result, the order of the operations has to be carefully designed. For example, if a task tree node *A1* is inserted below an existing node A, the number of child nodes of A is consequently increased by one. Hence, one must consider this new fact for subsequent operations (e.g., more insert operations) accordingly. As a further consequence, sophisticated user interfaces are required to ensure the sound creation of configuration specifications at design time.

6 Evaluation

A mature proof-of-concept implementation is required to conduct empirical studies based on the proCollab approach. To prepare such studies and to validate the technical feasibility, we developed a sophisticated proof-of-concept prototype including the key concepts presented in this work. The prototype is realized with Java EE 7 and relies on a multi-layer architecture (cf. Fig. 11 a) based on the Model–View–Controller design pattern. The application logic layer represents the core of the prototype realizing the key services of the proCollab approach and its key components. The RESTful interface enables web and mobile applications to communicate with the services. In particular, this includes the synchronized presentation of the proCollab components across connected clients. Hence, the

user interface of the web application (cf. Fig. 11 b) enables knowledge workers to collaboratively manage their projects or cases (i.e., proCollab processes) including task trees in the shape of to-do lists and checklists.

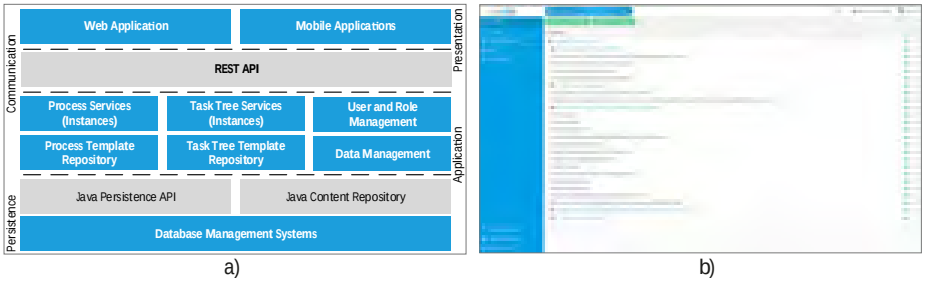

Fig. 11. Architecture and Screenshot of the proCollab Prototype

To validate the conceptual model of executable and configurable task structures, we applied proCollab to the SURPASS checklist[2] [16], which was designed for establishing a surgical patient safety system. The checklist is supposed to accompany a patient, who will get a surgery, during each step of the surgical pathway (cf. Fig. 12). In general, the checklist contains seven key parts (A0, A1, ..., E), connected to the different stages of the pathway, and two additional parts dealing with the transfer of patients (T1, T2). The SURPASS checklist features three main variants: one for clinical surgeries, one for outpatient surgeries, and one for emergency surgeries. The variants mainly differ from each other in terms of contained parts (e.g., the emergency variant omits A0) and in the number of corresponding tasks. For example, in the context of part A of the outpatient variant, a surgeon has to process five check items, whereas in part A of the emergency variant, he has to process eleven items (three being identical).

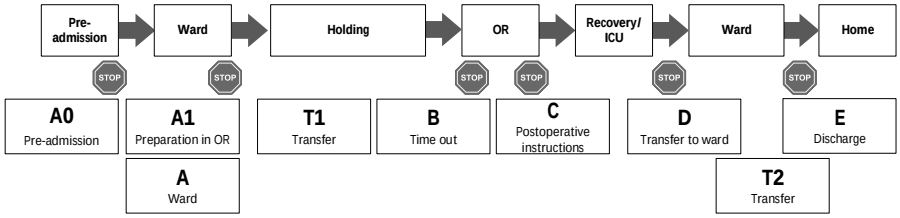

Fig. 12. SURPASS Checklist Parts in relation to Surgical Pathway

Altogether, the variants of the checklists could be well supported by the proCollab tree template configuration approach. For this purpose, we first identified the common parts shared by all variants (e.g., A1, T1) and added them to a basic task tree template of the *checklist* specialization type. Then, we modelled the individual components of the SURPASS checklist variants as separated checklist templates and included them based on the contextual situations "clinical

[2] http://www.surpass-checklist.nl/

environment", "outpatient environment", and "case of emergency". To illustrate the entire configuration process and the proper instantiation of the configurable SURPASS checklist template in detail, we refer to a created screencast[3].

7 Related Work

The roots of KiP support can be found in *Computer Supported Cooperative Work* in general and in groupware in particular [4]. The fields more closely related to proCollab are Business Process Management (BPM) and *Adaptive Case Management* (ACM) [5]. Originated from BPM research, ACM targets at the systematic support of KiPs based on the principles of *case management* and *cases*. In this context, the Case Management Model and Notation (CMMN) was developed as modeling notation to create, deploy, and interchange case-based specifications for supporting KiPs [10]. As CMMN does not provide a dedicated representation for task trees and relies on various specialized case elements, proCollab does not implement CMMN. However, its components *process* and *task* may be related to the CMMN elements *case* and *task*. Another approach comparable to proCollab is *Cognoscenti* [13], which allows modeling and using projects with *goal lists* and corresponding *goals*. In this context, goals are comparable to tasks, but the approach lacks an integrated support of templates and, especially, the generic task tree meta model. [11] introduced a notation for task models to specify a wide range of temporal relationships among tasks. The notation, which also employs a tree-based approach, focuses on the relationship between tasks and discusses the implications of temporal relationships among tasks regarding their execution. However, operations on task trees, integrated lifecycle support and configurations of task trees are not discussed in [11].

8 Conclusion

Tasks and task lists constitute the key objects for knowledge workers when it comes to KiP coordination. Consequently, the proCollab approach aims at systematic and sustainable KiP support based on integrated task management. This paper focused on the generic representation of task-based artifacts, i.e., checklists and to-do lists, through corresponding task structures. Based on the latter, KiPs can be supported through digital, synchronized, and configurable task lists. To make use of best practices and knowledge gained in similar KiPs, proCollab enables the process-aware provision of *task list templates* to allow knowledge workers to instantiate these templates on demand. To provide a context-aware support for large sets of task list templates, a corresponding configuration approach was presented to enable knowledge workers to configure task list templates on demand. Finally, the feasibility of the approach was demonstrated by a proof-of-concept prototype and its application to a use case from the healthcare domain.

[3] http://er2017.procollab.de

In future work, we will extend the proCollab approach and evaluate it in further case studies. Furthermore, the formal foundation of the proCollab meta model as well as constraints between proCollab components will be subject to future publications. Finally, we will consider the evolution of task tree templates and instances over time.

References

1. Bellotti, V., Dalal, B., Good, N., Flynn, P., Bobrow, D.G., Ducheneaut, N.: What a To-Do: Studies of Task Management Towards the Design of a Personal Task List Manager. In: Proc. CHI '04. pp. 735–742 (2004)
2. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-Intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches. J on Data Semantics 4(1), 29–57 (2014)
3. Drucker, P.F.: Knowledge-Worker Productivity: The Biggest Challenge. IEEE Engineering Management Review 34(2), 29 (2006)
4. Gutwin, C., Greenberg, S.: A Descriptive Framework of Workspace Awareness for Real-Time Groupware. CSCW 11(3), 411–446 (2002)
5. Hauder, M., Pigat, S., Matthes, F.: Research Challenges in Adaptive Case Management: A Literature Review. In: Proc. EDOCW'14. pp. 98–107 (2014)
6. Lenz, R., Reichert, M.: IT support for healthcare processes – premises, challenges, perspectives. Data & Knowledge Engineering 61(1), 39–58 (2007)
7. Mundbrod, N., Beuter, F., Reichert, M.: Supporting Knowledge-Intensive Processes through Integrated Task Lifecycle Support. In: Proc. EDOC 2015. pp. 19–28 (2015)
8. Mundbrod, N., Kolb, J., Reichert, M.: Towards a System Support of Collaborative Knowledge Work. In: BPM 2012 Workshops. LNBIP 132 (2013)
9. Mundbrod, N., Reichert, M.: Process-Aware Task Management Support for Knowledge-Intensive Business Processes: Findings, Challenges, Requirements. In: Proc. EDOCW'14. pp. 116–125 (Sept 2014)
10. OMG: Case Management Modeling and Notation (CMMN) 1.1 (2016), http://www.omg.org/spec/CMMN/1.1/
11. Paternò, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A diagrammatic notation for specifying task models. INTERACT'97 pp. 362–369 (1997)
12. Pryss, R., Mundbrod, N., Langer, D., Reichert, M.: Supporting medical ward rounds through mobile task and process management. Inf Sys and e-Business Management 13(1), 107–146 (2015)
13. Swenson, K.D.: Demo: Cognoscenti Open Source Software for Experimentation on Adaptive Case Management Approaches. In: Proc. EDOCW'14. pp. 402–405 (2014)
14. Tiedeken, J., Reichert, M., Herbst, J.: On the Integration of Electrical/Electronic Product Data in the Automotive Domain. Datenbank Spektrum 13(3), 189–199 (2013)
15. Vaculin, R., Hull, R., Heath, T., Cochran, C., Nigam, A., Sukaviriya, P.: Declarative business artifact centric modeling of decision and knowledge intensive business processes. In: Proc. EDOC'11. pp. 151–160 (2011)
16. de Vries, E.N., Hollmann, M.W., Smorenburg, S.M., Gouma, D.J., Boermeester, M.A.: Development and validation of the SURgical PAtient Safety System (SURPASS) checklist. Quality & safety in health care 18(2), 121–126 (2009)