Exception Handling in the Context of Fragment-Based Case Management

Kerstin Andree[1(✉)], Sven Ihde[1], and Luise Pufahl[2]

[1] Hasso Plattner Institute, University of Potsdam, 14482 Potsdam, Germany
kerstin.andree@student.hpi.de, sven.ihde@hpi.de
[2] Software and Business Engineering, Technische Universitaet Berlin, Berlin, Germany
luise.pufahl@tu-berlin.de

Abstract. Case Management supports knowledge workers in defining, executing, and monitoring the handling of their cases, e.g. in healthcare or logistics. Fragment-based case management (fCM) allows to define a case model with the help of several process fragments, which can be flexible combined at run-time based on case characteristics and the case worker's intuition. Cases are often influenced by unknown exception, e.g., the sudden change of patient condition's or a storm delaying transports. So far, fCM only reacts to known circumstances. In this paper, we want to extend fCM by an exception handling approach. Thereby, existing exception patterns for workflow systems are used and extended by the fragment-level for handling unknown events. In order to enable direct integration and avoid a duplication of semantics, precise rules are specified in order to clarify how to extend which pattern in detail. The applicability of the developed exception handling technique is exemplified on a last mile delivery for parcels.

Keywords: Case management · Exception handling · Business process modeling · Flexible process automation

1 Introduction

Business Process Management (BPM) enables companies to optimize their processes in such a way that an overall business goal is achieved. The main artifact of BPM are business process models [3]. Traditional control-flow oriented process models represent all possible execution paths of a business process and provide a complete description of possible alternatives.

The discrepancy of a business process between the planned flow and the reality is called an exception [12]. For a successful process execution, exceptions occurring during run-time (e.g., weather changes, missing data) need to be handled. Standard process modeling languages, such as Business Process Model and Notation (BPMN) [17], offers concepts to capture and handle exception. However, they often lead to difficulties to read complex process models [6].

S. Nurcan et al. (Eds.): BPMDS 2020/EMMSAD 2020, LNBIP 387, pp. 20–35, 2020.
https://doi.org/10.1007/978-3-030-49418-6_2

A structured overview on the capabilities of event handling in existing business process management systems (BPMSs) was offered by Russell et al. [20].

Another more flexible approach for process execution is Case Management, in which so-called *knowledge workers* determine the exact process path by their decisions at run-time and enable a non-deterministic process execution [10,14]. It supports variant-rich business processes, such as healthcare or logistic processes, and should by its nature already support known exceptions. Motahari-Nezhad and Swenson [16] distinguish between adaptive and production case management; whereas the first one assumes that knowledge workers work very independently based on their knowledge which can quickly change, in the latter certain structure in the work exists which can be also defined at design time. *Fragment-based Case Management* (fCM) offers one way of implementing the Case Management approach as a hybrid variant between the two paradigms [8,9]. Nevertheless, it keeps the control-flow-based approach of process models in parts to ensure that certain processes are executed correctly. A *case* of a fCM application consists of several smaller processes, the so-called fragments, which are designed in advance. Only within these fragments, the sequential flow of individual activities is relevant. Knowledge workers can then flexibly combine those during case execution in the fCM engine[1] and thereby determine the concrete process path. So, it is not predictable how the process will look like in detail at run-time and this can lead to unexpected behaviour during execution.

Already today, exceptions can be handled by adding new fragments [9], however, a structured Exception Handling approach, such as the exception patterns for BPMSs [20] does not yet exist for the fCM engine. There are no instructions or support to which knowledge workers can refer back when unexpected or even unknown events occur. A delay during the handling of cases due to incorrect data or missing information leads to unnecessary costs for organizations. That makes a good Exception Handling very important for business processes. This work provides a structured approach for handling exceptions in a fCM application which can be used to implement an user interface (UI) for supporting knowledge workers to react fast and structured to an unexpected event. Based on the exception pattern of Russell et al. [20] which fits very well for workflow systems, it provides a suitable extension of those to make them applicable to fCM.

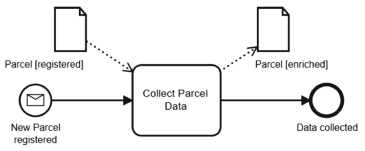
For better understanding, the explained approach is illustrated by an application example from the last mile delivery, an alternative delivery approach of parcels with a pickup place infrastructure researched in the SMile[2] project. In the remainder, this example is used to explain the concepts of fCM and further motivation in Sect. 2. The background on exception handling as well as used abbreviations of Russell et al. [20] are summarized in Sect. 3. The fourth section of this paper explains the concept of exception handling approach for fCM at run-time. Finally, we provide an application of the explained concept to the last mile scenario and give an example of a possible UI.

[1] https://github.com/bptlab/chimera.
[2] http://smile-project.de/smile-projekt/.

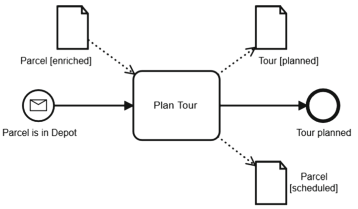
2 Fragment-Based Case Management and Motivation

To better understand the concept of fCM as well as its advantages and disadvantages in the context of exception handling, the last mile delivery scenario researched in SMile project [18] is explained briefly in this section by using a simplified fCM model. fCM reuses a subset of BPMN modeling concepts [17].

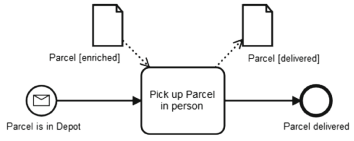
SMile tackles the problem of the last mile delivery which is defined as the movement of parcels from a high-capacity freight station or port to their final destination. The last mile takes up to 28% of the total delivery costs [19] because recipients are often not at home during delivery time, such that increased delivery tries or additional storage of deliveries are necessary. With a pick-up place infrastructure near to recipients, where parcels are delivered first, SMiles explores the last mile for parcels more efficient and user friendly [18]. Either, recipients can collect their parcels themselves or use the service of parcel delivery at a desired time frame by a local carrier, whereby the focus of the project is the latter one.



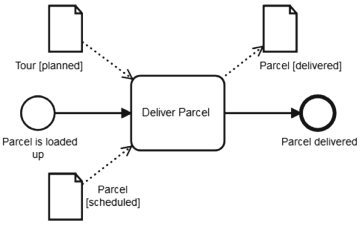
(a) Fragment 1: Collecting Parcel Data



(b) Fragment 2: Plan Tour



(c) Fragment 3: Pickup by the Customer



(d) Fragment 4: Deliver Parcel

Fig. 1. Simplified fCM model of SMile use case

Fragment-based case management consists of four basic concepts [6,8]: process fragments describing the flow of a case, the goal state describing when a case is finished, a data class diagram describing relevant data for a case and their structure, and finally, the object-life cycles for each data type describing the allowed changes on them.

In Fig. 1(a), we have visualized four fragments, in which the last mile process is split. The fragments are either triggered by external events (shown as message events) or by certain available data. The first fragment is triggered by an event called *New Parcel registered*, which is sent by the sender of the parcel. For registered parcels, the recipients are notified and asked when they want to receive their parcel. This is what happens in the activity *Collect Parcel Data*, which transfers the data object *Parcel* from its state *registered* to *enriched*. As soon as the information is received that the parcel has arrived at the depot (i.e. the pick-up place, e.g., a gas station or a greengrocer just around the corner), the parcel can be planned (cf. Fig. 1(b)), if it is *enriched*, into a delivery tour for a local carrier. The activity *Plan Tour* calls a complex planning service. This service collects at a certain point in time all parcels requesting a planning with the same postal code and looks for a suitable carrier who delivers in this area. It plans a route which ensures that each parcel will be delivered in its specific time slot. If a planned tour is allocated to a carrier, the parcel can be collected at the depot by the carrier. Alternatively, recipients can pick up their parcel in person as soon as the parcel has arrived at the depot but was not yet planned for a tour. Then, the third fragment Fig. 1(c) is executed instead of fragment 2 and 4. The process ends when the parcel is in state *delivered* (Fig. 1(d)), the goal state of this case model.

In fCM, not all of the fragments have to be used for the execution of a certain case to achieve the defined goal of the parcel delivery. Because fCM is both data-driven and event-driven, it is very important to have a valid *Object-Lifecycle (OLC)* which is consistent with the modeled fragments [8]. It describes the state transitions of each data object as shown in Fig. 2.

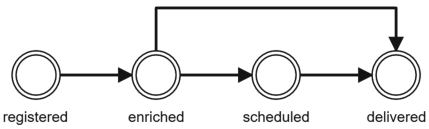


Fig. 2. OLC of data object *Parcel*

Predictable situations at design time, e.g., recipients get parcel in person, can be easily captured by designing fragments to cope with them. Unknown exceptions can be also solved, e.g. for misreadings of the recipient's address on the parcel by the scanner. The fCM engine allows knowledge workers to add fragments at runtime. So far, knowledge workers have to solve exceptions without any structured support in fCM and this is what we want to change. Given that the process fragments in fCM are small workflows, we want to reuse the exception handling patterns by Russell et al. [20] originally designed for workflow systems.

3 Background on Exception Handling

After introducing fCM, we provide in this section the background on exception handling. First, a precise definition of the term exception is given by classifying it into different exception types, each illustrated with an example of the introduced use case. Related work is discussed afterwards and the remaining text introduces the work of Russell et al. [20] as a fundamental basis for this paper.

3.1 Exceptions

Usually, we describe a discrepancy of a business process between the planned flow and the reality as an *exception*. Nevertheless, a discrepancy does not always have to be an exception. Therefore, Lohmeyer [12] distinguishes between in success and failure regarding the goal of a business process. If the goal is achieved although there is a deviation, we talk about a special variation which is already known by the process and pre-defined. However, if a deviation leads to failure, Lohmeyer talks about a real exception. It is a deviation which is unknown by the process.

The difference between *known* and *unknown* exceptions traces back to Luo et al. [13]: A deviation is unknown if it cannot be resolved with the rules of a

Table 1. Exception types according to [20]

Type	Explanation	Example from last mile delivery
Activity failure	Activity is not able to continue its execution [20]	Due to an ambiguous address, the activity *Plan Tour* (Fig. 1(b)) fails because the system cannot include it into the tour
Resource allocation		
a) Non-availability	Non-availability of resources causes non-execution of corresponding activity	No carrier can be found for a specific zip code In this context the carrier is the resource to which a parcel is allocated
b) Depleted capacity	Due to exhausted capacity of the resource during execution, the activity cannot be completed successfully	Parcel volume is bigger than space of transport vehicle, e.g. a cargo bike. Fragment 4 (Fig. 1(d)) can not be executed anymore
c) Wrong allocation	Resource is identified as incorrect after successful assignment	Scanner misreads the address. This causes wrong linking of parcel to recipient
External Trigger	Required information to start a fragment is missing	Database is not reachable. The activity *Plan Tour* does not start because of missing information (Fig. 1(b))

system that have been defined in advance. This means there is no alternative path in the process model. Moreover, an unknown and therefore unexpected exception cannot be handled according to [13].

Russell et al. [20] specify, additional to the distinction into known and unknown exceptions, different exception types. This paper is based on this grouping and works with exceptions as a clearly identifiable event which occurs at *run-time* of a business process. Three of those five types are exemplified in Table 1.

3.2 Related Work

Many authors have addressed the need of flexibility in business processes to enable reliability and support of static and dynamic changes [1,2,20]. Whereas adaptability is a key factor in the context of exception handling, it is also still a major challenge in Case Management [7]. Pattern like Rollback which are mentioned by [20] often address just an instance of a case and this does not suffice for all types of exceptions. Needed changes concerning the structure of a whole case are not covered [21].

Kurz et al. [11] differentiates between three types of exceptions: (1) routine, (2) minor, and (3) major exceptions. Whereas, routine ones can be handled with standard BPM techniques, such as boundary events in BPMN, the handling minor (predictable) and major (not predictable) needs flexibility to handle them at run-time. For handling the minor exceptions, Kurz et al. [11] propose a template-based strategy with best practices and guidelines to handle them, but unpredictable are handled outside of the case management system. Similarly, Fahland and Woith [5] proposes a flexible process execution system based on small Petrinets fragments similar to fCM – called Oclets –, which 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 [4]. A systematization is not given.

For predictable exceptions for which a handling can be defined in advance, there are lots of strategies of handling these. All of them are based on a systematization "in form of exception handling patterns" [15]. We use a similar concept in this paper and show how to adapt known workflow-based handling strategies for handling unpredictable exceptions in a fCM applications.

3.3 Pattern for Exception Handling in Workflow Systems

This section explains the pattern of Russell et al. [20] on which this paper is based on. After introducing the life cycle of an activity, an example of the pattern is used to explain the general structure of the exception handling strategies.

Business process activities can adopt different states, from the initial offering until final termination. To capture these states, life cycles as shown in Fig. 3 are useful for a detailed analysis of an activity during process execution. States are illustrated by boxes. Their abbreviations which are used in this paper are shown in brackets. Each activity can change its state via state transitions, which are

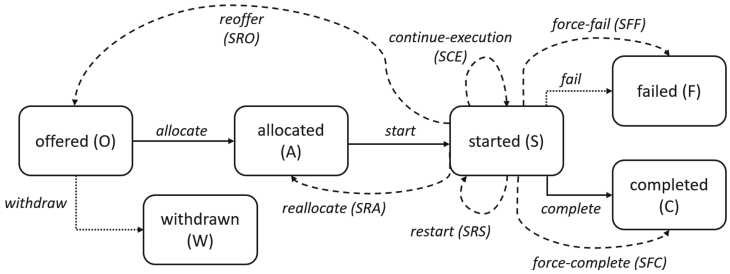


Fig. 3. Simplified life cycle of an activity according to [20]

represented as solid arrows. This is the case, for example, when an exception is triggered in state *started*. As a consequence, the activity changes into its state *failed*. The difference to the dashed arrows, which also represent state transitions, is that the transition is executed automatically whereas the transition using the dashed arrows is deliberately enforced from the outside. A state of an activity can change from *started* to *failed* by two different ways, the natural one or the enforced one. In this paper, we concentrate on activities which are already in state *started*[3]. Therefore Fig. 3 only shows the possible handling strategies for activities which are halted due to an exception.

Like the example pattern *SFF-CWC-COM*, each pattern of Russel et al. [20] is divided into three parts, each marked by an abbreviation which represents one of the three main aspects of exception handling in workflow systems which are defined as follows:

1. Handling of the activity which provoked the exception;
2. Handling of the following activities of the whole case; and
3. Recovery measures which are needed to remove the effects.

The handling of the activity which provoked the exception is summarized by the first part of the pattern. In the example this corresponds to the abbreviation *SFF*. The first three letters of each pattern has to be read separately, i.e. ***S**tarted **F**orce-**F**ail*, because the first letter explains in which state an activity is (cf. Fig. 3) when an unknown exception occurred and an automatic state transition can not be executed. The other two letters describe the status to which the activity is manually transferred. So *SFF* = ***S**tarted **F**orce-**F**ail* means that an activity is halted in state *started* and transferred to state *failed*.

Whereas the first part of a 3-tuple pattern handles one activity, the second part deals with the best strategy for handling on case-level, i.e. all of the following activities after the one which provoked the exception. This is necessary because the occurred exception could affect some or all following activities. The abbreviation by three letters describes one out of three possibilities for it:

1. *Continue Current Case (CWC)* - Every activity following will be executed without any interruptions, the workflow still exists.

[3] An overview of all possible state transitions can be found in [20].

2. *Remove Current Case (RCC)* - Either a selection or all of the activities are deleted.
3. *Remove All Cases (RAC)* - All cases of the same process model are deleted.

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 which are presented in [20] can be used:

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

The introductory example 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 case 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 Exception Handling for fCM

Because an exception is not only linked to a single activity and/or case but also to a fragment, it is no longer sufficient to handle exceptions only on the activity- and case-level. Therefore, a new level of abstraction is needed: the fragment-level.

There are two possible consequences when an exception occurs in a fragment. Either it leads to an inability of a fragment to start or it impedes the successful execution of the fragment. As a direct effect of the latter, needed state transitions, which are trigger for further fragments, cannot be fulfilled.

Since there is no predefined order how fragments are going to be executed, it does not suffice to just look at subsequent activities as we do in the patterns for workflow systems. In the fCM approach, all case fragments have to be considered. Their handling goes hand in hand with the handling of the case (second part of the 3-tuple explained in Sect. 3.3). Nevertheless, an extension of the workflow pattern is needed to cover the fragment-level. This section explains how this can be done by adding a fourth tuple element to the existing pattern of Russell et al. [20]. Whereas the first part discusses the possibilities of recovery measures on fragment-level, the second part focuses on the concrete extension by defining a notation and specific rules. The last section suggests how an integration of the provided exception handling pattern could look like.

4.1 Compensation and Rollback

The main component of a good exception handling is the compensation and rollback of effects. A fCM application offers knowledge workers the opportunity

to actively intervene the process by manipulating data and modeling new fragments. So there are different varieties to handle a fragment in which an exception has occurred. On the one hand, knowledge workers can start fragments manually and on the other hand, they can terminate them on purpose. It is up to the knowledge worker whether the termination condition is evaluated as successful or failed. Moreover, it is very important that the rules given by the model are strictly adhered to. This means, a knowledge worker has the obligation to compensate or withdraw the effects of an exception to ensure a correct execution to achieve the business process goal. There are four options to do so:

1. *Create new fragments*
   Creating new fragments can be helpful if an exception occurs and there is no alternative path in the process model, an extra fragment can be modeled. Nevertheless, it is important to check all of the following fragments to make sure everything can be executed afterwards e.g. through a compliance check explained in [10].
2. *Delete existing fragments*
   If a selection of fragments may no longer be executed, they can be removed by the knowledge worker. Note that a fragment is only removed in the current case so that it is not lost across the whole instance.
3. *Manipulation of states*
   There is a correlation between manipulation and the first two options. For both, adding a new fragment and remove one, state transitions have to be consistent. That is the reason why a continuous verification of the object live cycle is necessary. Therefore, it is important that knowledge workers have the right to put a data object in every state which is defined in the OLC. This means they can also adapt the OLC itself. Moreover, this option is important if you want to set back the execution of a fragment because you have to make sure that the data objects have the right state.
4. *Do nothing*
   This option should be chosen carefully and only if the effects of an exception have no influence on other fragments.

A strict differentiation in compensation and rollback as proposed in traditional exception handling is not possible anymore. Knowledge workers have to react situational. They have to decide for each exception individually if a rollback does make sense or not. So, the success of achieving a business goal lies with the knowledge workers. Due to the complexity of a fCM business process model, knowledge workers have to be more qualified in modeling skills than a ACM-User and has to have more rights than a PCM-User to be able to add new fragments or change states of data objects.

4.2 Notation

Exception handling on fragment-level is directly connected with the handling of the activity which has provoked the exception and the handling on case-level. That is the reason why an extension of the existent pattern for workflow

systems is best suitable for a good exception handling in the context of fCM. These pattern already suggest a strategy on how to handle an activity and the following ones.

Our idea is to modify the given notation of Russell et al. [20] by adding a fourth tuple element to the existing 3-tuple pattern. Amongst handling the activity which triggers the exception, the case-level and the concrete recovery measures, the notation is extended by exception handling on fragment-level. This section explains the extension by defining specific rules because not every extended pattern leads to meaningful exception handling strategies.

Like the first part of the given notation (i.e. the activity handling), the fourth part is also an abbreviation consisting of three letters which have to be read separately. The first letter specifies the further execution of the fragment in which the exception has occurred. Suggested recovery measures are covered by the last two letters. While the first aspect correlates directly with the handling of the activity (first tuple element), the second topic is connected to the recovery component which is the third part of the strategy tuple. This relation to the first and third element of the tuple is the general rule of how to extend an existing workflow pattern to fCM.

Whenever an exception occurs, it is important to decide whether a fragment should be continued (C) or terminated (T) in its execution. This decision relates to the handling of the activity which has provoked the exception. If the activity is forced to state *failed* (e.g. SFF = Started Force-Fail), the fragment has to be terminated manually by knowledge workers. Any further execution would endanger a correct execution path regarding OLCs because any failed activity do not trigger following activities within the same fragment. In contrast to that, there are methods which still allow a continuation of the fragment. For example, SCE (Started Continue-Execution) does not terminate any activity because there is not state transition to *failed* (cf. Fig. 3) as an effect of the exception. Following activities can therefore still be triggered. Table 2 shows this correlation of handling an activity and a fragment for exceptions which has occurred in activity state *started*.

Table 2. Correlation of handling an activity and a fragment

Termination (terminate T)	Continuation (continue C)
SFF (force failing)	SCE (continue execution)
	SRS (restart)
	SRA (reallocate)
	SFC (force completion)
	SRO (reoffer)

There are different ways to implement the presented recovery measures of compensation, rollback and faineance on fragment-level. Each method has an abbreviation by two letters which is used for the second part of the extension:

- NE - Adding a **ne**w fragment
- DE - **De**lete an existing fragment
- MA - **Ma**nipulation of states
- NT - Do **no**thing

This results in eight possible combinations (cf. Table 3). However, the notation of a pattern is always interpreted in the overall context. That means, a free combination of the presented possibilities with the patterns of Russell et al. [20] is not wise, but it is dedicated to some rules. Here comes the direct correlation of the handling of an activity and the recovery measures into play.

Table 3. Possibilities of handling exceptions on fragment-level

State of fragment	Recovery measures			
	Add	Remove	Manipulate	Do nothing
Terminate (T)	TNE	TDE	TMA	TNT
Continue (C)	CNE	CDE	CMA	CNT

For recovery measures, Fig. 4 gives an overview of the different rules. It is assumed that the compensation (COM) presented by Russell et al. [20] implies a manipulation of object states. This means that a pattern that relies on compensation as a recovery measure cannot contain state manipulation (MA) as the fourth tuple element. For the option of a rollback (RBK), i.e. the return of the process for a new execution, it is assumed that the object state must be manipulated too. If knowledge workers re-execute a fragment with different parameters, they have to ensure that the data conditions are consistent with the process model. For this reason, a rollback only specifies whether a change was made to the process model or whether the fragment is executed again without an active intervention by knowledge workers. If the exception does not have any serious consequences, in most cases no recovery action is needed (NIL). Knowledge workers can now consider whether to keep the process model as it is or whether it makes sense to add a fragment (NE) or remove an existing one (DE) for future executions of the process.

In combination with the rules specified for the first letter of the extension, each pattern of Russell et al. [20] can be extended by looking at the first and third element of the tuple. For example, the pattern *SFF-CWC-COM* can be extended to *SFF-CWC-COM-TNE* because the considered activity was forced to state *failed* due to SFF = Started Force-Fail. Therefore, the fragment has to be terminated (first letter of fourth element is T). As a compensation method (third element COM) the strategy pattern suggests the creation of a new fragment (last two letters of fourth element are NE).

There is more than one possibility to extend a pattern for workflow systems to make it suitable for fCM. The pattern *SFF-CWC-COM* from above has another extended version which is *SFF-CWC-COM-TDE* because a deletion of existing

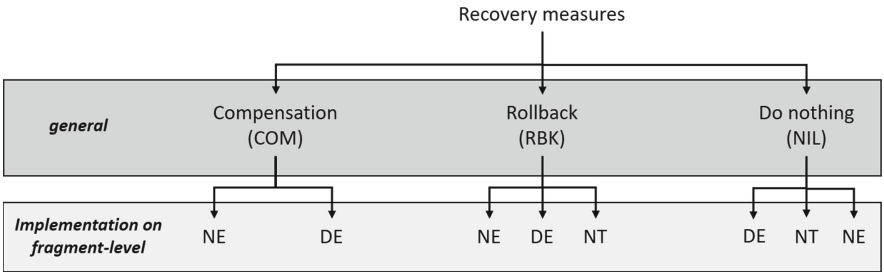


Fig. 4. Possibilities for implementing recovery measures

Table 4. Extension of exception patterns for workflow systems

Workflow system	fCM application
SFF-CWC-COM	SFF-CWC-COM-TNE
	SFF-CWC-COM-TDE
SRS-CWC-RBK	SRS-CWC-RBK-CNE
	SRS-CWC-RBK-CDE
	SRS-CWC-RBK-CNT
SRA-CWC-NIL	SRA-CWC-NIL-CNE
	SRA-CWC-NIL-CDE
	SRA-CWC-NIL-CNT
SFF-RCC-COM	SFF-RCC-COM-TNE
	SFF-RCC-COM-TDE
SCE-CWC-NIL	SCE-CWC-NIL-CNE
	SCE-CWC-NIL-CDE
	SCE-CWC-NIL-CNT

fragments can also be considered as a compensation method. Table 4 illustrates this aspect for the workflow pattern where the activity was in state *started* when an exception occurred.

4.3 Integration into a fCM Application

Knowledge workers often have to design own fragments or modify existing ones to implement the presented pattern. This has a huge implications on their process modeling knowledge, but also on the verification of fCM at run-time. They have to be adept in the area of the certain process to choose the best suitable pattern for handling an exception if there are more than one possible strategies. These enormously high demands on knowledge workers require a tool support to enable the feasibility of the presented concept of exception handling in fCM.

For implementation, the listing of possible pattern has to be automated. This can be done by analysing in which activity state an exception occurs and what exception type it relates to. The platform, e.g. an UI, has to have (a) a *notification function* to alert knowledge workers immediately whenever an exception occurs, (b) an *overview of all possible pattern* with their costs and consequences, (c) a *modeling space* to design new fragments or modify existing ones, (d) a *simulation* option to verify the changes of the process model and (e) a *compiling* option to integrate the changes to the process model. In the next chapter, we are going to show how a possible implementation, following our criteria, could look like on the example of an use case.

5 Application to Use Case

The concepts for exception handling in fCM described in the previous section are exemplified in this section with the help of the last mile delivery fCM model introduced in Sect. 2 as well as the exceptions described in Sect. 3.1.

First of all, the handling for the failure of an activity is discussed. This may be the case when a recipient's address does not exist. The algorithm for optimizing the delivery tour cannot be executed because it cannot identify the stop on a map. The activity *Plan tour* (Fig. 1(b)) is set to state *failed* and while the case continues in execution the fragment which includes the failed activity is terminated manually. Because there is no handling on the part of the process model, knowledge workers would be notified. The mock-up in Fig. 5 shows how this could look like. The interface provides an overview of what exception occurred and how it can be handled. Here, it is important that the case itself continues in executing to ensure a successful delivery of all other parcels. The parcel with the non existent address has to be scanned again or transferred to a human who can then correct the address in the system. This strategy conforms to the pattern *SFF-CWC-RBK-TNT* and *SFF-CWC-COM-TNE*. Both ensure the continuation of the case, but while the first one suggests a rollback by terminating the fragment and restart it, the second one recommends compensation by creating a new fragment e.g. for manual input of the address.

Secondly, an exception of type *Resource not available* is discussed: the case that no carrier can be found to deliver a given parcel. The third fragment can not terminate successfully because the allocation of the planned tour failed. In this case, the concept explained above suggests the pattern *SFF-CWC-COM-TNE*. The case has to continue (CWC) because the parcel has to be delivered to achieve the process goal. So, the best strategy to handle the situation is to compensate the effects (COM), e.g. through modeling new fragments which allow parking of parcels until a carrier is available. As an alternative, a new carrier could be employed. Although this handling would work, the strategy is expensive because ensuring a delivery within a two hour time slot can cost very much.

If the database is not reachable temporarily and does not receive any requests, many fragments will not be triggered. For handling, the patterns *SRS-CWC-RBK-CNT* and *SRS-CWC-RBK-CNE* are both possible. It is very useful, to

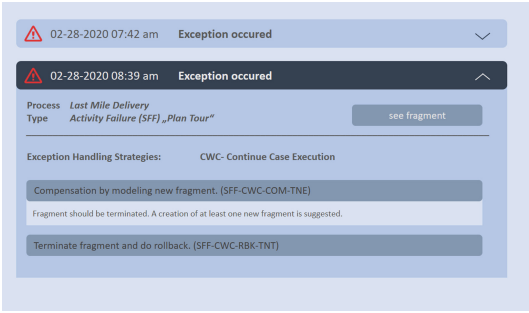


Fig. 5. Mock-up of UI showing occurrence of activity failure

restart the activity once it failed and moreover, there is no reason to stop the case. A rollback is necessary to ensure the correct state of the needed data objects, but on fragment-level a knowledge worker can either do nothing or add a new fragment notifying an engineer to fix the problem.

6 Conclusion

The topic of exception handling in context of fragment-based Case Management (fCM) during run-time is complex and requires a deep understanding of both, the technical context and advanced skills in process modeling. This paper explains a concept of how the strategies of Russell et al. [20] can be used for exception handling in fCM by introducing the fragment-level. Due to an extension of the notation by a fourth element, the handling of the fragment in which the exception occurred can be defined as well as the handling of subsequent fragments can be defined within one pattern. Each handling method for fCM can be mapped as a quadruple. Special rules avoid duplication of semantics of the patterns [20] and ensure direct integration into them. For each original exception handling strategy, there are at least two extensions that can be used in a fCM application. That is why exception handling in fCM is more powerful than in a control-flow based system.

This paper provides a foundation that knowledge workers and developers of an fCM application can use and build on to ensure efficient exception handling. In the future, we want evaluate the usability of our approach by checking the given requirements (see Sect. 4.3) with knowledge workers, as this is essential to guarantee the effectiveness of our future implementation.

Acknowledgement. The research leading to these results has been partly funded by the BMWi under grant agreement 01MD18012C, Project SMile. http://smile-project.de.

References

1. 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 2001, pp. 42–51. Association for Computing Machinery, New York (2001)
2. Agostini, A., De Michelis, G.: Improving flexibility of workflow management systems. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 218–234. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45594-9_14
3. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-662-56509-4
4. Fahland, D.: From scenarios to components. Ph.D. thesis, Humboldt University of Berlin (2010)
5. Fahland, D., Woith, H.: Towards process models for disaster response. In: Ardagna, D., Mecella, M., Yang, J. (eds.) BPM 2008. LNBIP, vol. 17, pp. 254–265. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00328-8_25
6. Gonzalez-Lopez, F., Pufahl, L.: A landscape for case models. In: Reinhartz-Berger, I., Zdravkovic, J., Gulden, J., Schmidt, R. (eds.) BPMDS/EMMSAD -2019. LNBIP, vol. 352, pp. 87–102. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-20618-5_6
7. 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, September 2014
8. Hewelt, M., Pufahl, L., Mandal, S., Wolff, F., Weske, M.: Toward a methodology for case modeling. Softw. Syst. Model. **2019**, 1–27 (2019). https://doi.org/10.1007/s10270-019-00766-5
9. Hewelt, M., Weske, M.: A hybrid approach for flexible case modeling and execution. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNBIP, vol. 260, pp. 38–54. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45468-9_3
10. Holfter, A., Haarmann, S., Pufahl, L., Weske, M.: Checking compliance in data-driven case management. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) BPM 2019. LNBIP, vol. 362, pp. 400–411. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37453-2_33
11. 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 2013. CCIS, vol. 360, pp. 123–149. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36754-0_8
12. Lohmeyer, B.: Writing Use Cases: Exception or Alternate Flow? Lohmeyer Business UX (2013). https://www.lohmy.de/2013/03/06/writing-use-cases-exception-or-alternate-flow/. Accessed 28 Feb 2020
13. Luo, Z., Sheth, A., Kochut, K., Miller, J.: Exception handling in workflow systems. Appl. Intell. **13**, 125–147 (2000). https://doi.org/10.1023/A:1008388412284
14. de Man, H.: Case management: a review of modeling approaches. Technical report, BPTrends (2009)
15. Marin, M.A., Hauder, M., Matthes, F.: Case management: an evaluation of existing approaches for knowledge-intensive processes. In: Reichert, M., Reijers, H.A. (eds.) BPM 2015. LNBIP, vol. 256, pp. 5–16. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42887-1_1

16. Motahari-Nezhad, H.R., Swenson, K.D.: Adaptive case management: overview and research challenges. In: 2013 IEEE 15th Conference on Business Informatics (CBI), pp. 264–269. IEEE (2013)
17. OMG: Notation BPMN Version 2.0. OMG Specification, Object Management Group, pp. 22–31 (2011)
18. 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: Business Information Systems 2020. Springer, Cham (to be published)
19. Ranieri, L., Digiesi, S., Silvestri, B., Roccotelli, M.: A review of last mile logistics innovations in an externalities cost reduction vision. Sustainability **10**(3), 782 (2018)
20. Russell, N., van der Aast, W., ter Hofstede, A.: Exception handling patterns in process-aware information systems. Technical report. Queensland University of Technology/Eindhoven University of Technology(2006)
21. 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)