Software and Systems Modeling (2022) 21:939–962
https://doi.org/10.1007/s10270-022-00993-3

SPECIAL SECTION PAPER

An exception handling framework for case management

Kerstin Andree[1] · Sven Ihde[1] · Mathias Weske[1] · Luise Pufahl[2]

Received: 11 November 2020 / Revised: 12 February 2022 / Accepted: 22 February 2022 / Published online: 8 April 2022
© The Author(s) 2022

Abstract
In order to achieve their business goals, organizations heavily rely on the operational excellence of their business processes. In traditional scenarios, business processes are usually well-structured, clearly specifying when and how certain tasks have to be executed. Flexible and knowledge-intensive processes are gathering momentum, where a knowledge worker drives the execution of a process case and determines the exact process path at runtime. In the case of an exception, the knowledge worker decides on an appropriate handling. While there is initial work on exception handling in well-structured business processes, exceptions in case management have not been sufficiently researched. This paper proposes an exception handling framework for stage-oriented case management languages, namely Guard Stage Milestone Model, Case Management Model and Notation, and Fragment-based Case Management. The effectiveness of the framework is evaluated with two real-world use cases showing that it covers all relevant exceptions and proposed handling strategies.

Keywords Exception handling · Knowledge-intensive processes · Flexible processes · Case management

1 Introduction

Business process management (BPM) is the core of each organization to optimize their business processes in such a way that an overall business goal is achieved. The main artifact of BPM are business process models [6]. Traditional control-flow-oriented process models represent all possible execution paths of a business process and provide a complete description of possible alternatives.

Therefore, ensuring the business processes fulfill their purpose is a necessary endeavor. However, looking at the real world it is impossible to always predict the future, thus resulting in unforeseeable events (e.g., weather changes, missing data) happening, which interrupt the execution of a business process [19]. In general, these unforeseeable events are called *exceptions*. In the worst case, these exceptions will not only interrupt the execution but also may lead to irrecoverable errors if not handled correctly. That is why exception handling is a vital method for an organization to get its business process back on track. Standard process modeling languages, such as Business Process Model and Notation (BPMN) [22], offer concepts to capture and handle exceptions in a rather detailed manner, which can lead to complex process models. An overview of state-of-the-art methods of event handling in existing business process management systems (BPMSs) is given by Russell et al. [27].

In recent years, increasing use and importance of dynamic and knowledge-intensive processes can be noticed (e.g., in the areas of medical treatment, education, etc.) [5,12]. For this processes, so-called *knowledge workers* determine the exact process path by their decisions at runtime [5,15]. Different case management languages exist to support case management, such as Case Management Model and Notation (CMMN) [23]—the standard for case management, Guard-Stage-Milestone (GSM) approach [16]—the underlying concept of CMMN, and Fragment-based Case Management

Communicated by Selmin Nurcan and Pnina Soffer.

✉ Sven Ihde
sven.ihde@hpi.de

Kerstin Andree
kerstin.andree@student.hpi.de

Mathias Weske
mathias.weske@hpi.de

Luise Pufahl
luise.pufahl@tu-berlin.de

[1] Hasso Plattner Institute, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany

[2] Software & Business Engineering, Technische Universitaet Berlin, Sekr. EN 6, Einsteinufer 17/3rd floor, 10587 Berlin, Germany

(fCM) [14]—an approach using BPMN process fragments triggered by certain conditions.

Exceptions can be handled by defining additionally fragments in fCM or stages in CMMN and GSM. This approach is limited because not all exceptions occurring at runtime can be imagined at design time, so-called unpredictable exceptions. Moreover, the resulting model would increase in complexity. Unfortunately, an adaption of the exception handling to flexible processes has not happened yet, leaving the knowledge worker to identify and handle exception on their own without any support with the risk for delays or issues in the cases. Thus, in this work, we aim to apply the exception handling patterns by Russel et al. [27], which were originally developed for well-structured processes, and develop an structured exception handling framework for case management languages. The exception handling patterns offer a broad range of possible handling strategies for different exception types structurally identified from business process execution systems [27]. This categorization can potentially support knowledge worker in a structured manner to handle unpredictable exceptions to return to the normal case execution in a fast and professional way. Thereby, we understand a normal case execution as the predictable variations and exceptions in the execution of cases in contrast to unpredictable exceptions. The aim of this work is to re-engineer the exception handling patterns to case management in order to enable case workers to handle major exceptions based on current case management literature and advances in exception handling in process and case management systems. Following the structure of [27] and extending it, we define exception handling in four levels: (1) Activity Level, (2) Fragment/Stage Level, (3) Case Level and (4) Recovery Measurements on Case Level.

In a previous work [3], we have presented the application of the exceptions handling patterns to fCM. In this work, we want to generalize the approach to other relevant stage-oriented case management languages, such as GSM and CMMN. Additionally, we have defined requirements for exception handling in case management and added a discussion on the limitations of the current exception handling patterns. Furthermore, the coverage of the framework is evaluated by an elicitation of exceptions from two knowledge-intensive processes.

In the remainder of this paper, we introduce the basics and related work on exceptions and case management in Sect. 2. Following that, we present a real-world case model example in Sect. 3 and discuss the requirements for exception handling in case management. In Sect. 4, the exception patterns are introduced as the foundation to our work. Then, we extend the current exception handling to adapt to the increased requirements of dynamic, knowledge-intensive processes in Sect. 5. Lastly, the coverage on real-world use cases is evaluated in Sect. 6 and the findings of this work are concluded in Sect. 7.

2 Basics on exceptions and case management

This section introduces the concept of exceptions and their classifications in Sect. 2.1. Further, case management is introduced as an approach to support knowledge-intensive processes in Sect. 2.2. Finally, related work is presented in Sect. 2.3.

2.1 Exceptions: terms and definitions

In general, exceptions are triggered by certain causes that lead to a deviation of the normal process execution [18]. While a cause can occur at any time (even outside of process execution), we only speak of an exception when the effects hinder an activity from being executed. Thus, we can assign a specific process state to each exception, which is defined by the states of all activities involved according to the activity life cycle. Consider the following example: *Due to an external cause, a resource is already faulty before process execution.* We do not yet speak of an exception in this case because there are no effects or relations to a process yet. Only when the process starts and the resource is allocated to an activity that cannot be executed as a direct consequence of the defect, an exception is thrown. So, we define in the scope of this research work exceptions in relation to the activity that threw it and the current process state.

Kurz et al. [18] differentiates between three types of exceptions regarding their predictability: (1) routine, (2) minor and (3) major exceptions. Exceptions, in addition to their predictability, can also be classified into types that provide more information about their cause. Table 1 shows the five different types introduced by Russell et al. [27]. Besides the very general exception type *Activity Failure*, there are more specific types of exceptions like *Deadline Expires* or wrong *Resource Allocation*. An external trigger—external

Table 1 Exception types defined for an activity based on [27]

Exception type	Explanation
Activity failure	Activity is not able to continue its execution.
Deadline expires	Deadline (i.e., time when it should be finished) of an activity is not reached
Resource allocation	1. No fitting resource was found 2. Issue with a resource occurs after allocation 3. Wrong resource was allocated
External trigger	External information leads to an interruption of the execution.
Constraint violation	Invariant of an on-going process is violated.

information— can also lead to an exception. This differentiation into different types serves to limit the variety of possible handling strategies in order to propose more specific and therefore more useful handling.

As routine and minor exception defined by Kurz et al. [18] are predictable, only the handling of routine exceptions is modeled in the process model. The handling of minor exceptions is not explicitly modeled for reasons of model complexity. However, this type of exceptions can be handled, for example, with a template-based strategy with best practices and guidelines presented by Kurz et al. [18]. Another possibility is the use of so-called exception handling patterns introduced by Russell et al. [27]. Major exceptions, though, have to be handled currently ad-hoc outside the case management system.

In this work, we want to focus on major exceptions and want to provide a structured approach to handle them in a case management system. The exceptions will be classified into the five event types introduced by Russell et al. [27].

2.2 Knowledge-intensive processes

In recent years, research has addressed increasingly the need for flexibility in business processes to enable reliability and support of static and dynamic changes [1,31]. Reichert and Weber [26] distinguish four major flexibility needs, namely (a) variability, i.e., the need to have different process model variants to handle different groups of customers, products, etc., (b) looseness, which is the need to leave some aspects unspecified during modeling because cases are unpredictable etc., (c) adaptation as the need to react to exceptions or special cases by adapting the process, and (d) evolution is the need for process model changes over time due to changing business environments.

So-called *Knowledge-intensive processes* (KiPs) have a high need of variability and looseness. They are defined as "processes whose conduct and execution are heavily dependent on *knowledge workers* performing various interconnected knowledge-intensive decision-making tasks" [5]. Process execution heavily relies on the exchange of knowledge in which content only emerges during execution.

Case management is a management approach to support the modeling, execution and analysis of KiPs [20]. Often *case models* expressed in a certain case management language, such as CMMN, are used to represent the possible executions of a *case*, i.e., an instance of a case model. It stresses the variability and looseness by under-specifying the control flow between the *case activities*, so that the knowledge worker can control the order of activities individually for a specific case [14]. In general, we differentiate in this work between three different case management paradigms depending on their main notion to realize the variability and looseness:

1. *Constraint-oriented* In these languages, the variability and looseness is mainly realized by focusing on constraints between case activities, such as the declare language [24] or DCR (Dynamic Condition Response) graphs [29]. In essence, knowledge workers are allowed to do everything as long as they comply with the existing rules and constraints.
2. *Data-oriented* These languages focus on the data objects which are processed in a case [30], the so-called *case objects*. Variability and looseness is achieved in such approaches by splitting the process definition in life cycle definitions for the individual objects and a model defining their interactions. At runtime, case objects can be created and updated flexibly whereby the life cycle definitions need to be followed. Furthermore, allowed creations and updates might be further restricted by the interaction model. An example is PHILharmonic Flows [17], where the micro-processes describe the detailed activities and relations allowed on the involved data objects and macro-processes specify how the micro-processes can interact with each other.
3. *Stage-oriented* Knowledge-intensive processes are subdivided into stages or fragments, which can then be dynamically executed based on the needs and expertize of a knowledge worker. An activity here can read and change multiple case objects. Modeling languages for stage-oriented case management are the Guard Stage Milestone Model (GSM) [16], Case Management Model and Notation(CMMN) [23] and Fragment-based Case Management (fCM) [14].

In contrast to the constraint and data-oriented case management languages, stage-oriented languages have more concepts to be handled by a knowledge worker in case of an exception, such as activities, stages and case objects. Therefore, we think that knowledge worker might have a higher need for a structured exception handling approach in order to handle the different concepts. Thus, in this research work, we focus on stage-oriented case management languages.

2.3 Related work

Exception handling in structured processes to handle unforeseen errors immediately has been widely researched and analyzed, resulting in the well-known exception handling patterns [27].

In addition to support adaptability [26] which is essential for exception handling, a structured elicitation of change patterns for business process execution systems by Weber et al. [32] provides a broad range on how a specific process case can be adapted by domain experts, e.g., by adding or deleting process fragments. However, case studies have shown that the use of change patterns for more complex

adaptions requires a high skill cap of the process designer [33]. Similarly, Zimmermann and Döhrig [34] provide adaptation patterns for BPMN, where the exception handling patterns [27] are considered as well. How they realize the exception handling patterns as BPMN adaptation patterns is not discussed in detail. Recovery measures in case of exception has been researched for BPEL (Business Process Execution Language), an XML-based language to define the execution of structured processes. Fahland and Reisig [7] provide for BPEL an execution semantics to handle exceptions. Therefore, process scopes are defined in which certain errors are relevant and can be handled by a so-called fault handler. The fault handler is waiting for exceptions, and if one occurs, it stops the execution in the scope and executes an exception handling activity. This work has also influenced the execution handling in BPMN, where error boundary events at an activity or subprocess can catch errors, interrupt the execution and can trigger a flow to handle the error [22]. In contrast to this work, it provides a concept for predictable exceptions that are pre-specified in a fault handler.

For exception handling during runtime in case management, adaptability is a key factor but still an open topic [12]. Hybrid process modeling notations that combine imperative with declarative process modeling can also provide support for knowledge-intensive processes. A structured overview of those is given by Andaloussi et al. [2]. Although many languages aim at adaptability during the process execution, a systematic exception handling approach is not supported.

Next, we want to present existing exception handling techniques for the different case management paradigms:

Constraint-oriented. Constraint-oriented languages, such as DECLARE [24], provide constraints as a way to restrict the case execution. This provides the knowledge worker a lot of freedom to handle predictable and unpredictable exception during the case execution. Thus, no specific approach exists so far to support exception handling in constraint-based approaches.

Object-oriented. For Philharmonic flows, Andrews et al. [4] provide a technical concept for the case engine to support ad-hoc changes on the life cycles of case objects. It allows the user to adapt a case to unknown exceptions but the user has to initiate it on its own and no holistic support for handling unknown exceptions is given. A more automatic-driven exception handling technique for object-oriented case management is provided by Sid et al. [28]. The authors use AI planning to support the execution of knowledge-intensive processes by providing the case worker a guidance on how to execute a case. In case of an exception that is defined as an external event, which requires changes on data attributes, a re-planning is initiated. Thus, exceptions in this work [28] can be directly mapped to changes on specific case objects. We consider unknown exceptions as a more general concept where a concrete mapping to concrete options of exception handling is not directly given.

Stage-oriented. Steinau et al. [30] show in their framework for data-centric approaches that GSM does not support for ad-hoc changes and error handling. In contrast, in fCM [14], new fragments can be added during runtime which support a kind of exception handling. Knowledge workers have to rely on their experience for handling an exception and have to make sure that the case model is still valid when they add a fragment. Fahland and Woith [8] propose a flexible process execution system based on small Petri nets fragments, called Oclets that can be dynamically combined at runtime, and additional Oclets can be defined to handle predictable exceptions. Furthermore, unpredictable exceptions can be handled by adapting the case model while running, which are also verified [9]. Still, a systematization about the possibilities for handling an exception is not yet given.

3 Motivation and requirements

This section motivates the need for a structured framework for case management languages that handles exceptions caught during runtime. We first introduce a knowledge-intensive process in Sect. 3.1—the organization of a course—for explaining the three different modeling techniques for stage-oriented case management: GSM, CMMN, and fCM, and its already existing exception handling methods including their limitations in Sect. 3.2. The use case as well as the models were developed in collaboration with two knowledge workers who are working in this field. Based on this, requirements for an effective and structured exception handling framework are deduced in Sect. 3.3

3.1 Motivating scenario

To organize a university course, the knowledge worker—the professor and/or the teaching assistant (TA)—have to decide on each step the most appropriate execution path to deliver a successful course. A course exists in different formats, e.g., with graded exercises and without, and might have to be adapted to different parameters (e.g., holidays during the semester, previous knowledge of the students, etc.). In the given process, the knowledge worker has to work on three main fields, namely the course, the exercise, and the exam, which are represented as stages in CMMN. In order to have a successful and insightful course in the upcoming semester, the knowledge worker can plan the course. This includes figuring out the amount of occurrences of the course depending on the holidays during the semester, as well as the maximum allowed amount of students in the course. Only based on this planning, the knowledge worker can prepare the course

Table 2 Overview of exceptions that might occur when organizing a university course and are not modeled

Exception	Exception type	General handling strategy
Corona regulations prohibit presence events	Activity failure	complete change of course format (presence to online), use of prerecorded material
Overbooking of the lecture hall	Resource allocation	Request new room and reschedule/complete loss of course
Student gets sick, no exam can be handed out	Resource allocation	Get illness validation document and schedule a catch-up date
Students have questions after or outside of the course	External trigger	Unstructured solution: answer questions directly/structured solution: offer a scheduled time for Q&A

content (slides, learning material, etc.), or book the needed rooms to hold the lectures/exercises/exam. For each of the planned dates, the knowledge worker then holds the lecture. Optionally, in addition to the lectures, sessions can be held that may include hand-on-sessions to deepen the engagement of the students with a certain topic.

Complementary to the lectures, the knowledge worker can decide on handing out exercises and even grade them. However, as the workload for one person is high, tutors are often hired to help with exercises and grading, if necessary.

Finally, an examination has to be held to grade students. The exam has to be prepared, e.g., by booking a room for a written exam. After the exam was prepared, it will be eventually conducted and then graded.

Identified in collaboration with TAs, Table 2 shows a selection of possible major exceptions, which can occur in this scenario and would not be captured in the case model. The assignment of the exceptions to the different exception types was approved by the TAs. Similarly, the handling strategies described are the methods that are also used in reality.

3.2 Stage-oriented case management languages

GSM. The Guard-Stage-Milestone (GSM) language [16] was created for *Business Artifacts* [21] with focus on the high-level data—data artifacts—that are handled during a case. A GSM model consists of four major components: the information model, the stages, the milestones, and the guards. The information model defines the data artifact and the allowed operations on them which is expressed in a lifecycle model, which in turn can use stages, milestones, and guards for the definition. Stages are mainly used to structure the process. They split the process into logical parts that have a meaningful impact on the business artifact. An atomic stage is defined as a stage that does not contain any other stages. From a semantic point of view, we can treat atomic stages the same as an activity in other modeling languages, e.g., fCM or CMMN. Milestones are a concept used to define the outcome of a successful stage. Therefore, they describe the goal that should be fulfilled by the stage. This also means a stage can have their milestone fulfilled even if not all contained stages reached their milestones. Lastly, guards are used to enable stages. As soon as their guard condition has fulfilled, a stage can be executed. Usually, stages have no inherent order for their execution, except for the hierarchical order. By using guards and milestones, the modeler can however force a more controlled execution order. An example of an application of the GSM models is shown in Fig. 1. As there should be an order between the stages/activities "Prepare exam" and "Conduct exam," the guard condition of "Conduct exam" is equal to the milestone of "Plan exam."

Routine exception can be captured as additional stages in a GSM model. Knowledge workers can use stages to define optional handling strategies in case of an exception specified as entry condition. For minor and major unpredictable exceptions, no support exist.

CMMN. Case Management Model and Notation (CMMN) [23] is historically based on the concepts provided by the GSM language. The case model shown in Fig. 2 does not include a predefined order of the activities. It is completely up to the knowledge workers to define the process path during runtime. Therefore, stages, stage entry conditions (guards), and stage exit conditions (milestones) are concepts also contained in CMMN. In the scenario of organizing a university course explained above, three stages can be identified, one for each building block: the course, the exam, and the exercise. They are similar to the ones in the GSM model. In contrast to GSM, entry and exit conditions (called Sentries) are linked to activities or milestones. This means, for example, that the activity *Prepare course content* can only be executed when the activity *Plan course* has been completed. Milestones are "achievable target[s], defined to enable evaluation of progress of the Case" [23]. For example, an important milestone in the university course organization process is reached, when all exams are graded. CMMN offers the concept of *discretionary* and *non-discretionary* activities.

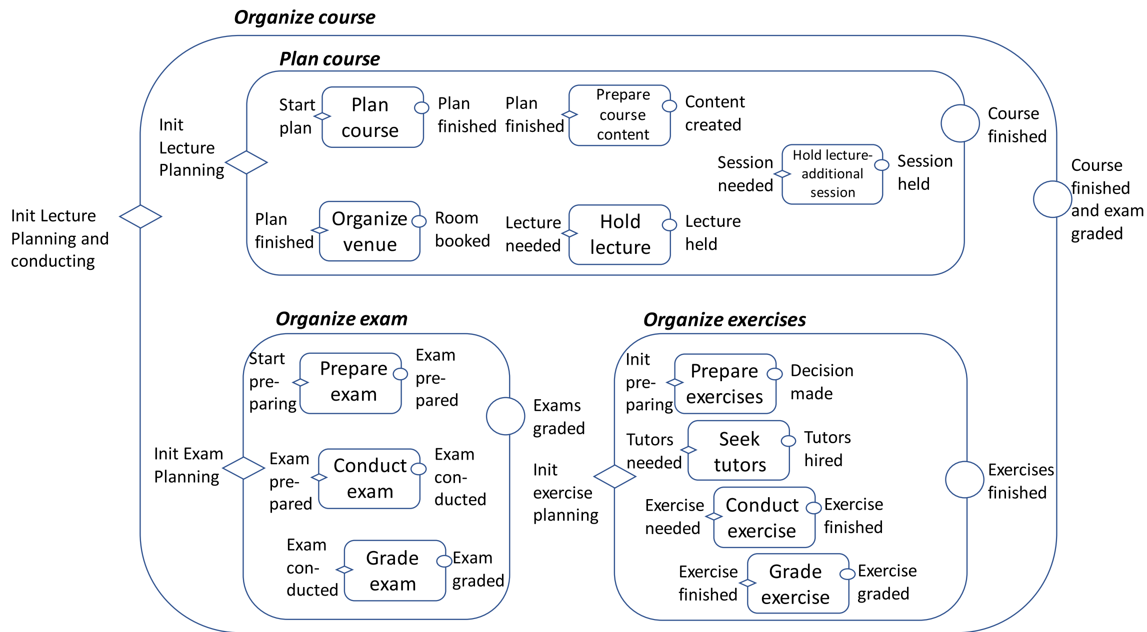


Fig. 1 Process of organizing a university course captured as GSM model

If an activity is discretionary, knowledge workers are allowed to include or skip the activity in the execution path since it is not an essential part of most of the cases but could be important for special ones. They are modeled with a dotted line, e.g., *Seek tutors*. Non-discretionary activities need to be always executed. Moreover, CMMN enables knowledge workers to directly terminate activities or entire stages [23], i.e., to immediately stop the execution of activities/stages. If a stage instance is terminated, all included activities, milestones, and stages have to terminate too.

CMMN does not provide explicit modeling elements, such as known from BPMN, for exception handling for routine exceptions. However, discretionary tasks can be used to define optional handling strategies in case of an exception. Knowledge workers can add them during runtime to the case model. Similarly, milestones can be used as a form of exception handling. For example, the milestone that students have questions after a lecture was conducted. This means the case worker would need to hold lecture-additional session to

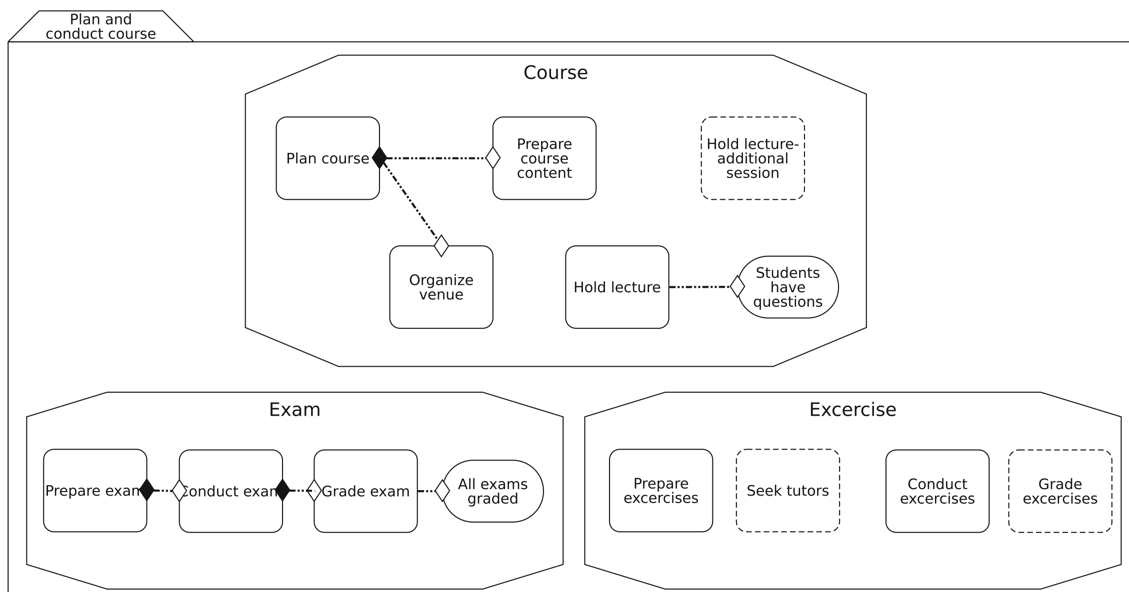
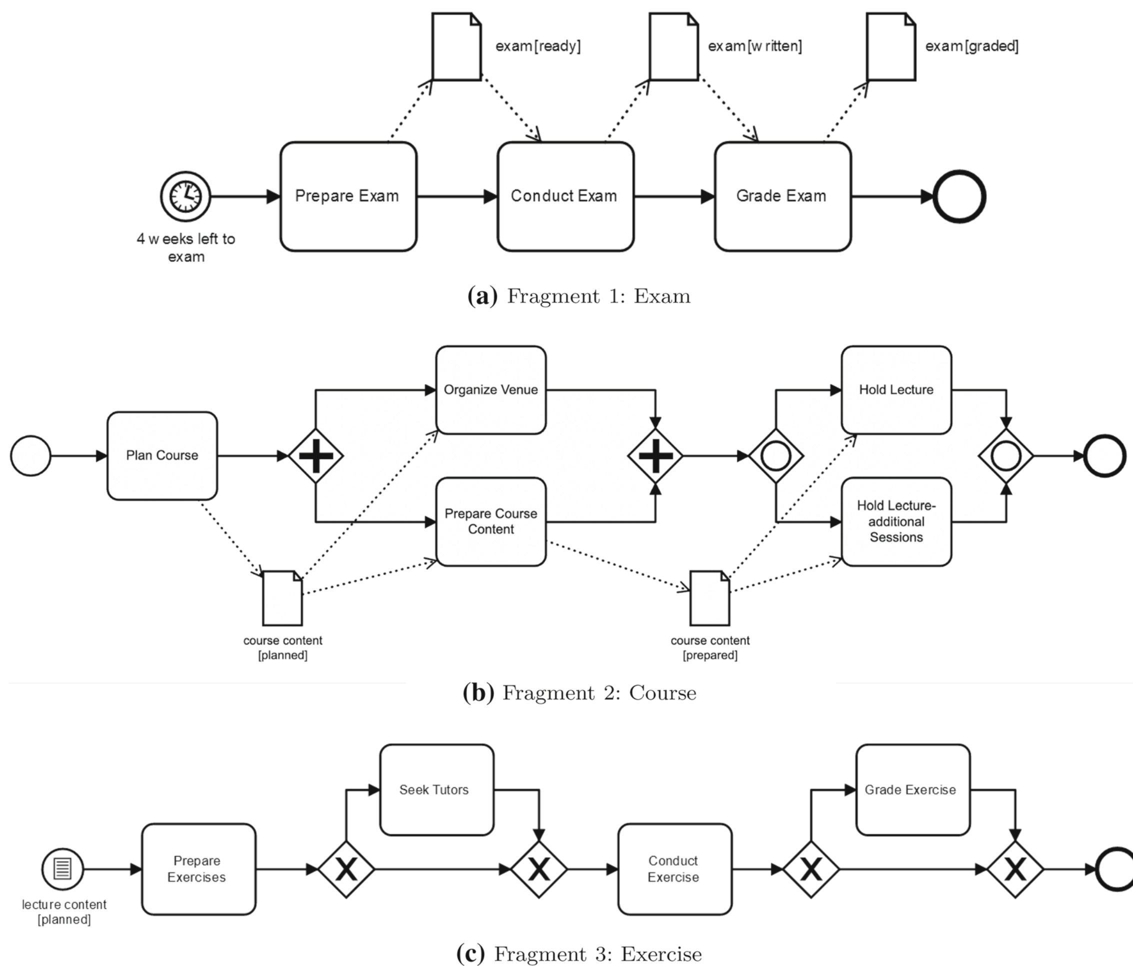


Fig. 2 CMMN diagram of a university course organization process



(a) Fragment 1: Exam

(b) Fragment 2: Course

(c) Fragment 3: Exercise

Fig. 3 Process of organizing a university course captured as fCM model with *exam[graded]* as termination condition

address and answer them. Nevertheless, these strategies do not cover minor and major exceptions.

fCM. Fragment-based Case Management (fCM) offers a hybrid variant for case modeling with having static parts that are executed in a specific order and can be flexible combined at runtime. It is an actively researched case management language [10,13,15] with a prototypical case engine [11] and an application to real-world [25]. A *case* in fCM consists of several *fragments*, each of them having an entry condition: either they are enabled at any time (e.g., Fragment 2 in Fig. 3b), or enabled if certain data conditions are fulfilled (e.g., Fragment 3 in Fig. 3c is only enabled if the course content is [planned]), or when a certain external event has occurred (e.g., Fragment 1 is enabled four weeks before exam period, cf. Fig. 3a). Each fragment is a smaller imperative process model; for this, fCM reuses a subset of the BPMN notation [22]. Activities and fragments are connected by data objects and their states as entry and exit conditions. If an activity is executed, a change of the data object takes place, i.e., a transaction in the database and thus a state change on the knowledge object. Allowed state transitions are defined in the so-called *Object Life Cycle* (OLC) provided as a state-transition diagram that defines how a case object evolves during runtime. FCM offers knowledge workers the possibility to decide which fragments are executed at runtime. Enabled fragments with no or a fulfilled data condition are enabled but do not need to be executed, only started fragments by an external event. For example, fragment 3 can be ignored if no exercises are planned. A case is terminated by the knowledge worker if the goal condition is fulfilled. The case goal in fCM is specified as a number of data states which need to be reached. However, this is not tied to the completed execution of the fragment which is responsible for the transition to the target state.

To handle routine exceptions, it is possible to add boundary events to certain activities as known from BPMN. It is also possible to model fragments with a start event that is triggered by a predictable event. For exception handling at runtime, new fragments can be added which support minor and major exceptions [14]. It is also possible to re-execute fragments. However, these strategies are limited by the fact that it is not allowed to modify OLCs. Knowledge workers

have to rely on their experience for handling an exception and have to make sure that the case model including OLCs are still valid when they add a fragment. Structured support for exception handling is not yet supported.

3.3 Requirements

Due to the support of variability and looseness of the presented case management languages, a handling of predictable exceptions can be already specified at design time. However, due to the associated increased complexity of the model, this strategy is often omitted. In this case, knowledge workers are responsible for handling the exceptions on their own. Also for handling unpredictable exceptions, which cannot be modeled either, knowledge workers have to react ad-hoc. While there are ways to handle exceptions, such as adding fragments during runtime in fCM, these methods are unstructured and place high demands on knowledge workers in terms of experience and expertize. There is no procedure for effective and structured exception handling for unpredictable (major) exceptions which is why we aim at a framework that provides structured exception handling for these exceptions. This framework defines handling strategies which enable further correct process execution. In this work, we focus on high coverage of exceptions instead of usefulness for knowledge workers.

The requirements for such an exception handling framework derive from the general characteristics of KiPs [5,12,20] as well as from the above discussion of the stage-oriented case management languages. Each requirement listed below applies to both KiPs and case management in general. If a requirement needs to be defined more precisely due to the characteristics of a specific case management language, this is indicated by a subcategory and the abbreviation of the language[1].

R1 Case Data Case knowledge has to be kept up-to-date [5]. If data are incorrect due to an exception, this must be detected and corrected [28]. Especially in fCM, object life cycles have to be followed. In case of changing a data object, the execution of fragments must still be possible.

R2 Case Goal Reaching the case goal has highest priority [5]. Exception handling strategies should not terminate the process if it is still possible to ensure achievement of the business goal. For example, in case of the university process, if in-present lectures are not possible due to the pandemic, it should be still possible to finish the course and grade the exams.

R3 Exception Handling on Global Case Level Exceptions can affect the entire case. In addition to exception handling for the activity that triggered the exception, the suggested strategy must also consider case-related effects and provide methods for them.

R3.1 (fCM) Fragments Exception handling has to provide a handling for the fragment in which the exception was triggered as well as for following fragments. For example, in case of the university process, if in-present lectures are not possible due to pandemic, a complete change of the course style might be required which influences the *Course* fragment.

R3.2 (GSM & CMMN) Stages Exception handling has to provide a handling for the stage in which the exception was triggered. Similar to the example given in R3.1, the *Course* stage needs to be handled.

R4 Recovery Measurements Exception handling must be able to offer various recovery measures [30], such as rollback in execution or compensation of the exception effects, without endangering further process execution.

R5 Flexibility Exception handling strategies should not limit the flexibility of knowledge workers in their case execution [5,20]. This means that the knowledge worker should have several option to handle the case and continue the case execution.

R6 Structured Support Exception handling strategies should be defined in a uniform structure that is generally applicable [12]. For the same exception in a specific state, the proposed handling methods should be reproducible.

For the realization of the exception handling framework, we decided to use a pattern-based approach as introduced by Russell et al. [27] which is presented in the next section with its capabilities and limitations with regard to case management.

4 Foundation on exception handling patterns and their limitations

This section explains the general structure of the patterns for workflow systems introduced by Russell et al. [27]. The proposed exception handling strategy consists of a tuple structure with three different components that form the pattern:

1. Handling of the activity which provoked the exception (*activity level*);
2. Handling of the following activities of the whole case (*process case level*); and

[1] E.g., *R3.1 (fCM) means that requirement 3* has to be defined in more detail for fragment-based case management.

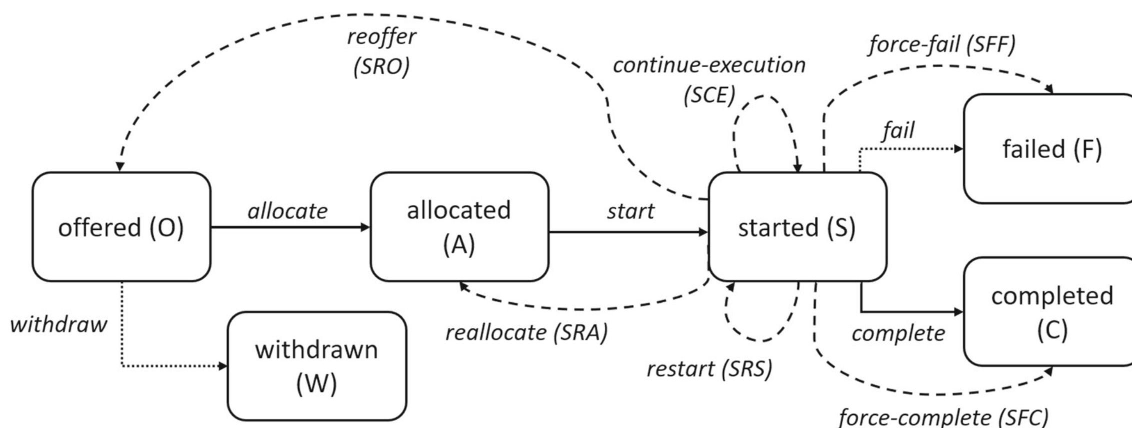


Fig. 4 Life cycle of an activity and forced transition of exception handling from [27]

3. *Recovery measures* which are needed to remove the effects.

In the following, each component of the framework is explained in more detail. Finally its limitations with regard to case management are discussed.

4.1 Activity level

The handling of the activity, which caught the exception, is summarized by the first part of the pattern. It describes a state transition, which depends on the state of the activity when the exception is caught.

In general, business process activities can adopt different states, from the initial offering until final termination as shown in Fig. 4. Each activity can change its state via state transitions represented as solid arrows for those being executed in the workflow system, and as dashed arrows, which are deliberately enforced from the outside as exception handling. For example, a state of an activity can change from *started* to *failed* in two different ways, the natural one (fail) or the enforced one (force-fail). Next, we want to briefly introduce the structure for the exception handling patterns on the activity level with the focus on exceptions that occur while the related activities are in state *started*[2].

If an exception occurs in an activity state, in this case *started*, the activity can be moved to any state according to the life cycle. Each possibility is identified in the pattern with a three-letter abbreviation, e.g., *SFF*. These identifiers can be found in Fig. 4 within the brackets. The three letter abbreviation is read in two parts. While the first letter represents the state of the activity in which the exception occurred, the two other letters define the state to which the activity is transferred as part of the exception handling. Consider the example *SFF*. An exception occurred in the activity state *S* = *started* and is handled by a state transition of the activity to *failed* via a *FF* = *force-fail* (cf. Fig. 4).

[2] An overview of all possible state transitions can be found in [27].

4.2 Case level

The second part of the pattern for handling exceptions in a business process engine addresses the best strategy for handling the case and all subsequent activities after the activity that triggered the exception. Since a process instance consists of multiple activities connected by input and output conditions, the effects of exceptions have an impact on these constraints concerning the execution of following activities. There are three possibilities for handling an exception on the case level:

1. *Continue Current Case (CWC)*—Every activity following will be executed, the process instance still exists. The activity handling has to be done in such a way, that further activities can be triggered, e.g., because a resource is now available.
2. *Remove Current Case (RCC)*—Either a selection or all of the activities are removed from the runtime database, i.e., selected activities cannot be executed anymore.
3. *Remove All Cases (RAC)*—All process instances of the same process model are removed from the runtime database, i.e., all cases interrupt in their execution and cannot be executed anymore. This strategy handles the worst case of exceptions affecting all process instances of a model. It can cause a change in modeling. Such handling becomes necessary, for example, when laws change or in case of a pandemic.

4.3 Recovery measurements

The last part of each pattern deals with recovery. Recovery describes the action performed in order to remove any

aftereffects of an exception to ensure the possibility of still achieving the business goal. Three methods can be used [27]:

1. *Do nothing (NIL)*
2. *Rollback (RBK)*—The effects of the exception are reversed, i.e., the state of the process instance is reset to a previous process state shortly before the time at which the exception occurred.
3. *Compensate (COM)*—The damage caused by the exception that has occurred will be compensated.

For example, the pattern *SFF-CWC-COM* can therefore be interpreted as a strategy that transfers the activity, of which the exception occurred, into the status *failed* (SFF). The process instance as well as the following activities are continued in execution without making any special changes (CWC). Any damage that the exception caused is compensated (COM).

4.4 Limitations

If an exception occurs in a knowledge-intensive process, (1) the activity during which the exception occurred, and (2) the case are affected similar to traditional processes (Requirement R3). For exception handling, (3) recovery measurements are required in addition to handle possible effects and consequences of the exception (Requirement R4). These three levels covered by the original exception handling patterns [27] are also relevant for case management. However, case management languages differ from traditional business process languages, such as BPMN in being strongly data-driven and more flexible in their execution. Thus, not all requirements introduced in Sect. 3 are already covered by the exception handling patterns [27]. For each requirement, Table 3 provides the current support and a brief reason why it is already covered, why it is only moderately covered, or not covered at all.

Even though R3 is included in its general definition, it is not possible to directly apply the handling strategy Continue Current Case (CWC) to fCM. It is not defined whether the activities within the fragment containing the activity that triggered the exception should continue in execution or whether the activities in other, subsequent fragments have to continue. For example, if the knowledge worker decides to not include exercises at all because no tutors (exception is triggered by activity *Seek Tutors*) can be found, all activities of Fragment 3 (cf. Fig. 3) do not continue. Nevertheless, the course takes place in the next semester, i.e., Fragment 1 and 2 continue. The same scenario also occurs in GSM and CMMN. Here, the strategy does not distinguish between activities within

Table 3 Overview of requirements already covered by the exception handling patterns of Russell et al. [27]

Requirement	Covered	Explanation
R1 Data	–	No definition of how to deal with state transitions of data objects. Traditional business process systems are not as data-driven as KiPs. The exception handling patterns do not deal with state transitions of data objects.
R2 Goal	+	Except for RAC, it is still possible to achieve the goal since the exception handling patterns do not include termination as a handling strategy.
R3 Exception Handling on Global Case Level	+/–	The exception handling patterns provide a handling for the activity that triggered the exception as well as for the case. However, they have to be re-defined in the context of the specific case management language.
R3.1 (fCM) Fragments	–	Not included since the exception handling patterns were designed for traditional business process systems.
R3.2 (CMMN) Stages	–	Not included since the exception handling patterns were designed for traditional business process systems.
R4 Recovery Measurements	+ /–	Exception handling patterns provide three recovery measurements. However, they have to be re-defined in the context of the specific case management languages.
R5 Flexibility	+	It is often the case that there are more possible strategies in handling an exception. Knowledge worker are still flexible in their decisions.
R6 Structured Support	+	The pattern-based approach provides a general structure by defining different tuple elements.

the stage where the exception was triggered and the activities of other stages (including the outermost stage) that are not initially affected by the exception. Therefore, we need to extend them to provide methods handling affected fragments and stages (R3.1 and R3.2).

Recovery measurements (R4) have to be revisited for case management. The high involvement of data objects causes additional challenges in performing a rollback and compensation. Rolling back in the process execution must be done in accordance with state transitions of the related data objects. It is, for example, more difficult to perform a restart on an activity because the entry conditions might be not met since the data object has already been updated (R1).

In summary, the discussed limitations of the exception handling patterns of Russell et al. [27] show that the framework is applicable to case management approaches, but needs to be extended in certain aspects.

5 Exception handling framework for case management

This section presents the exception handling framework for case management languages in detail. First the extension to Russel et al. [27] is introduced in Sect. 5.1. A classification of the patterns by exception types is provided in Sect. 5.2. In Sect. 5.3, we provide a functional design of an exception service for a case management system.

5.1 Extension of the exception handling patterns

This section explains the general structure of the exception handling patterns for case management shown in Fig. 5 and explains the required extensions to the framework of Russell et al. [27].

For the proposed framework for exception handling in case management, we adopt the general structure of the exception handling patterns in terms of the element-wise representation of a pattern defining exception handling on different levels [27]. However, the case management approaches fCM, GSM, and CMMN include special concepts: Fragments/stages are essential flexibility concepts which group a set of activities that are available if certain conditions hold true but do not have to be executed necessarily to reach the case goal. Consider this in the context of exception handling: An exception is always caught by an activity. Depending on the case management language, the activity is either part of a fragment (fCM) or stage (GSM/CMMN). Therefore, exceptions can lead to the inability of a fragment/stage to start or they can impede the successful execution of a fragment/stage. Whereas the handling of the activity is covered by the first tuple element and following activities outside the fragment or stage are covered by the third tuple element, a new tuple element is required that defines the exception handling for succeeding activities *within* the fragment/stage, in which the exception has been observed. Therefore, an additional component, the fragment/stage level, needs to be added to the three tuple elements of the original patterns. Overall, we make following changes for the exception handling framework for case management:

1. *Redefinition of handling at activity level:* Adapt the methods according to the different life cycle of activities in the context of case management.
2. *New definition of handling at fragment/stage level:* Include the handling of activities and any recovery measurements needed to resolve an exception within a fragment/stage.
3. *Redefinition of handling at case level:* Cover the handling for fragments/stages of the case that do not contain the activity which caught the exception. We keep the three possible handling strategies CWC, RCC, and RAC.
4. *Redefinition of recovery measurements:* Any measurement defined by this component only refers to the case

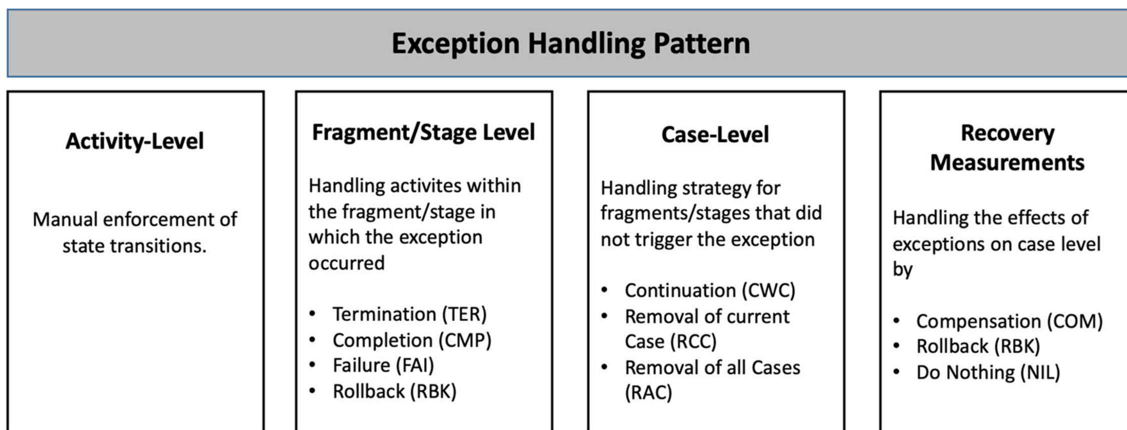


Fig. 5 Structure of exception handling patterns for case management

Table 4 Demands for extensions of case management languages

GSM & CMMN	fCM
Knowledge workers are allowed to change the discretionary attribute of activities	Fragments can be removed from the runtime database, i.e., set to a read-only state
Knowledge workers are allowed to add completely new tasks to the plan model	Knowledge worker can terminate fragments
Knowledge workers can force an activity to each state defined in Sect. 5.1.1	Knowledge worker manually change OLCs during runtime
	Knowledge worker can force an activity to each state as defined in Sect. 5.1.1

level, i.e., to fragments/stages that do not contain the activity which triggered the exception. Methods defined by Russell et al. [27] are kept.

Thus, the exception handling for case management is defined by four levels: (1) Activity Level, (2) Fragment/Stage Level, (3) Case Level, and (4) Recovery Measurements on Case Level. In order to ensure applicability of the methods of the framework presented here in the respective approaches, we work with several assumptions regarding rights and capabilities of knowledge workers, which are not yet supported by the respective approaches. We, therefore, have to define several demands on the case management approaches summarized in Table 4.

In the following subsections, the notation and logic of each level are explained. Moreover, we show how the original patterns can be applied and implemented into the case management languages CMMN, GSM, and fCM.

5.1.1 Activity level

The activity level defines the handling of the activity that has caught the exception. It is based on the associated life cycle of activities. For case management, we introduce a new, adapted life cycle for case activities as shown in Fig. 6. It summarizes the possible states and state transitions of activities in fCM, GSM, and CMMN in a simplified and adapted form and allows a more general exception handling approach for case management applications[3]. Although the life cycle shows strong similarities to the life cycle defined in Fig. 4, there are significant differences to exception handling for case management.

Figure 6 differentiates in three types of arrows: (1) Solid lines representing automatic state transitions, which are provided by the system, (2) slightly dotted lines representing state transitions because of known exceptions and decisions taken by knowledge workers, and (3) dashed lines representing state transitions an activity can be forced to due to exception handling on activity level. The state *enabled* (including the disabling function) allows for retrieving an enabled activity without doing a re-allocation. Furthermore, case management applications distinguish between the final states *completed* and *terminated* [23]. Activities in the terminated state can either have been executed successfully, and thus, fulfill the exit criterion, but they can also have been stopped in their execution by a knowledge worker. In contrast, activities only can be in the completed state if and only if the work of the activity has been conducted. This way it can be distinguished whether the work of the activity has been done (completed) or whether the exit criteria has already been reached (terminated).

An activity is set to *failed* if a routine exception occurs, which is caught by a boundary event in fCM, for example, and triggers an alternative path. In contrast, the state *suspend* can be reached when a knowledge worker suspends an activity or when an unknown exception occurs. Reasons for intentionally suspending activities can be, for example, a known exception that includes as treatment a best practice that knowledge workers perform from their experiences and is not yet defined as a routine exception, since the latter would cause a state transition to *failed*. In case of an unknown exception, however, the related activity is immediately set to state *suspend*, i.e., it is automatically halted in its execution (instead by a knowledge worker). The handling at activity-level, therefore, *always* starts with an activity being suspended in contrast to Russell et al.'s definition [27].

Defining exception handling independently of the activity state in which the exception was caught reduces the complexity of the framework with respect to the possible activity-level strategies. Exception handling according to Russell et al. [27] either goes back in the life cycle or leads to immediate termination of the activity. Transitions to running states that were not reached before the exception occurred can only be realized through automatic state transitions (solid lines). This results in 15 different treatment strategies for an activity. We reduce these strategies to almost half by allowing knowledge

[3] The naming of the states in white boxes are semantically identical with the states of Fig. 4. Since the naming of the state shown here is more common in case management, we decided to use them for the case activity life cycle.

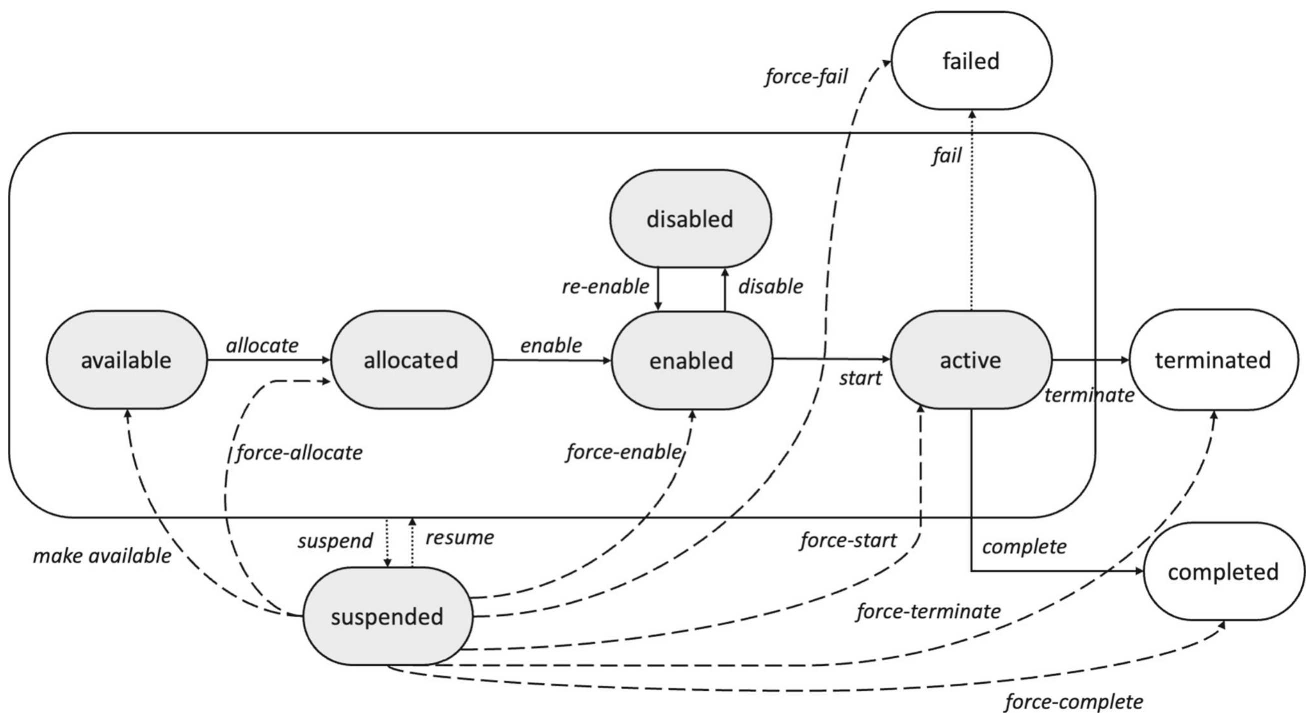


Fig. 6 Life cycle for activities in case models

Table 5 Exception handling methods at the activity level, if the activity is in the state *suspended*

Exception Handling at Activity Level
suspended make available (SMV)
suspended force-start (SFS)
suspended force-enable (SFE)
suspended force-allocate (SFA)
suspended force-fail (SFF)
suspended force-complete (SFC)
suspended force-terminate (SFT)

workers to force any state shown in the life cycle of Fig. 6. This has the advantage of allowing states to be skipped, which can contribute to more efficient process execution, and gives knowledge workers more flexibility in handling exceptions. For example, an activity that has caught an exception in state *allocated* can be transitioned directly to state *active*, rather than re-allocating the activity and waiting for the automated state transitions *enable* and *start*.

Dashed lines shown in Fig. 6 represent the states an activity can be forced to. In the framework, these transitions are abbreviated using the three letter abbreviation as shown in Table 5. While the first letter describes the initial state, the last two letters define the state into which the activity is forced.

5.1.2 Fragment/Stage level

The second tuple element describes the exception handling on fragment/stage level dealing with all following activities within a fragment/stage after the exception was triggered. Applicable methods are presented in this section.

In general, there are four different options to handle an exception on fragment/stage level:

1. *Termination (TER)*—A fragment/stage is directly terminated by knowledge workers. All components within the fragment/stage, e.g., tasks, milestones, are set automatically to state *terminated*.
2. *Continuation (CON)*—All following activities of the fragment/stage can be further executed.
3. *Failure (FAI)*—All following activities within the fragment/stage are set to state *failed*, i.e., the entire fragment/stage is set to a read-only state.
4. *Rollback (RBK)*—This strategy refers to the recovery measurement RBK on case level, i.e., the resetting of the process state through state changes of knowledge objects and at least two activities. The only difference is, that a rollback on fragment/stage level is performed within the fragment/stage.

These strategies are derived from the handling at activity level explained in Sect. 5.1.1 and recovery measurements introduced by Russell et al. [27]. The specific method, how-

Table 6 Relation between exception handling at activity level and fragment/stage level

	Fragment/Stage Level			
Activity Level	TER	CON	FAI	RBK
SMV		x		
SFS		x		
SFE		x		x
SFA		x		
SFF		(x)	x	x
SFC		x		
SFT	x	x		x

ever, directly depends on the handling on activity level. Table 6 provides an overview of what methods are applicable for a given activity handling.

Termination (TER) allows re-execution of fragments/stages. It is only triggered by forcing an activity to state *terminated* (SFT) because termination of a fragment/stage automatically terminates all contained activities. However, such a handling is only defined for CMMN [23] which is the reason why we need to extend this possibility to GSM and fCM.

In contrast, failure (FAI) forbids re-execution of fragments/stages, i.e., reuse is not possible. Therefore, FAI cannot be combined with a handling that includes further execution of the fragment/stage. It is only triggered by forcing an activity to state *failed* (SFF).

Making an activity available again (SMV), forcing it to start (SFS), force-allocate it (SFA), and forcing it to completion (SFC) directly aim to complete the fragment/stage. Therefore, all of these strategies can only be combined with a continuation (CON) on fragment/stage level because they enable the execution of following activities and thus enforce the continuation of the fragment/stage instead of aiming at a rollback (RBK). Whereas the first four strategies aim at a re-execution of the activity that triggered the exception, SFC ensures on the process level that the compensation on the part of the knowledge worker is handled correctly even if the activity itself was not executed as intended. If an activity is set to failed (SFF), i.e., no entry criteria is met that may be needed for other activities in the fragment/stage, CON can only be applied if and only if the activity is marked as optional. We define an activity as optional if it is a discretionary task (CMMN) or if it is part of an XOR gateway path (fCM) meaning that there exists an alternative path that can be executed, e.g., by performing a rollback (RBK).

Forcing an activity to state *failed* (SFF) in combination with a rollback (RBK) on fragment/stage level means that the framework recommends to roll back the process a few steps within the fragment, but not to execute the activity marked as failed when the process is executed again.

A force-enable (SFE) allows both continuation (CON) and rollback (RBK) since it does not define if the activity that caught the exception has to be directly executed or at a later point of time, i.e., after performing a rollback. Similarly, a force-terminate (SFF) on activity level enables a continuation (CON) as well as a rollback (RBK) since other activities of the fragment/stage can still be executed.

5.1.3 Case level

Case level handling is important to mitigate the possible effects of an exception on the entire case. Strategies acting on case level affect all subsequent fragments or stages, i.e., fragment/stages in which the exception was not caught. Three strategies are available for this purpose: Continue Current Case (CWC), Remove Current Case (RCC), i.e., the current case instance, and Remove all Cases (RAC) [27].

In this paper, we work with the life cycle for cases shown in Fig. 7. It originates from CMMN [23] but is also valid for fCM and GSM. Besides an active state, it distinguishes between four end states from which the case can be reactivated: *completed*, *terminated*, *failed* and *suspended*. Only the state *closed* is final. The state transitions for cases in fCM and GSM are also included in this life cycle. The fragment-based language allows the termination of case instances after the fulfillment of the exit condition (terminated), and thus a removal from the runtime database (closed). GSM also defines an active and a fulfilled state, which leads to the termination of a case, similar to fCM. The states added by CMMN do not interfere with the handling for the other languages, because each method can be implemented without them [3].

When an exception is triggered, the case is always set automatically to state *failed* until knowledge workers start the exception handling. The proposed framework aims at providing an exception handling that re-activates a case to continue its execution in order to close it via the state *completed* meaning that all milestones, stages/fragments, and task instances are completed or terminated without any active (executing) activities. Using the exception handling patterns, knowledge workers get an idea of whether they can use the existing fragments/stages for exception handling or define new fragments/stages in order to not have to terminate the case. In the following, we explain how the strategies for handling following activities can be implemented based on the case life cycle shown in Fig. 7:

Continue the current case (CWC). Continuation of the current case ensures the execution of following fragments/stages. There are two possibilities when CWC is applicable: (1) Completion of the activity does not directly affect entry or exit criteria of fragments/stages being relevant for achieving

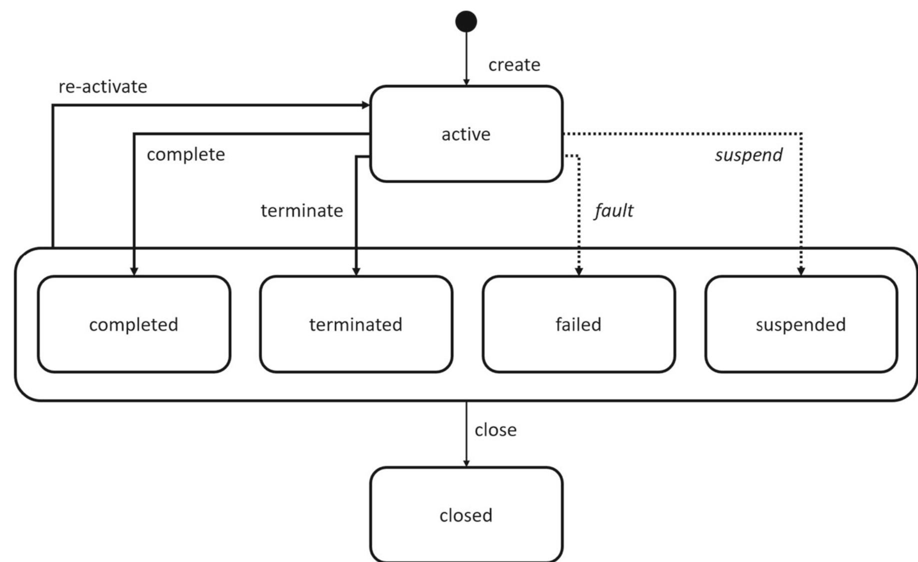


Fig. 7 Lifecycle of cases in CMMN [23] and also applicable to fCM and GSM

Table 7 Relation between exception handling at fragment/stage level and case level

	Case Level		
Fragment/Stage Level	CWC	RCC	RAC
TER	x	x	x
CON	x	x	
FAI	x	x	x
RBK	x	x	

the business goal, e.g., the activity is discretionary or part of an optional fragment. (2) Relevant fragments/stages depend on the completion of the activity, but further execution can be ensured via recovery measurements. CWC is applicable for each handling method on activity and fragment/stage level (see Table 7). The fragment/stage strategies CON and RBK imply that the fragment/stage containing the activity that caught the exception can continue to execute successfully. Thus, a continuation of the case is guaranteed. A combination with TER and FAI is possible no matter whether the fragment/stage, in which the exception occurred, is optional (e.g., indicated by a discretionary activity in CMMN), or fundamental for further execution. However, the latter requires recovery measurements in the fourth tuple in order to guarantee CWC. After exception handling, the case is re-activated and set back to state *active*.

Removing the current case (RCC). RCC enables knowledge workers to modify the current case instance at the model level by adding new modeling elements to the case, and removing existing ones. This results in a change at the execution level, as new process paths can be defined while the case as a whole continues to exist. To keep state transitions of the data objects up-to-date, recovery measurements are required (cf. R1). After removing a selection of activities, a case can then be reactivated to fulfill requirement R2. The strategy is applicable for all handling methods (cf. Table 7).

Consider the example of the exception that Corona regulations prohibit presence events (cf. Table 2). The knowledge worker decides to hold lectures online and record the sessions. Thus, written consent must first be obtained from each student to record the lecture. However, neither such an activity nor the new conditions for the activity "Hold Lecture" exist. RCC would suggest to remove the activity "Hold Lecture," replace it by an online version of it, and add a new activity to ensure the consent from the students.

The strategy requires knowledge workers to be able to select a set of activities and fragments, and remove them from the current case, i.e., set them to read-only, and be able to model and add new activities and fragments to the case at runtime. It is also possible to select all fragments/stages and remove them from the runtime database [27], because the case instance may be no longer relevant or applicable.

In fCM, it is neither possible to remove single activities nor fragments at runtime. Because it is only allowed to add new modeled fragments to the case, fCM needs to be extended in terms of functionality and knowledge workers' rights (cf. Table 4). CMMN allows knowledge workers to terminate activities at any point of time during execution. They can add additional activities and remove existing ones by changing their state to *terminated*, i.e., remove them from the runtime database. Nonetheless, knowledge workers must have the right to add additional activities that are not defined in the plan model (cf. Table 4). Similar to CMMN, GSM does not allow the case worker to change the model during runtime, thus limiting the exception handling capabilities. Therefore, this would also require an extension of case workers' permissions.

Remove all Cases (RAC). The RAC strategy is used only in exceptional cases endangering the achievement of the business goal, e.g., when laws change or a pandemic occurs. It is an escalation of the strategy RCC. Instead of a selection of activities or fragments/stages, all cases are removed from the runtime database. This is done via a state transition of the case from *failed* to *closed*. No more activities can be executed and the data objects are set to a read-only status. Additionally the case is marked as failed. Thus, RAC handles the most fatal exceptions that cannot be handled during runtime. The strategy can be implemented in all case management languages but is only applicable if the handling on fragment/stage level leads either to termination (TER) or failure (FAI) of the fragment/stage (cf. Table 7). Other strategies do not imply a closing of the case via state *failed*.

5.1.4 Recovery measurements

Recovery is important to handle or even remove the effects of an exception that has occurred. The concrete recovery measurement is defined in the fourth tuple element of a pattern. It does not cover the recovery for activities inside the fragment or stage in which the exception was triggered but for other fragments/stages included in the case model. In general, there are three strategies for recovery measures: compensation, rollback, or do nothing [27]. All of these strategies are applied in combination with the activity and case level and is implemented in different ways depending on the respective case management approach. FCM works with fragments as the flexibility component and can be added or removed during runtime. GSM and CMMN work with single activities.
Compensation (COM). Compensation can require three different activities on the part of the knowledge worker in the context of case management: (1) add flexibility components, (2) remove flexibility components, and (3) modify knowledge. The recovery measurement may be a combination of these three options, or it may include only one. We deliberately refrain from specifying the exact possibilities in the framework in order not to restrict knowledge workers in their flexibility on the one hand and to reduce complexity on the other hand. Similar to case handling strategy RCC, we need to make demands on knowledge worker's rights that are not yet present in case management languages: Adding and removing flexibility components. The possibility of knowledge modification (3) includes in fCM the manipulation of OLCs. This means knowledge workers must be authorized to adjust and change the OLC during runtime. However, state transitions of knowledge objects are not as specifically defined in GSM and CMMN as in fCM. They implement knowledge modification by the following state transitions: update, replace and delete [23]. Nonetheless, this possibilityis only needed in GSM and CMMN if the execution of activities fails because of the content of the knowledge object or its existence.
Rollback (RBK). A rollback resets the process execution to a previous process state. In the context of a recovery on the case level, it implies the re-execution of a completed fragment/stage. However, this requires that entry conditions of the related fragment/stage are fulfilled in order to enable both data flow and control flow. This can be enabled by allowing that the knowledge workers can change objects states regardless of their current data values, or by storing a snapshot for each previous process state to which the knowledge worker can go back. While in the first version, the data are overwritten, in the second one, the data are rewritten. A rollback can also include the modeling of new flexibility components which are added to the case. It is also possible to remove a set of activities before performing a rollback. Although these combinations refer to compensation techniques introduced before, we do not specify them in the pattern. We want to put the rollback strategy in the focus of the pattern and give knowledge workers the flexibility to decide on their one how to perform it (cf. Requirement R5). Nonetheless, there are different techniques concerning how GSM, CMMN, and fCM implement a rollback we want to mention. In fCM a rollback can be performed by resetting the process to one the snapshots taken at the execution start of each fragment. However, it is important to control the flow of data. Data objects change their states as execution progresses and can therefore endanger a successful rollback. For this reason, the action of rollback always implies a change in the states of data objects. GSM and CMMN offer the concept of milestones which can be used to perform a rollback. Nevertheless, rolling back the process to a milestone could be a large step back in execution, since milestones can complete large process sections.
Do nothing (NIL). If the exception does not have any serious consequences, in most cases no recovery action is needed *(NIL)*. It is up to the case workers if they want to add new tasks or remove ones. The framework does not recommend any recovery measurements in terms of compensation or rollback.

In general, compensation techniques such as adding a fragment or an activity can always be performed during runtime but it is not always part of exception handling. Table 8 provides an overview of the relation between the handling on case and fragment/stage level and the recovery measurements that are applied on case level.

Consider the handling strategies CON and RBK at fragment/stage level. Then, case handling methods can be applied as follows: If the case continues in its execution (CWC), no recovery is needed (NIL) since any effects are already handled within the fragment/stage. A combination with RCC allows knowledge workers to, for example, add fragments during runtime. This might be useful for exceptions affecting

Table 8 Relation between exception handling at fragment/stage level, case level, and recovery

	CWC				RCC				RAC		Case Level
	TER	CON	FAI	RBK	TER	CON	FAI	RBK	FAI	TER	Fragment/Stage Level
COM	x		x		x	x	x	x	x	x	Recovery
RBK	x		x		x		x				Measurements
NIL	x	x	x	x							

Table 9 Classification of patterns by Exception Type

Activity Failure	Deadline Expires	Failure in Resource Allocation	External Trigger	Constraint Violation
SFS-CON-CWC-NIL	SFS-CON-CWC-NIL	SFA-CON-CWC-NIL	SFS-CON-CWC-NIL	SFS-CON-CWC-NIL
SFC-CON-CWC-NIL	SFS-CON-RCC-NIL	SMV-CON-CWC-NIL	SFS-CON-RCC-COM	SFS-CON-RCC-COM
SFF-CON-CWC-NIL	SFA-CON-CWC-NIL	SMV-RBK-CWC-NIL	SFF-CON-CWC-NIL	SFF-CON-CWC-NIL
SFF-FAI-CWC-NIL	SMV-CON-CWC-NIL	SFF-CON-CWC-NIL	SFF-FAI-CWC-COM	SFF-FAI-CWC-COM
SFF-FAI-CWC-COM	SMV-RBK-CWC-NIL	SFF-FAI-CWC-COM	SFF-FAI-RCC-COM	SFF-FAI-RCC-COM
SFF-RBK-CWC-NIL	SFF-CON-CWC-NIL	SFF-FAI-RCC-COM	SFF-FAI-RCC-RBK	SFF-FAI-RCC-RBK
SFT-TER-CWC-COM	SFF-FAI-CWC-COM	SFF-FAI-RCC-RBK	SFF-FAI-RAC-COM	SFF-FAI-RAC-COM
SFT-TER-CWC-NIL	SFF-FAI-RCC-COM	SFF-FAI-RAC-COM	SFC-CON-CWC-NIL	SFC-CON-CWC-NIL
SFT-TER-RCC-COM	SFF-FAI-RCC-RBK	SFC-CON-CWC-NIL	SFT-CON-CWC-NIL	SFT-CON-CWC-NIL
SFT-TER-RCC-RBK	SFF-FAI-RAC-COM	SFT-CON-CWC-NIL	SFT-TER-CWC-COM	SFT-TER-CWC-COM
SFT-TER-RAC-COM	SFC-CON-CWC-NIL	SFT-TER-CWC-COM	SFT-TER-CWC-NIL	SFT-TER-CWC-NIL
SFT-RBK-CWC-NIL	SFC-CON-RCC-COM	SFT-TER-CWC-NIL	SFT-TER-RCC-COM	SFT-TER-RCC-COM
SFT-CON-CWC-NIL	SFT-TER-CWC-NIL	SFT-TER-RCC-COM	SFT-TER-RCC-RBK	SFT-TER-RCC-RBK
	SFT-TER-RCC-COM	SFT-TER-RCC-RBK	SFT-TER-RAC-COM	SFT-TER-RAC-COM
	SFT-TER-RCC-RBK	SFT-TER-RAC-COM	SFT-RBK-CWC-NIL	SFT-RBK-CWC-NIL
	SFT-TER-RAC-COM	SFT-RBK-CWC-NIL		
	SFT-RBK-CWC-NIL			
	SFT-TER-CWC-COM			

only other fragments/stages of the case. Therefore, compensation (COM) is needed. RAC, however, cannot be applied if CON and RBK is suggested at fragment/stage level.

In contrast, the strategies termination (TER) and failure (FAI) at fragment/stage level can be combined with all three methods on case level. CWC suggests the continuation of fragments/stages of the case not containing the activity that have triggered the exception. In this case, recovery measurements may be useful but are not required. For example, if the fragment/stage that was terminated or set to failed was marked as optional, recovery is often not needed (NIL). However, if the fragment/stage containing the activity that have triggered the exception is fundamental for process execution, recovery in form of compensation (COM) or a rollback (RBK) are possible. As explained above, RCC requires recovery measurements. Therefore, it is not possible to combine RCC with NIL but with COM and RBK. Removing the entire case (RAC) is only possible in combination with a compensation technique (COM) to fulfill requirement R2 (goal achievement). Knowledge workers must find a solution for that exceptional case of handling.

5.2 Classification of patterns

According to the allowed combinations between case level, fragment/stage level and the recovery measures introduced before, there are many possibilities to map patterns to a given exception type. However, not every pattern can be applied to each exception type. Table 9 provides an overview of the support by the exception handling patterns per exception type. It is derived from the mapping of the exception patterns introduced by Russell et al. [27].

5.3 Functional design of an exception service

Our framework offers the possibility to provide structured support for knowledge workers in case of major exceptions in the case execution. In the following, we describe the functional design of an exception service in a case management system supporting the case execution with the help of an example exception shown in the mock-up in Fig. 8.

The exception service including a UI needs to have (a) a *notification function* to alert knowledge workers immediately

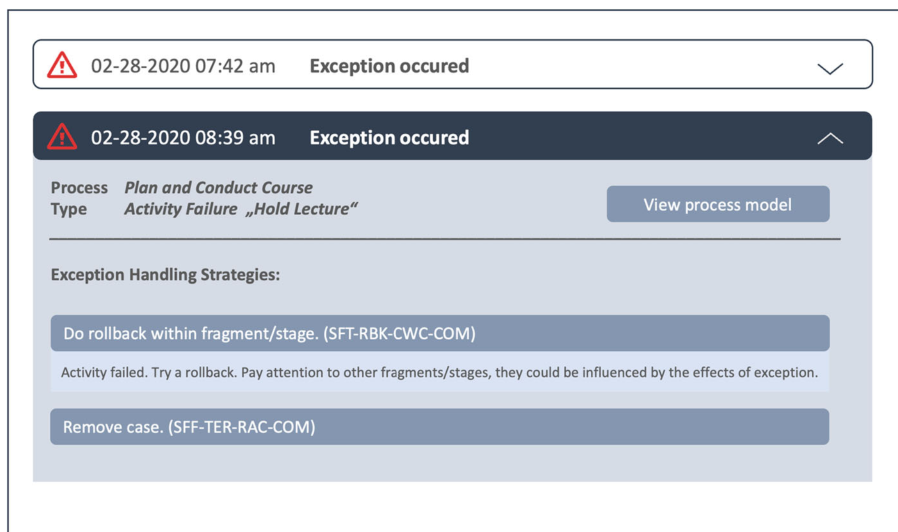


Fig. 8 Mock-up of UI showing occurrence of activity failure

whenever an exception occurs, (b) a *function to select the exception type*, (c) an *overview of all possible pattern* for that exception type with their costs and consequences, (d) a *modeling and configuration space* to design new fragments or modify existing ones, (e) a *simulation* option to verify the changes of the process model and (f) a *compiling* option to integrate the changes to the process model.

If an exception occurs, we assume that the knowledge worker categorizes it into one of the available exception types (or it is done automatically with the help of natural language processing techniques, etc.). Based on the exception type, the corresponding exception handling patterns are identified with the help of Table 9. The patterns are then provided to the knowledge worker in the user interface. For example, Fig. 8 shows a mock-up for the situation when an exception occurs during the activity execution of *Hold Lecture* (cf. motivating example in Sect. 3). The knowledge worker has already chosen the exception type *Activity Failure*. At the bottom, all possible strategies for handling the exception suggested by the framework are listed. A short headline explains the core idea of the suggested handling, but more information is provided for further explanation. In this example, the framework suggests a rollback within the fragment/stage.

Having the pattern overview, the knowledge worker can walk though them and decide on a handling strategy. In our example, the knowledge worker would apply the suggested strategy by forcing the activity to the state *terminated* and perform a rollback by going back, for example, to activity *Prepare Course Content* in order to record videos for an online course (**SFT-RBK**-CWC-COM). The pattern also reminds him or her to have a look at other fragments/stages that might be affected (SFT-RBK-**CWC-COM**).

Once a pattern is selected, the necessary state transitions of the activity and the actions at the case and fragment/stage level can be automatically performed by the exception service in interaction with the case management system. The knowledge workers do not have to handle them manually, e.g., ensuring a valid rollback. Nevertheless, compensation methods and the extent of the rollback are selected by the knowledge worker.

6 Evaluation

In this section, we want to present a first evaluation of our approach to check the coverage of our exception handling framework. We analyzed two real-world scenarios with the help of knowledge workers involved in these scenarios: (1) the preparation and execution of a university course as introduced in Sect. 3 and (2) a project planning process. Project planning includes capacity planning, determination of a project team, and setting up a IT project plan before the kick-off of a project. First, our elicitation procedure is explained. Then, the results are presented and discussed with the first use case being explained in detail.

6.1 Procedure

For the evaluation of coverage, we interviewed knowledge workers to ask them about their experience with major exceptions and how they usually handle them.

For the first use case, we interviewed two teaching assistants (TAs) who were involved in the organizations of different courses at the same university. The second scenario was discussed with one knowledge worker that is involved in the project planning process.

We followed in both use cases the following procedure: In the first interview session, we identified the case model that were also validated with the knowledge workers. In a second interview session, the exceptions and typical handling

strategies were elicited. Afterwards, we matched exception and handling strategy to possible patterns (according to Table 9) and discussed in the group of co-authors whether they were reasonable in terms of content and handling. The mapping of a pattern to a handling of the knowledge worker is initially based on the type of exception. Table 9 shows the subset of all patterns which are applicable for the respective exception type. Based on the exception handling of the knowledge worker, it is then interpreted what the handling means for the individual activity, for the fragment/stage and for the case. Types of activities and fragment/stages (optional or relevant) have to be taken into account. In general, three approaches are possible to map a pattern to a handling strategy defined by knowledge workers: (1) start with the activity which caught the exception, (2) start with the fragment/stage containing this activity, and (3) start with the handling on case level. These approaches simplify the mapping by specifying a beginning of the exception handling. Strategies for the other levels can then be derived from that.

6.2 Results

Conduct a University Course. For the first use case, the university course organization and execution, twelve exceptions and their typical handling strategies were identified; all of them could be successfully assigned to a pattern. Furthermore, it was possible to map each exception to at least one activity that could trigger it. Table 10 shows an overview of the exceptions mentioned during interviews, their classification into an exception type and a suitable pattern. In

Table 10 Example manual mapping of real world exceptions to corresponding exception handling pattern for the *Conduct a University Course* use case

Activity	Exception (Type)	Handling by Knowledge Worker	Patterns
Hold Lecture	(1a) Corona regulations prohibit presence events (Activity Failure)	Complete change of style of course (present to online), use of prerecorded material	SFT-RBK-CWC-NIL
	(1b) Students have questions after or outside of the course	Unstructured solution: directly answer questions	SFS-CON-RCC-COM
	(External Trigger)	Structured solution: offer a scheduled time for Q&A	SFS-CON-RCC-COM
	(1c) Overbooking of the lecture hall	Cancel course	SFF-FAI-RCC-COM
	(Resource Allocation)	Request new room and possibly reschedule	SFT-RBK-CWC-NIL
	(1d) Errors in course material (External Trigger)	Fix errors and send updated version of slide to students	SFS-RBK-CWC-NIL
	(1e) Fire alarm (Activity Failure)	Leave building and wait for further instructions, continue lecture if possible	SFE-CON-CWC-NIL
Conduct Exercise	(2a) Tutor gets sick (Resource Allocation)	Find replacement person and date	SFS-CON-CWC-NIL
	(2b) Technical error of projector	Get technical support	SFT-CON-RCC-COM
	(Resource Allocation)	Switch to mobile projector if available	SFS-CON-CWC-NIL
Seek Tutors	(3a) End of semester and not enough tutors	Allow larger tutor groups	SFC-CON-CWC-NIL
	found (Deadline expires)	No graded exercises	SFF-CON-CWC-NIL
Grade Exam	(4a) Unexpected answers in exam (Activity Failure)	Find objective measure for grading	SFS-CON-CWC-NIL
Conduct Exam	(5a) Cheating on exam (Constraint Violation)	Look for evidence and decide whether student fails or pass the exam	SFS-CON-RCC-COM
	(5b) Student gets sick, no exam can be handed out (Resource Allocation)	Get illness validation document and schedule a catch-up date	SFS-CON-RCC-COM
Plan course	(5c) No person for guest lecture is found	Find replacement lecturer	SMV-RBK-CWC-NIL
	(Resource Allocation)	(5d) Cancel guest lecture	SFT-CON-CWC-NIL

the following, we will explain the mapping of the handling strategies for three above described approaches for several examples in detail.

(1) Start with activity. When not enough tutors could be found as originally planned and the course has already started, the activity *Seek Tutors* (see Table 10 Exception 3a)) cannot be terminated since its exit condition (enough tutors found) is not fulfilled. Knowledge workers then suggest several handling options for the exception: allow larger groups of exercises, i.e., more students per tutor than originally intended, or change to non-graded exercises.

If larger groups are formed, the task of the *Seek Tutors* activity can be considered as completed because knowledge workers compensate the workload by increasing the number of students per tutor. Therefore, a force-complete (SFC) at activity level fits best the knowledge workers' strategy. According to Table 6, the fragment/stage "Exercise" then continues in its execution (CON). The handling of knowledge workers implies that the case also continues in execution (CWC) and no further recovery is required (NIL). The pattern SFC-CON-CWC-NIL thus fits the appropriate handling strategy.

If one decides not to grade the exercises, the tutor search may become obsolete and is therefore set to state *failed* (SFF). Only this state ensures that no related fragments/stages and activities, e.g., tutor payment, are enabled. Since the activity *Seek Tutors* is optional, the fragment/stage can continue in its execution (CON) as well as the case (CWC). Recovery is not required (NIL). This results in the pattern SFF-CON-CWC-NIL.

(2) Start with fragment/stage. Corona regulations prohibit in-person lectures, and the activity *Hold Lecture* catches an exception of type *Activity Failure* (see Table 10 Exception 1a)). Appropriate handling suggests a complete restructuring of the lecture style. This means that originally planned lectures have to be re-prepared in terms of venue and setting, i.e., activity *Organize Venue* has to be re-executed. However, the content does not change. Thus, we interpret the handling of the knowledge worker as a rollback within the fragment (RBK as second pattern element). According to Table 6, the exception handling at activity level enforces the state transition of the activity Hold Lecture into *failed*, *terminated*, or *enabled*. The state *enabled* is inappropriate here because the activity is not enabled until there is a venue. The transition to *failed* means that "no subsequent work items are triggered" [27]. However, since the activity is essential for further execution, i.e., it is neither part of optional fragment nor marked as discretionary, it is forced to state *terminated* and can thus be executed again at a given time (SFT as first pattern element). All other fragments/stages are still executed and the case is continued (CON as third pattern element). There are no additional recovery actions on case level (NIL as fourth pattern element). Thus, the pattern SFT-RBK-CON-NIL exactly covers the handling strategy of the knowledge worker. This mapping strategy also applies to the exception of overbooking of the lecture hall (see Table 10 Exception 1c)). It is a wrong resource allocation and causes the canceling of the course meaning that the fragment/stage "Course" cannot be executed anymore. Moreover, the fragment/stage should not trigger any other fragments/stages in the case since the course does not take place anymore. Therefore, the right exception handling on fragment/stage level is failure (FAI). Accordingly, the activity *Hold Lecture* is set to state *failed*, and the case is removed from the runtime database (RCC). Compensation (COM) is required if, for example, tutors were already employed. This results in the pattern SFF-FAI-RCC-COM.

3) Start with case level. Consider the exception of students having questions after or outside of the course (see Table 10 Exception 1b)). It is caught by the activity *Hold Lecture* and from type *External Trigger*. Two handling strategies are suggested by knowledge workers: directly answer upcoming questions, or the offer of a scheduled Q&A session.

We interpret the first option as providing a forum where questions can be asked enabling lecturers as well as tutors to respond directly. Since setting up a forum is not yet considered in the model, this strategy requires a change in the case (RCC). This can be done, for example, by adding a new fragment/stage. Modification of the case in this context neither affects the fragment/stage containing the activity which caught the exception nor the activity itself. Therefore, the fragment/stage continues in execution (CON) and the activity *Hold Lecture* is forced to state *started* meaning that it continues in its execution. Compensation on case level (COM) is required as defined in Table 8. The pattern SFS-CON-RCC-COM thus realizes the handling by knowledge workers.

Offering a scheduled time for Q&A also requires additional activities to organize a venue. The handling strategy RCC on case level allows to add further elements to the process model at runtime, and thus, fits best the appropriate handling. Similar to the handling above, the activity *Hold Lecture* can continue in its execution and is therefore forced back to state *started*. The fragment/stage also continues in execution (CON). This results in the pattern SFS-CON-RCC-COM.

Project Planning Process. For the second use case, we were able to identify seven exceptions in collaboration with the knowledge worker. The handling strategies of the knowledge workers can be covered by the patterns (cf. Table 11). Many of the listed exceptions, e.g., absence due to illness or the request for re-prioritization, can occur at any time and can therefore be assigned to several activities. Thus, the activities listed in Table 11 are only examples. Moreover, it is important to note that the methods used by the knowledge workers mostly involve collaboration with other process participants.

Table 11 Example manual mapping of real world exceptions to corresponding exception handling pattern for the *Project Planning Process* use case

Activity	Exception (Type)	Handling by Knowledge Worker	Patterns
Internal Kick-off	(1a) Employees absent due to illness (Resource Allocation)	Find someone who is available	SFA-CON-CWC-NIL
	(1b) Customer changes period of project (External Trigger)	Redefine capacity planning and determine new project team	SFT-RBK-CWC-NIL
Preselect Project Team	(2a) Employees absent due to parental leave (Resource Allocation)	Find replacement person and date	SFA-CON-CWC-NIL
	(2b) Staff shortage due to missing or already utilized competences (Resource Allocation)	Office department books freelancer	SFT-CON-RCC-COM
Set up Project Plan	(3a) New project order with high priority (External Trigger)	Direct reprioritization of projects	SFT-TER-RCC-RBK
Determine Project Manager	(4a) Project manager declines to participate in project (Activity Failure)	Team manager checks whether refusal is granted	SFF-TER-RCC-COM
Determine Project Team	(5a) Project team member declines to participate in the project (Activity Failure)	Team manager checks whether refusal is granted	SFF-TER-RCC-COM

6.3 Discussion

Our first observation is that even though the interviewees agreed on the case model as being complete, a multitude of exceptions could be found, which are not covered in the case model. This supports our assumption that an additional exception handling support is necessary.

During our post-processing step, we were able to map all exceptions and their handling to at least one pattern of our framework. As mentioned in our results section, we only provided an example of a matching pattern to a concrete exception handling proposed by the interviewers. An example is the larger groups exception handling, that was matched to the pattern SFC-CON-CWC-NIL. Here, we argued that the task *Seek Tutors* can be considered as completed, as the goal of the original task was fulfilled. However, depending on the semantic or a concrete implementation, the task *Seek Tutors* might also be considered as terminated, which would mean the first pattern could also been the force-termination SFT. This means the pattern belonging to a concrete exception handling is unambiguous and can thus adapt to the unique nature of a process and its semantics/implementation. Therefore, we can conclude that our framework is able to represent a realistic and reasonable approach to identify and propose exception handling measures that can support knowledge workers.

In comparison with our patterns, the handling strategies of the knowledge workers are concrete for both use cases. The presented framework of this research work does not define exactly how an exception has to be handled in terms of concrete specifications, e.g., how compensation has to be executed exactly. Furthermore, it does not include any aspect of collaboration or delegation of exception handling. Some exceptions of the second use case, e.g., staff shortage, are not handled by the knowledge workers who execute the activity but by other process participants. Although, a more precise specification and the introduction of a more detailed set of rules would better support individual processes, it would limit the general applicability of the patterns. Deciding on the specific handling should be the responsibility of the knowledge workers since we do not limit the flexibility of knowledge workers in the case execution (cf. R5).

The presented exception handling patterns provide knowledge workers a structural way of handling strategies ensuring further process execution by specifying relation of different levels that have to be considered. For instance, in the above pattern SFC-CON-CWC-NIL, which is used to handle the exception “Not enough tutors found” by allowing larger groups of exercises, the activity *Seek Tutors* is set to status *completed*. This enables the fragment or stage to continue in its execution. A manual termination of the fragment/stage, for example, would require a manual completion or termination of all following activities and knowledge worker would have to manage the workload on their own. To avoid unnecessarily costly alternative options, the framework specifies rules, so that knowledge workers are made aware how to handle which level; a highly relevant support especially for more complex case models.

Finally, we want to discuss the threats of validity for the evaluation results. As we interviewed only a representative set of knowledge workers for the two use cases, the set of provided exceptions might not be complete for the use cases, but they provide a representative relevant set, for which we

could check the coverage. Furthermore, the mapping from the handling strategy of the knowledge worker to a pattern could be influenced by bias. We tried to mitigate this subjectivity by discussing each mapping within the group of authors.

7 Conclusion

In this work, we presented a framework for exception handling that can be applied to case management. The framework is based on existing work on exception handling for strictly structured business processes. We adapted the existing work for case management. In order to ensure a broad application to different case management notations, we analyzed three different modeling languages. We generalized their characteristics in one exception handling framework, which was then used on a real-world business process by interviewing knowledge workers. In contrast to the original three elements *activity-level, case-level, recovery*, we added a fourth layer to cover the additional fragment/stage introduced by case management approaches, resulting in four elements *activity-level, fragment/stage-level, case-level, recovery*. Whereas the first three elements describe the nature of the exception and its influence on the corresponding levels, the last element describes the nature of the recovery measure. An alternative solution would be to integrate the last element into the third element, the *case-level* as both the exception and its recovery measure operate on the case-level. We decided for the separation of these two elements, similar to [27], because we think that a clear separation of the recovery and exception description has its advantages in the application.

Resulting from the evaluation, we can conclude that our framework covers all observable exceptions and their handling. It could be observed that the patterns do not give concrete actions on how to handle exceptions but a source of inspiration and a frame supporting the knowledge worker in applying the exception handling in such a way that no problem occurs in the execution of a case and its elements. Currently, we only conducted our evaluation on two cases and checked the coverage of the patterns. In future, we plan to extend this to other cases to better generalize and also to check the usefulness of the patterns in a real-world use case with knowledge workers.

In the current framework, knowledge workers still have some manual work: on the one hand, they have to select the exception type for getting the exception handling possibilities, and on the other hand, they have to select the most useful handling pattern from multiple possibilities for the occurred exception. The wide range of patterns provides knowledge worker flexibility and freedom in specifying how to handle an exception; still, however, the high number of choices also increases the complexity. In future, we want to more automatically support the classification of the exception types and the selection of the most useful patterns for a certain situation with the help of machine learning techniques, e.g., learning from historic data. Additionally, the framework focuses on stage-oriented case management languages; in future, its applicability to constraint-oriented case management, such as DCR graphs [29], and object-oriented case management, such as Philharmonic Flow [17] could be assessed.

All in all this work presents a novel framework, which adapts exception handling patterns for case management the first time. Moreover, this is a first step to support knowledge workers in handling exception appropriate while managing a case by offering abstract handling strategies and giving automatic support in keeping a case in a valid execution state.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Agostini, A., De Michelis, G.: Improving Flexibility of Workflow Management Systems, pp. 218–234. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
2. Andaloussi, A.A., Burattin, A., Slaats, T., Kindler, E., Weber, B.: On the declarative paradigm in hybrid business process representations: a conceptual framework and a systematic literature study. Inf. Syst. **91**, 101505 (2020)
3. Andree, K., Ihde, S., Pufahl, L.: Exception handling in the context of fragment-based case management. In: Enterprise Business-Process and Information Systems Modeling, pp. 20–35. Springer, New York (2020)
4. Andrews, K., Steinau, S., Reichert, M.: A tool for supporting ad-hoc changes to object-aware processes. In: 2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW), pp. 220–223. IEEE (2018)
5. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches. J. Data Semant. **4**(1), 29–57 (2015)
6. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer, New York (2018)
7. Fahland, D., Reisig, W.: ASM-based semantics for BPEL: the negative control flow. In: Abstract State Machines, pp. 131–152 (2005). https://dblp.org/db/conf/asm/asm2005.html#FahlandR05
8. Fahland, D., Woith, H.: Towards process models for disaster response. In: Business Process Management Workshops, BPM 2008 International Workshops, Milano, Italy, September 1-4, 2008. Revised Papers, vol. 17, pp. 254–265. Springer (2008)

9. Fahland, D.: From scenarios to components. Ph.D. thesis, Humboldt University of Berlin (2010)
10. Gonzalez-Lopez, F., Pufahl, L.: A landscape for case models. In: Reinhartz-Berger, I., Zdravkovic, J., Gulden, J., Schmidt, R. (eds.) Enterprise, Business-Process and Information Systems Modeling, pp. 87–102. Springer International Publishing, Cham (2019)
11. Haarmann, S., Podlesny, N.J., Hewelt, M., Meyer, A., Weske, M.: Production case management: a prototypical process engine to execute flexible business processes. BPM (2015)
12. Hauder, M., Pigat, S., Matthes, F.: Research challenges in adaptive case management: a literature review. In: 2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations, pp. 98–107 (2014)
13. Hewelt, M., Pufahl, L.,Mandal, S. et al.: Toward a methodology for case modeling. Softw. Syst. Model **19**, 1367–1393 (2020). https://doi.org/10.1007/s10270-019-00766-5
14. Hewelt, M., Weske, M.: A hybrid approach for flexible case modeling and execution. In: International Conference on Business Process Management, pp. 38–54. Springer (2016)
15. Holfter, A., Haarmann, S., Pufahl, L., Weske, M.: Checking compliance in data-driven case management. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) Business Process Management Workshops, pp. 400–411. Springer International Publishing, Cham (2019)
16. Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., Heath, F.T., III., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., et al.: Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In: DEBS 2011, pp. 51–62. ACM, New York (2011)
17. Künzle, V., Reichert, M.: PHILharmonicFlows: towards a framework for object-aware process management. J. Softw. Maint. Evolut. Res. Pract. **23**, 205–244 (2011)
18. Kurz, M., Fleischmann, A., Lederer, M., Huber, S.: Planning for the unexpected: exception handling and bpm. In: Fischer, H., Schneeberger, J. (eds.) S-BPM ONE - Running Processes, pp. 123–149. Springer, Berlin Heidelberg, Berlin, Heidelberg (2013)
19. Luo, Z., Sheth, A., Kochut, K., Miller, J.: Exception handling in workflow systems. Appl. Intell. **13**, 125–147 (2000)
20. Marin, M.A., Hauder, M., Matthes, F.: Case management: an evaluation of existing approaches for knowledge-intensive processes. In: BPM Workshops 2015, pp. 5–55. Springer, New York (2015)
21. Nigam, A., Caswell, N.S.: Business artifacts: an approach to operational specification. IBM Syst. J. **42**(3), 428–445 (2003)
22. OMG: Notation BPMN Version 2.0. OMG Specification, pp. 22–31. Object Management Group, Needham (2011)
23. OMG: Notation CMMN Version 1.0. OMG Specification. Object Management Group, Needham (2014)
24. Pesic, M., Schonenberg, H., Van der Aalst, W.M.: Declare: full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), pp. 287–287. IEEE (2007)
25. Pufahl, L., Ihde, S., Glöckner, M., Franczyk, B., Paulus, B., Weske, M.: Countering congestion: a white-label platform for the last mile parcel delivery. In: International Conference on Business Information Systems. Springer, Cham, pp. 210–223 (2020)
26. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies. Springer Science & Business Media, New York (2012)
27. Russell, N. C., Aalst, van der, W. M. P., Hofstede, ter, A. H. M.: Workflow exception patterns. In: Dubois, E., Pohl, K. (Eds.), Advanced Information Systems Engineering: 18th International Conference, CAiSE 2006, Luxembourg, Luxembourg, June 5-9, 2006: Proceedings (pp. 288–302). (Lecture Notes in Computer Science; vol. 4001). Springer (2006). https://doi.org/10.1007/11767138_20
28. Sid, I., Reichert, M., Ghomari, A.R.: Enabling flexible task compositions, orders and granularities for knowledge-intensive business processes. Enterp. Inf. Syst. **13**(3), 376–423 (2019)
29. Slaats, T., Mukkamala, R.R., Hildebrandt, T., Marquard, M.: Exformatics declarative case management workflows as dcr graphs. In: Business Process Management, pp. 339–354. Springer, New York (2013)
30. Steinau, S., Marrella, A., Andrews, K., Leotta, F., Mecella, M., Reichert, M.: Dalec: a framework for the systematic evaluation of data-centric approaches to process management software. Softw. Syst. Model. **18**(4), 2679–2716 (2019)
31. van der Aalst, W.M.P., Berens, P.J.S.: Beyond workflow management: Product-driven case handling. In: Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work, GROUP '01, p. 42–51. Association for Computing Machinery, New York, NY, USA (2001)
32. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data Knowl. Eng. **66**(3), 438–466 (2008)
33. Weber, B., Pinggera, J., Torres, V., Reichert, M.: Change patterns in use: a critical evaluation. In: Enterprise, Business-Process and Information Systems Modeling, pp. 261–276. Springer, New York (2013)
34. Zimmermann, B., Doehring, M.: Patterns for flexible bpmn workflows. In: Proceedings of the 16th European Conference on Pattern Languages of Programs, pp. 1–9 (2011)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Kerstin Andree is a master student at the Hasso Plattner Institute, University of Potsdam, Germany. During her studies, she focuses on flexible business processes, especially roles and responsibilities, and exception handling of case management approaches at runtime. Currently, she is working on design-time support for fragment-based case management.

Sven Ihde is a research assistant at the Chair for Business Process Technology at the Hasso Plattner Institute, University of Potsdam, Germany. His current research is focused on increasing the robustness and efficiency of business process and their execution. This includes the handling of exception at runtime as well as the management of resources—especially optimized resource allocations—in organizations.

Mathias Weske is chair of the business process technology research group at Hasso Plattner Institute at the Digital Engineering Faculty, University of Potsdam, Germany. The research group aims at addressing real-world problems in business process management with formal approaches and engineering useful prototypes. His research focuses on the engineering of process oriented information systems, process mining, and event processing. The BPT research group has a track record in engineered prototypes with a significant impact on research and practice, including projects like Oryx and jBPT. He co-founded Signavio and he is business angel at Synfioo. Dr. Weske is author of the first textbook on business process management and he held the first massive open online course on the topic in 2013. With Matthias Kunze, he published a textbook on behavioral models. He is on the Editorial Board of Springer's Computing journal, and he is a founding member of the steering committee of the BPM conference series and, since September 2017, chair of the steering committee.

Luise Pufahl is a postdoctoral researcher in the Software and Business Engineering group at TU Berlin, Germany. Her current research interests are flexible business processes, process analysis, and improvement, and resource management in business processes based on operations research, simulation, and machine learning techniques. Application domains are mainly health care and logistics. Her publication record includes more than 30 articles published in peer- reviewed journals, conferences and workshops, and she was in the winning teams of the last two Business Process Intelligence Challenges in 2020 and 2019.