Flexible Task Management Support for Knowledge-Intensive Processes

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

***Abstract*—Knowledge-intensive processes (KiPs) are driven by knowledge workers utilizing their skills, experiences and expertise. As KiPs are emergent and unpredictable by nature, their operational support is challenging. For coordinating and synchronizing their work, usually, knowledge workers rely on simple task lists like to-do lists or checklists. Though these instruments are intuitive and prevalent, their current implementations tend to be ineffective and error-prone. Tasks are neither made explicit nor are they synchronized. In addition, no task lifecycle support is provided and media disruptions aggravate task management. As a consequence, the efforts knowledge workers spent in task management are not exploited for optimizing future KiPs. This work presents the proCollab approach, focusing on its stateful and customizable components of processes, task trees, and tasks. proCollab processes may constitute KiPs in the shape of projects and cases, while generic task trees and tasks support required digital task lists of any kind. To enable domain-specific task support, the proCollab state management allows to integrate domain-specific procedure models (e.g., Scrum) and to enrich proCollab components with customized states. Finally, this customizable task management support fosters knowledge workers' coordination, increases work awareness, reduces media disruptions, and enables the reuse of valuable coordination efforts and knowledge.**

Keywords*—*task management, knowledge-intensive process, task list, checklist, to-do list, knowledge worker

I. Introduction

Residing in sensitive key business areas like research, engineering, or service management, knowledge-intensive processes (KiPs) have become the centerpiece for creating value in many companies [1], [2]. Driving KiPs, knowledge workers make use of their distinguished skills, experiences and expertise. Thus, the systematic support of knowledge workers in the context of KiPs is a prerequisite for achieving business goals. Note that such KiP support poses one of the biggest challenges companies are facing today [3].

KiPs can be characterized as *non-predictable*, *emergent*, *goal-oriented*, and *knowledge-creating* processes [1]. In general, their elements (e.g., activities, resources, or artifacts) cannot be foreseen a priori. As a consequence, KiPs have not been fully supported by any contemporary information systems at the operational level so far. Instead, most knowledge workers still use simple, paper-based task lists (e.g., to-do lists, checklists) to cope with more sophisticated KiP tasks as well as to coordinate the various KiP activities [4]. Though paper-based task lists are intuitive and prevalent, so are they error-prone and ineffective. Moreover, tasks miss an explicit representation as coordination artifacts as well as they are not provided in a personalized manner and often spread over different localities [5]. As a consequence, knowledge workers lack a stateful, synchronized and lifecycle-based task management support for KiPs. The task states and their synchronization with the work progress are crucial for knowledge workers to efficiently perceive work awareness (who is doing what) and to effectively plan ongoing activities of a KiP. The lack of a lifecycle-based task support, in turn, thwarts the reuse of existing artifacts (e.g., task lists) in similar KiPs. If knowledge workers can utilize existing task lists representing best practices, and modify or combine them on demand, redundant planning and coordination efforts will be reduced significantly.

To enable a systematic and sustainable support of KiPs, we introduced the *proCollab*[1] approach in [6]. As tasks constitute the key entities for knowledge workers when it comes to coordination in KiPs, proCollab provides the foundation of a process- and lifecycle-based task management. Especially, it provides *process* and *task list templates*, which may be *instantiated* by knowledge workers on demand. Thereby, the templates enable knowledge workers to make use of best practices and knowledge gained in comparable KiPs. This paper especially focuses on the flexible state management and customization concepts provided by proCollab. In particular, these concepts allow for the support of different types of KiPs, task lists, and tasks including their domain-specific requirements (e.g., states, properties, constraints). Further, these concepts may increase work awareness and empower knowledge workers to collaborate and to plan their work more effectively. The support of an integrated, stateful task management lifecycle is demonstrated by a proof-of-concept prototype. Moreover, the approach is evaluated along a real-world scenario from software engineering. Altogether, the proCollab approach will improve coordination and synchronization among knowledge workers, prevent media disruptions, and foster the reuse of coordination knowledge.

The remainder of this paper is structured as follows: Section II presents fundamentals and introduces an application scenario. Section III discusses the proCollab key components. Section IV then presents the generic state management concept for supporting a variety of KiPs with proCollab. On this basis, Section V deals with the proCollab specialization concept, which refines proCollab components to provide templates and instances of different types of KiPs (e.g., projects or cases) and task lists (e.g., to-do lists or checklists). Sect. VI evaluates the approach and Sect. VII discusses related work. Finally, Sect. VIII summarizes the paper and gives an outlook.

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

II. FUNDAMENTALS

While [2] provided a detailed discussion of different KiP notions and definitions, this paper uses the notion of *knowledge-intensive processes* as introduced in [7] to establish a common understanding of KiPs: *"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."*

To systematically examine how knowledge workers collaborate in the context of KiPs and how they accomplish their tasks, we studied characteristic application scenarios (e.g., automotive engineering and healthcare) to derive key challenges and requirements for the support of KiPs [5], [8]. Due to lack of space, we cannot recap the full set of requirements. To still draw attention to the challenges and requirements of a stateful, lifecycle-based support for KiPs, we introduce an application scenario from the automotive domain (cf. Fig. 1) [9]:

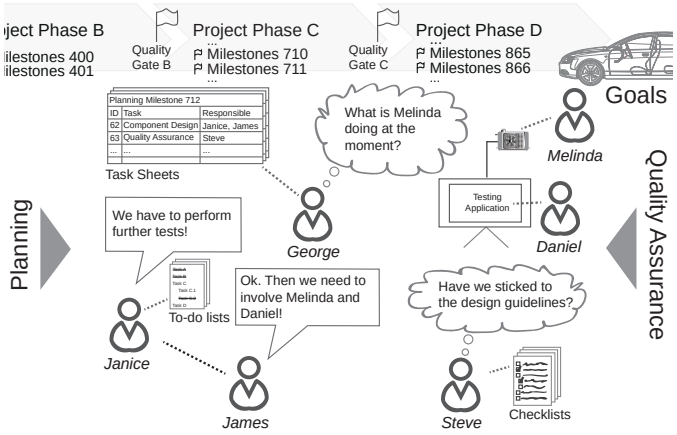

Fig. 1. Application Scenario

Example 1: While engineering electrical and electronic (E/E) car components, the involved knowledge workers aim to develop 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 development, the knowledge workers follow a methodology with sub-goals, e.g., *quality gates* or *milestones*. Each development phase may comprise subphases and 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 increase 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 quality assurance officers. Usually, the currently relevant check items are regularly discussed with the project members to provide KiP guidance. 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 tasks within small specialized teams.

Example 1 constitutes a characteristic example of how knowledge workers collaborate and apply a methodology to achieve a common KiP goal. Methodologies are used to cope with the *emergent* and *unpredictable* nature of KiPs [1]. However, methodologies are often specific for certain domains (e.g., the V model). In general, they can be abstracted by the generic Plan-Do-Study-Act (PDSA) cycle [1], [8] (cf. Fig. 2): Collaborating knowledge workers iteratively stride through the stages of planning work, performing work, studying work results, and optimizing work plans. The planning and studying stages, in particular, are utilized to establish efficient coordination as well as to assure quality and effectiveness. Furthermore, in these stages, knowledge workers rely on task lists of different types as their key artifacts in use (cf. Fig. 2). *Proactive task lists* (e.g., to-do lists) are used by knowledge workers to dynamically plan and coordinate the tasks emerging in a KiP, whereas *retrospective task lists* (e.g., checklists) are used for quality assurance. Moreover, both proactive and retrospective task lists increase *work awareness* [10], i.e., the awareness of who has been doing what in the context of a specific KiP.

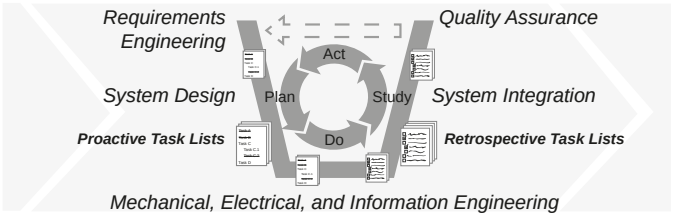

Fig. 2. V Model: PDSA-based Methodology of Application Scenario

The interrelationship of KiPs and tasks, which are part of task lists, is crucial. *A task describes a piece of work to be performed within a certain time*. Through task lists, knowledge workers iteratively refine (coarse-grained) tasks by defining more detailed sub-tasks. Hence, a certain task may be connected to an arbitrary set of subordinated tasks, which are supposed to be successfully performed in order to complete the actual task itself. In the context of a KiP, tasks are continuously defined, updated, and, finally, performed by knowledge workers. Regarding the performance of tasks, three basic cases exists (cf. Fig. 3). First, a task may be performed by one or more knowledge workers autonomously (*atomic task*). Second, knowledge workers may collaboratively accomplish a task in the scope of a subordinated KiP, which may comprise other tasks (*process task*). Third, a task may be performed by any kind of external, possibly standardized, process (*standardized process task*). In this case, the task commonly expresses that the external process needs to be triggered by knowledge workers in future and that the result is of interest for the KiP.

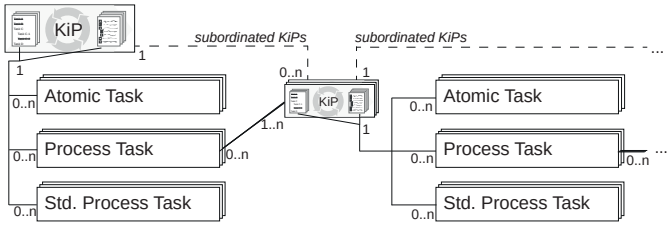

Fig. 3. Interrelationship of KiPs and Tasks

Finally, we want to emphasize that the *current state* of a KiP is strongly connected to the current states of its tasks. In turn, the states of process tasks are directly connected to the states of the triggered sub-processes. Hence, the synchronized states of KiPs and tasks are of high importance for knowledge workers as they provide the basis for effective planning, quality assurance, and work awareness. As a consequence, a task management support for KiPs must consider the *representation of the different states of processes and tasks as well as the proper synchronization of theses states*. Note that this is challenging as knowledge workers usually apply domain-specific methodologies, necessitating the integration of domain-specific procedure models (e.g., V model) and states of KiPs, task lists, and tasks.

III. ProCollab Key Components

To enable the systematic support of KiPs, we developed the proCollab approach (cf. Fig. 4) [6], which provides a lifecycle-based task management support for KiPs. Specifically, proCollab takes into account that knowledge workers repetitively perform the stages of planning work, performing work, studying work results, and optimizing plans (cf. Sect. II). In these stages, knowledge workers use task-based artifacts to achieve common goals. Examples of such artifacts include to-do lists and checklists. Consequentially, proCollab relies on the key entities *processes*, *task trees*, and *tasks* to realize a framework for representing KiPs and the task-based artifacts used by knowledge workers for properly coordinating their work. To enable a lifecycle-based task management for KiPs, proCollab 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. To support the domain-specific requirements of knowledge workers, proCollab components expose states based on a flexible state management (cf. Sect. IV) as well as they can be specialized (cf. Sect. V) to make these generic components as specific as required.

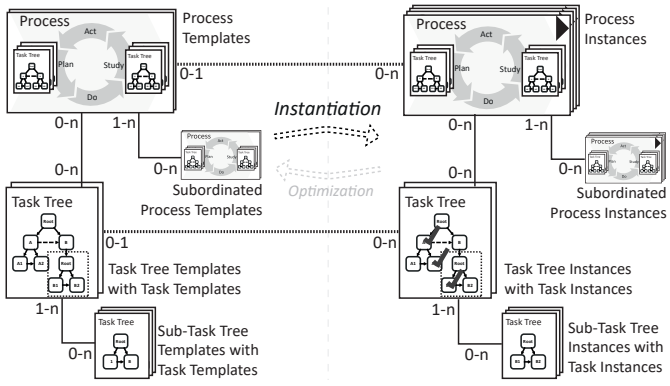

Fig. 4. Overview of proCollab Components

In practice, knowledge workers collaborate in the scope of *projects*, *cases*, or *temporary endeavors* [1]. To generalize from these terms, the notion of *process* is used. Thereby, every process may be arbitrarily nested (e.g., sub-projects) and expose a goal, the knowledge workers want to achieve. Furthermore, proCollab processes are refined into the lifecycle components *process templates* and *process instances*. Finally, every process may be linked to various *task trees* whose tasks enable knowledge workers to better plan and check their work. Task trees enable KiP support through process-related task lists and provide a solid meta model for representing the latter. A task tree includes *tasks* and *subordinated task trees* (cf. Fig. 5). Every task features a work description, a current state (cf. Sect. IV), and a set of constraints (e.g., required inputs, due dates). Knowledge workers, using task lists (based on task trees) may iteratively refine coarse-grained tasks by adding more fine-grained sub-tasks.

A task tree exposes a root node with ordered child nodes (cf. Fig. 5). The child nodes, in turn, themselves may have ordered child nodes, and so forth. Except the root node, every task tree node either corresponds to a specific task or an embedded task tree (nesting). The recommended order, in which tasks of a task tree shall be processed, is specified by *hierarchical edges* and *ordering edges*. Hierarchical edges constitute the hierarchical relationship of task tree nodes, whereas ordering edges define the order of task tree nodes on the same tree level. Consequently, the pre-order traversal of a task tree provides the recommended task sequence. When deriving to-do lists or checklists from task trees, knowledge workers may perform various low-level operations on a task tree on demand, e.g., adding, updating or removing tasks and subordinated task trees respectively. Further, they may perform high-level operations based on these low-level ones, such as moving, splitting, and merging tasks within or across task tree instances. The proCollab framework encompasses a well-defined set of operations for manipulating task trees, i.e., task tree templates and instances. Fig. 5 illustrates the application of four basic operations to a task tree.

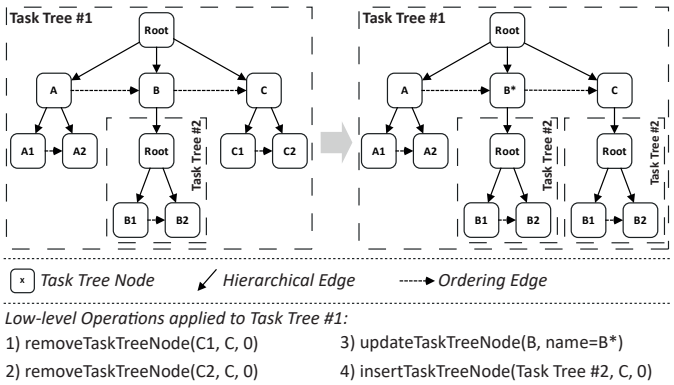

Fig. 5. Exemplary Task Tree Manipulated by Basic Operations

proCollab *process templates* and *task tree templates* shall enable knowledge workers to accelerate the planning of their tasks based on best practices or domain-specific standards. As a proCollab process, every process template may have an arbitrary number of subordinated process templates and feature various properties, constraints, and linked resources. Most importantly, every process template may be associated with task tree templates. A task tree template, in turn, consists of *task templates* and, optionally, subordinated task tree templates. Furthermore, every task tree template contains a description of its purpose and pre-defined properties for corresponding task tree instances. A task template, in turn, denotes a task that occurs in one or several task tree templates. It may comprise constraints (e.g., duration), assignments (based on roles), and linked resources (e.g., documents) as well. If a subordinated task tree template or a task template is linked to several,

parental task tree templates, any update of these components indirectly updates linked task tree templates as well. In general, a task tree template and its task templates reflect best practices for planning (to-do list) or quality assurance (checklist) in accordance to the PDSA cycle. For example, a standardized checklist ensuring functional safety (e.g., ISO26262) may be specified as a task tree template. Overall, process templates may ease the initial setup regarding KiP planning as knowledge workers can directly start to use (instantiated) process and task tree templates.

At run time, knowledge workers may collaborate in the context of proCollab *process instances*. A process instance represents a *running* project, case, or collaboration. Additionally, it may contain several subordinated process instances to focus on specialized sub-goals of the KiP. Further, it has a start date, a desired duration or end date, assigned goals, and linked resources (e.g., documents). Importantly, every process instance may comprise multiple *task tree instances* with corresponding *task instances*. A task tree instance, in turn, corresponds to the generic representation of a common task list (e.g., a to-do list) in use. For example, an automotive E/E engineering project with to-do lists for planning and checklists for quality assurance can be supported by a corresponding proCollab process instance with two or more associated task tree instances with types *to-do list* and *checklist* (cf. Sect. V-A). In general, knowledge workers may create a process instance based on an existing process template or starting without any template. If a process template gets instantiated, all linked task tree templates also become automatically instantiated, and the generated task tree instances are linked to the process instance accordingly. Furthermore, when executing a proCollab process instance, knowledge workers may dynamically instantiate task tree templates or add new, initially empty task tree instances on demand. Furthermore, they may add, update, and remove task instances as well as embedded task tree instances on demand. Additionally, they may perform high-level operations based on these operations, e.g., to move, split, or merge task instances. Finally, knowledge workers may dynamically assign tasks to themselves or other knowledge workers participating in the process instance. Based on this flexibility, planning efforts can be significantly eased, best practices be promoted, and process knowledge be efficiently adopted by knowledge workers.

IV. proCollab State Management

To support a wide range of KiPs with their methodologies (cf. Sect. II), proCollab incorporates a generic state management concept for its stateful key components, i.e., process templates, process instances, task tree templates, task tree instances, task templates, and task instances. In particular, this concept enables us to integrate common domain-specific methodologies as well as to manage different types of proCollab components in a controlled manner. For example, the V model, which was used in the automotive engineering project presented in Sect. II, refines the key phases of this PDSA cycle to match the specific requirements in engineering. Furthermore, and this aggravates proper state management significantly, some of the tasks defined in *Example 1* require different specialized states (e.g., "completed with approval") in comparison to tasks used in other domains, e.g., software engineering. To reflect such domain-specific variations, the proCollab state management employs *reference state models*, *state models*, and *state model instances* (cf. Fig. 6). While a reference state model is declaring the states and state transitions the entities of a particular type share, a state model may refine a reference state model to meet domain-specific requirements. Finally, every stateful proCollab component references exactly one state model instance relying on a preselected state model.

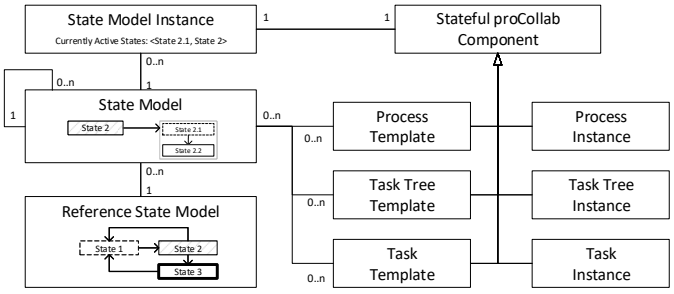

Fig. 6. proCollab State Management Entities

A. Reference State Models

A *reference state model* consists of a *state transition graph*, a scope, and a set of refinable states. Thereby, a *state transition graph* corresponds to a *finite-state machine*, i.e., it consists of *states* and state *transitions*. Further, it exposes a starting state and one or more final states. In general, every state is defined by a label and a *state type*. In turn, a transition is defined as an edge leading from a source to a target state. A transition between two states constitutes the pre-specified way to leave the source state to enter the target state. Fig. 7 illustrates the reference state models of the key proCollab components.

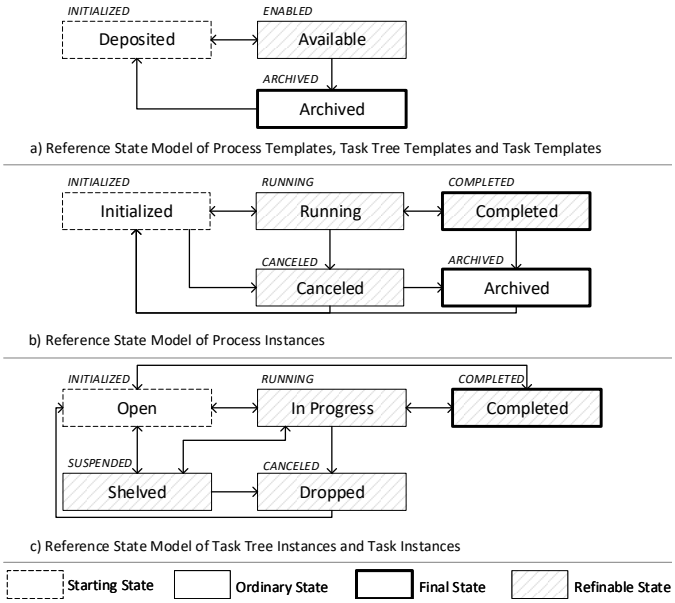

Fig. 7. proCollab Reference State Models

The state types, in turn, are required as the proCollab framework shall provide functionality with respect to the current states of proCollab components. For example, knowledge workers may only instantiate *enabled* process templates, or they may solely retrieve *completed* task instances. For this purpose, state types allow distinguishing different states based on their semantics and independent from

their labeling. To provide such a customized support, we defined the following state types (cf. Sect. V): $INITIALIZED$, $ENABLED$, $RUNNING$, $COMPLETED$, $CANCELED$, $SUSPENDED$, $ERROR$, and $ARCHIVED$. While the semantics of most state types can be intuitively perceived (cf. Fig. 7), it is important to explain that $INITIALIZED$ expresses that the proCollab component has been created and is always used for all starting states. Furthermore, state type $ENABLED$ signs enabled proCollab templates, i.e., the latter are available for instantiation. The scope of a reference state model determines for which proCollab component the model may be used. For example, if the scope is 'Process Instance' (cf. Fig. 7b), the reference state model constitutes the basis for all state models of process instances. Consequently, a specific reference state model comprises the states shared by all stateful proCollab components of a certain type (e.g., all process instances). To allow for the domain-specific customization of proCollab, a reference state model exposes states that may be further refined through derived state models.

B. State Models

A *state model* is linked to a reference state model and may optionally expose a parental state model (cf. Fig. 6). Thereby, a state model inherits the scope of the corresponding reference state model. Hence, it may be solely used for entities matching the given scope (e.g., process instances). In addition, a state model may refine those states of the reference state model that are marked as refinable. For this purpose, one has to design a state transition graph and then assign it to any refinable states. Fig. 8 shows an example of a state model that refines the reference state model of a process instance presented in Fig. 7. In particular, Fig. 8 illustrates how the individual states of the V model may be incorporated to provide a corresponding state model for process instances in proCollab.

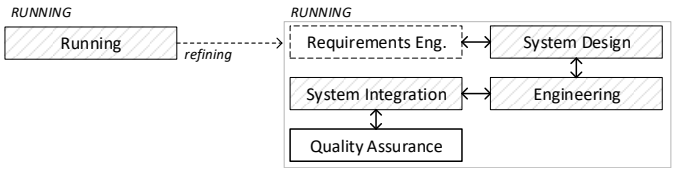

Fig. 8. Individual State Model for proCollab Process Instances

To further increase flexibility, a state transition graph refining a given state may contain refinable states itself. A state model, which is linked to a parental state model, may therefore have state transition graphs refining the refinable states of the parental state model. For example, the state model depicted in Fig. 8 may be refined by child state models to further customize the three refinable states. However, proCollab state models must not override any refinement already provided by a parental state model.

C. State Model Instances

State model instances make proCollab components stateful. Every state model instance references a state model and it exposes a sequence of *currently active states* as well as a constraint named *strict*. When creating a stateful proCollab component, one must select a corresponding state model. Based on the latter, a state model instance is created and linked to the stateful proCollab component. Consequently, every proCollab component features a state model instance referencing a state model, that, in turn, relies on a reference state model (cf. Fig. 9). To ease the selection of a state model, each proCollab template component may reference a state model supposed to be used for the derived instances. For example, a process template may reference a specific state model with scope "Process Instance". If knowledge workers instantiate this process template, the linked state model will be used to create a corresponding state model instance for the new process instance. Note that the selection of a specialization type may limit the number of state models that may be linked to a specific template (cf. Sect. V-A).

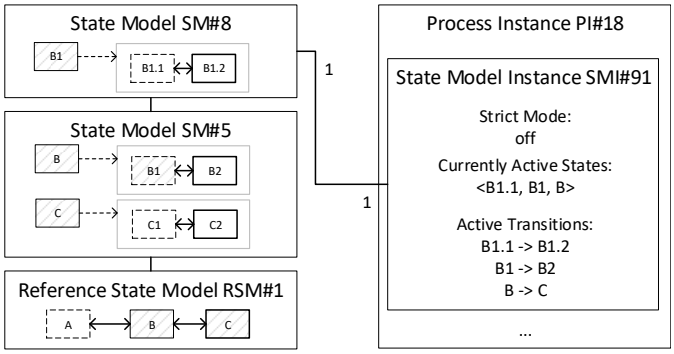

Fig. 9. Exemplary State Model Instance with Corresponding State Model

To manage the currently active states of a state model instance of a stateful proCollab component, it is important to recap that states may be refined by a state model (cf. Sect. IV-B). Hence, a state model instance exposes a sequence of active states (cf. Fig. 9)—from the most specific to the most abstract one, i.e., the one belonging to the reference state model. As every state transition graph has exactly one starting state, the state model instance is initialized with the starting state of the state transition graph of the reference state model. Alternatively, if the starting state is refined, a sequence of starting states, which belong to state transition graphs refining the corresponding states, will be used.

If the *strict* constraint is not set, knowledge workers may use *any* outgoing transitions of the currently active states to switch to another active state. For example, the state model instance depicted in Fig. 9 allows knowledge workers to select states $B1.2$, $B2$, or C. However, to support KiPs in which knowledge workers shall strictly follow the sequence of predefined states of a state model (including all refined states), the *strict* constraint may be set to *active*. In the latter case, for a state model instance, the outgoing transitions of refined states will be only usable if the refining state transition graph exposes a final state. If a final state has not been reached yet, knowledge workers must first use one or several transitions via the refining state transition graph to reach a final state. Consider Fig. 9: knowledge workers will only be able to change the state model instance to state $B1.2$ if *strict* constraint is *active*. In the latter case, knowledge workers will be only able to switch to state *B2* if final state *B1.2* of the refining state transition graph is set first. Thereby, the interaction of knowledge workers with proCollab components can be controlled to meet domain-specific requirements of the given application scenarios at hand (e.g., following the phases of a pre-specific methodology). Note that proCollab automatically enables the starting state

of a refined state as soon as this state becomes active. For example, if knowledge workers change the state model instance from state B to state C, state $C1$ will be automatically added to the sequence of currently active states; the latter will contain accordingly: $\langle C1, C\rangle$). The reason is that C is refined in the state model $SM1$ by a state transition graph containing $C1$ and $C2$, and $C1$ is marked as starting state.

V. proCollab Specialization and Customization

Overall, the proCollab state management provides a sophisticated and flexible foundation to enable customizable support of a wide range of KiPs. Nonetheless, the full potential of the presented state management concept is exploited as soon as the generic proCollab components are specialized and customized. In the automotive domain, e.g., knowledge workers (i.e., engineers) collaborate in the scope of projects, rely on a engineering methodology (e.g., V model), and use task sheets and checklists with specialized states for coordinating their work. By contrast, in a hospital, physicians take care of patients in the context of cases, follow the *diagnostic/therapeutic lifecycle*, and employ patient safety checklists [11], [5].

A. Specialization Types

To allow for the type- and domain-specific specialization and customization, proCollab employs *specialization types* enhancing the generic data structures of processes, task trees, and tasks (cf. Fig. 10). Accordingly, the most common types are *process types*, *task tree types*, and *task types*. Further, we pre-defined six specialization types: The process types *project* and *case* are usable to specialize process templates and instances. Furthermore, the task tree types *to-do-list* and *checklist* may refine task tree templates and instances, whereas the task types *to-do item* and *check item* may be used for specializing task templates and instances.

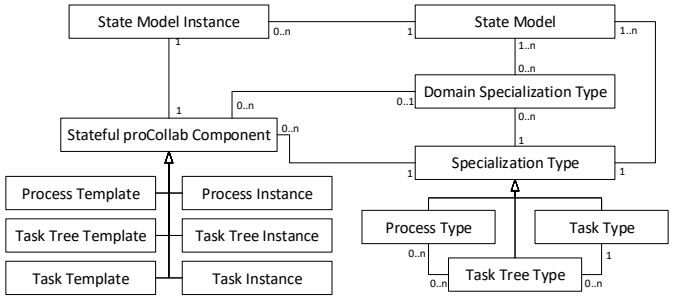

Fig. 10. proCollab Specialization Entities

Every specialization type may refer to a set of state models that can be used for templates and instances (depending on the scope of the reference state model). For example, the *project* process type may refer to multiple state models with scope "Process Instance" realizing common procedure models (e.g., the V model). If a knowledge worker wants to create a process instance based on the *project* process type, she may select one of the provided state models and, subsequently, retrieve a project relying on the chosen state model. As a result, the process of selecting an appropriate state model is eased significantly and knowledge workers are enabled to more effectively create customized process instances. To integrate specialization types with the phases of the PDSA cycle, every specialization type exposes a *temporal perspective*. The latter denotes whether proCollab components, which rely on a specific specialization type, may be used for planning (*prospective* temporal perspective), for quality assurance (*retrospective* temporal perspective), or for both aspects (*hybrid* temporal perspective). For example, the *to-do-list* task tree type and the *to-do item* task type both expose the *prospective* temporal perspective, whereas the *checklist* and *check item* types expose the *retrospective* one. To ensure that certain specialization types are coherently used together (e.g, based on their common temporal perspective), the types may be interlinked (cf. Fig. 10). For example, the *to-do-list* task tree type is interlinked with the *to-do item* task type. Hence, task trees of type *to-do-list* may only contain tasks of type *to-do item* (and none of type *check item*).

Depending on the chosen specialization type, a proCollab component may feature additional properties and constraints (e.g., editing order). If a task tree instance is linked to the *to-do list* task tree type, e.g., it will be interpreted as a "to-do list instance" with corresponding properties and an appropriate representation. Moreover, it will be treated as a prospective task list to be used by knowledge workers for planning work (cf. Fig. 11). By contrast, if the same task tree instance is linked to the *checklist* task tree type, it will be interpreted as retrospective checklist used for quality assurance. Finally, to enable further refinement of specialization types, proCollab employs *domain specialization types* (cf. Fig. 10). In general, a domain specialization type relies on an existing specialization type and may feature additional properties and constraints. Most notable, a domain specialization type may refer to a set of state models, which further narrows the set of state models belonging to the specialization type. For example, a domain specialization type *Automotive Project* may limit the set of possible state models for process instances to one reflecting the V model solely.

B. Task and Process Synchronization

Tasks can be distinguished regarding the way they are performed (cf. Sect. II)—there are atomic tasks and composite ones. The latter either require subordinated tasks of any kind, the performance of one or more KiPs in order to accomplish the task, or the performance of one or more standardized processes for completion. As a consequence, proCollab introduces *task execution types* to enable the tight coupling of tasks and subordinated processes based on the presented state management concept. If a task instance requires the creation and completion of one or even several KiPs, it is linked to a corresponding *process task execution type*. The latter allows synchronizing the task instance with one or more linked process instances. For this purpose, the process task execution type may refer to a process template, which is supposed to be instantiated by knowledge workers to actually perform the respective task. If no process template is deposited, a process instance may be dynamically created and linked to the process task execution type. To increase work awareness, the progress of the linked process instance(s) is then synchronized with the current state of the task instance. Therefore, the reference state models of tasks and processes are utilized as they allow for a state mapping. In particular, if only one process instance is linked to the task instance via a process task execution type, the state of their reference state models can be directly mapped

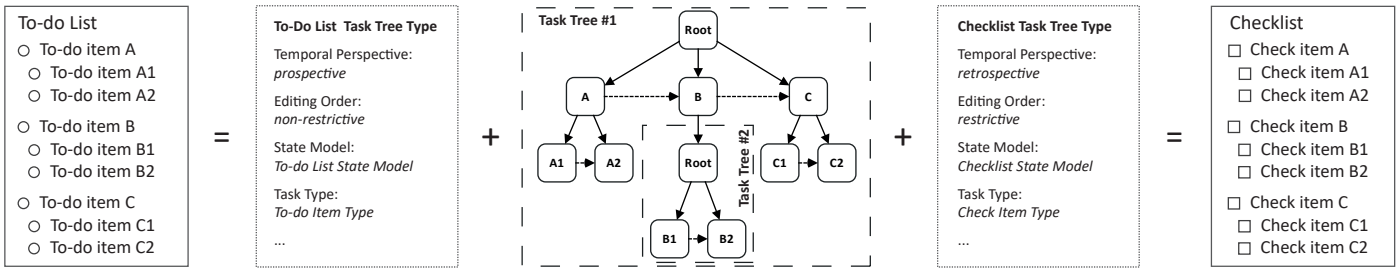

Fig. 11. To-do List and Checklists Specializations of a Generic Task Tree

to each other based on the common state types. Fig. 12 shows an example based on *Example 1*. Task instance "Validate Use Case" requires the creation of a subordinated process instance as various activities have to be performed to accomplish the task. Moreover, the current state of the task instance shall reflect the progress in the subordinated process to increase work awareness and to ease planning based on tasks. If the the process execution type of a task instance links several process instances, *synchronization rules* are utilized to keep the state model instance of the task instance in sync with the progress of the linked process instances. For example, the task instance exposes state *Open* if all linked process instances are currently in state *Initialized*. A task instance will expose state *In Progress* if one of the linked process instances is in state *Running* and none of them is in states *Canceled* and *Archived*. Notably, such synchronization rules are also used to synchronize the state of a task instance with the states of subordinated task instances. Further, synchronization rules can be utilized if the state of a process instance shall be synchronized with the ones of task instances, which are part of the task tree instances belonging to this process instance. Finally, note that such synchronization rules may be only generally applicable due to the fact that proCollab components rely on state models and, in particular, reference state models.

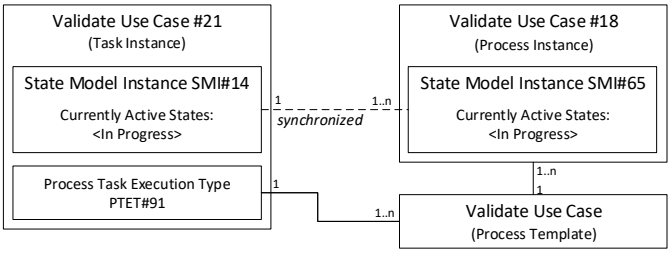

Fig. 12. Task Instance with Process Task Execution Type

VI. Evaluation

The proCollab approach with its integrated task management lifecycle shall allow for the support of a variety of KiPs. As most KiPs take place in sensitive business areas, thorough preparations are required. Especially, the elaboration of concepts and the provision of a mature implementation addressing further issues (e.g., role management) are needed to conduct valuable empirical studies validating the entire approach. To prepare such studies and to demonstrate the technical feasibility, a sophisticated proof-of-concept prototype was developed covering the concepts presented in this work as well. The prototype was realized with Java EE 7 and relies on a multi-layer architecture (cf. Fig. 13).

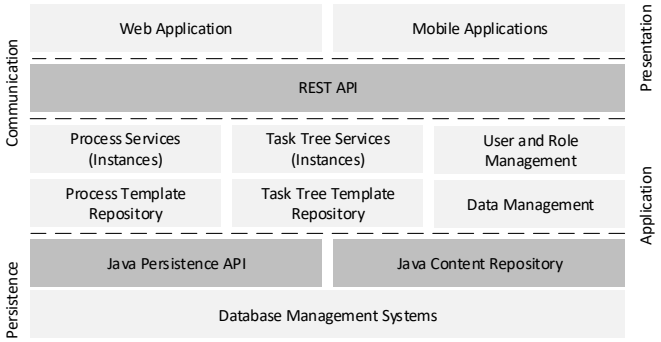

Fig. 13. Architecture of the proCollab Prototype

To validate the flexible and stateful task management support for KiPs, we integrated the agile SCRUM methodology[2] in proCollab. This integration augurs interesting results as it contrasts the traditional V model methodology discussed in Sect. II. Furthermore, Scrum is widely used in software development projects and, hence, its support can be considered as a crucial evolution step for proCollab. Fig. 14 illustrates common phases of a Scrum development project [12] including the tasks to be accomplished as sub-processes in each phase.

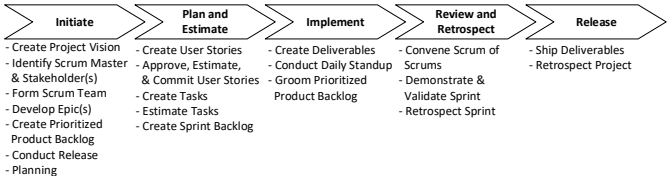

Fig. 14. Scrum Phases

To integrate these phases and processes, we first designed a dedicated *Scrum Software Development* state model for process instances based on the common phases applied in a Scrum project (cf. Fig. 15a). The phases are represented by states of a state transition graph refining the *running* state of the process instance reference state model. Demonstrating the integration of domain-specific states, a dedicated state model was created, among others, for the (knowledge-intensive) process of *Retrospect Sprint* (cf. Fig. 15b). Finally, to empower knowledge workers to directly create and use process templates and instances of the domain-specific type "software project", we created the *software development project* domain specialization type referencing the created Scrum-related state models. Based on these preparations, we created a *Scrum* process template using the *software development project* domain specialization

[2]http://www.scrum.org/

type and referencing the *Scrum Software Development* state model to be used by derived process instances. Furthermore, the *Scrum* process template contains the Scrum processes represented as task templates with the proper *process task execution type*. Regarding the latter, dedicated state models are deposited to let knowledge workers directly create and use process instances of the appropriate specialization type and with the desired state models. As discussed in Sect. V-B, the state models of the subordinated process instances are then synchronized with the state model instances of the task instances—enabling better planning, quality assurance, and awareness for knowledge workers. In addition, the integration of the key Scrum phases and processes showed the capability of proCollab to support a variety of KiPs.

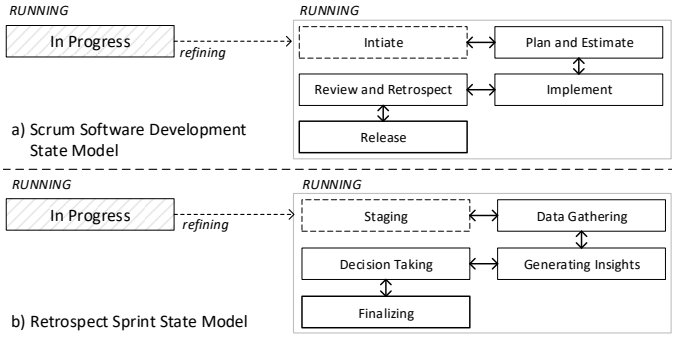

Fig. 15. Scrum-related State Models

VII. Related Work

KiP support has its roots in *Computer Supported Cooperative Work* and groupware [10]. Work more closely related to proCollab, however, can be found in the areas of Business Process Management (BPM) and *Adaptive Case Management* (ACM) [13]. Originating from BPM research, ACM targets at the systematic support of KiPs based on the concepts 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 [14]. As CMMN does not provide a dedicated representation of task trees and relies on various specialized case-related elements, proCollab does not implement CMMN. However, its components *process* and *task* can be easily mapped to CMMN elements *case* and *task*. Comparable to proCollab, *Cognoscenti* [15] allows modeling and using projects with *goal lists* and corresponding *goals*. In this context, goals are comparable to tasks. Cognoscenti lacks, for example, an integrated support of templates and the flexible state management concept of proCollab. To support software development processes, the issue management system *Atlassian Jira*[3] relies on adaptable schemes for projects and issues based on state models as well. More precisely, Jira allows for the definition of workflow schemes for projects as well as schemes for different types of issues (e.g., bugs, tasks). However, the Jira schemes do not rely on common reference models. Moreover, they are not refinable. Finally, Jira does not support state-specific functionality; instead, it focuses on the representation of these states and provides filter functions to distinguish between entities with different states.

[3]https://www.atlassian.com/software/jira

VIII. Summary and Outlook

This paper presents flexible and stateful task management for KiPs based on the proCollab approach. We focused on the proCollab state management concept, which provides state-specific functionality to knowledge workers and enables them to work with domain-specific procedure models and states. To establish such a flexible support, we considered that knowledge workers iteratively stride through the stages of planning work, performing work, studying work results, and optimizing plans in KiPs. To further enhance KiP support, we presented the proCollab specialization entities as well as the integration and synchronization of proCollab processes and tasks. These concepts may empower knowledge workers to plan and conduct their work more effectively. Finally, we presented an evaluation based on a proof-of-concept prototype and its application to a use case from software engineering. In future research, we will evaluate proCollab more thoroughly. In particular, we aim to investigate how knowledge workers define and use the domain-specific state models provided by proCollab.

References

[1] N. Mundbrod, J. Kolb, and M. Reichert, "Towards a System Support of Collaborative Knowledge Work," in *BPM 2012 Workshops*, 2013.

[2] C. Di Ciccio, A. Marrella, and A. Russo, "Knowledge-Intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches," *J on Data Semantics*, vol. 4, no. 1, pp. 29–57, 2014.

[3] P. F. Drucker, "Knowledge-Worker Productivity: The Biggest Challenge," *IEEE Eng Management Review*, vol. 34, no. 2, p. 29, 2006.

[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," *Inf Sys and e-Business Management*, vol. 13, no. 1, pp. 107–146, 2015.

[6] N. Mundbrod, F. Beuter, and M. Reichert, "Supporting Knowledge-Intensive Processes through Integrated Task Lifecycle Support," in *Proc. EDOC 2015*, 2015, pp. 19–28.

[7] 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.

[8] N. Mundbrod and M. Reichert, "Process-Aware Task Management Support for Knowledge-Intensive Business Processes: Findings, Challenges, Requirements," in *Proc. EDOCW'14*, 2014, pp. 116–125.

[9] 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.

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

[11] R. Lenz and M. Reichert, "It support for healthcare processes – premises, challenges, perspectives," *Data & Knowledge Engineering*, vol. 61, no. 1, pp. 39–58, 2007.

[12] SCRUMstudy, "Scrum phases and processes," 2017. [Online]. Available: http://www.scrumstudy.com/WhyScrum/Scrum-Phases-And-Processes

[13] M. Hauder, S. Pigat, and F. Matthes, "Research Challenges in Adaptive Case Management: A Literature Review," in *Proc. EDOCW'14*, 2014, pp. 98–107.

[14] OMG, "Case Management Modeling and Notation (CMMN) 1.1," 2016. [Online]. Available: http://www.omg.org/spec/CMMN/1.1/

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