Implementation Framework for Production Case Management: Modeling and Execution

Andreas Meyer, Nico Herzberg, and Mathias Weske
Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
14482 Potsdam, Germany
{Andreas.Meyer,Nico.Herzberg,Mathias.Weske}@hpi.uni-potsdam.de

Frank Puhlmann
Methodology & Solution Architecture
Bosch Software Innovations GmbH
10785 Berlin, Germany
Frank.Puhlmann@bosch-si.com

***Abstract*—Nowadays, business process modeling and system-supported executions have become a commodity in many companies. Most systems, however, focus on modeling and execution of static, pre-defined processes with standards like the Business Process Model and Notation (BPMN). While these static process executions are applicable to a number of traditional processes like purchase orderings or back orderings, they fail at representing variant-rich, flexible processes. One solution for supporting flexible processes is Adaptive Case Management (ACM), where a case manager creates an individual execution path for each process instance, such as a doctor defining a clinical pathway for a specific patient. We found out, however, that both approaches are too strict, either supporting static process definitions with only a limited set of pre-defined flexibility or allowing maximum flexibility but requiring a highly skilled knowledge worker. To overcome this problem, we propose an implementation framework for Production Case Management (PCM) that combines concepts from traditional process management and adaptive case management. PCM combines the modeling of small, static process fragments with the execution flexibility of ACM.**

Keywords*—Process Flexibility, Process Modeling, Process Execution, Methodology

I. Introduction

Production Case Management (PCM) is a paradigm to organize and structure the daily work within organizations that "is used when there is a certain amount of unpredictability in the work, but still a large enough volume to make identifying and codifying regular patterns" [1]. In contrast, traditional business process models – like those expressed in BPMN [2] – focus on mostly static process models [3]. If more execution flexibility is required, Adaptive Case Management (ACM) is a well-investigated solution [4], [5]. Comparing ACM and PCM, the latter defines execution alternatives at design-time from which are chosen during run-time while ACM proposes next process steps based on historical information and given guidelines and constraints but the process participant is free to choose any task for execution.

While there exists a lot of related work on extending static process models to specify variants at design-time, e.g., [6]–[10], or to decide the execution at run-time, e.g., [11], [12], none of them describes a generic implementation framework for Production Case Management.

In this paper, we discuss an implementation framework for Production Case Management by introducing the notion of process components (fragments) derived out of notations like BPMN that can be added, changed, or removed at run-time to describe the run-time behavior of the process. Instead of requiring a skilled knowledge worker to decide which fragment should be executed, we rely on proven concepts from data- and state modeling [13]–[16]. Furthermore, a crucial point in introducing more flexibility to business users, especially at the modeling side, is an easily understood modeling paradigm or methodology.

The remainder of this paper is structured as follows. We start with a real-world example in Section II that we will use in subsequent sections to illustrate our framework. Section III introduces the core terminology followed by the methodology to model a process model as a set of process components in Section IV and the corresponding execution semantics in Section V. The application of the introduced concepts in a customer project is discussed in Section VI. Finally, we discuss related work in Section VII and Section VIII concludes the paper.

II. Example

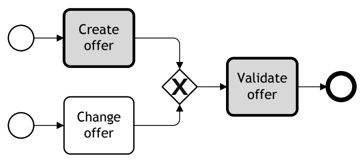
In this section, we introduce an example inspired from collaboration with a large tourism corporation. The example is centered on a travel agency that receives a request for a quote. The request is then processed and the corresponding offer is finally sent to the customer.

As you can imagine, multiple ways of processing individual customer requests exist which cannot be easily anticipated and captured in one single process model. On the other hand, the "usual" processes occur often and in many cases, they consist of the same activities. Hence, the process has too much variability which is hard to capture in traditional, static process models (e.g., BPMN). At the same time, it is not that individual such that Adaptive Case Management is a good representation, since in most cases we have to deal with the same behavior in different orders.

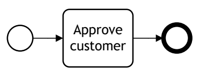
To this end, we propose to model and to execute the processes of the travel agency using the Production Case Management paradigm. We limit ourselves to a subset of the complete process with a focus on introducing all concepts.

A. Simple Process Components of the Travel Agency

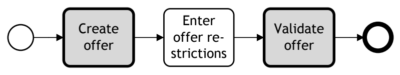
To get started, we model the different alternatives and single steps in various process components, each being a single



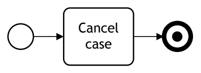
(a) Process component p1 that takes care that each created or changed offer is validated.



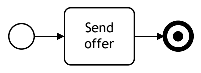
(b) Process component p2 that allows approval of customers.



(c) Process component p3 that allows capturing restrictions for new offers.



(d) Final process component p4 that sends the offer and ends the process.



(e) Process component p5 that allows to cancel the offer at any time.

Fig. 1. Example process components of a travel agency.

process diagram that informally captures a specific procedure of a process. Fig. 1 shows five process components representing a reduced – but for the scope of this paper sufficient – view on the scenario from the agency's point of view.

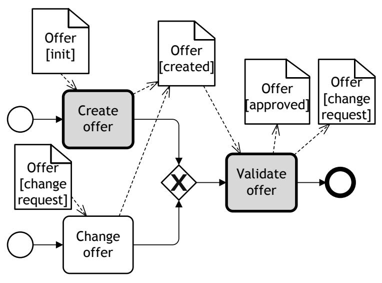
Standard Procedures. Fig. 1a presents the standard procedure for creating respectively changing an offer, depending on whether iteration is required or a new request is placed before the offer gets validated. The following mandatory validation either approves the offer or requests an iteration prior approval.

Variant Procedures. In some cases, the creation of an offer requires an additional step – the insertion of offer restrictions – before the validation can take place. To retain small and easy understandable process models, we model this option in a separate component (see Fig. 1c). Please note that the shaded activities with the same labels (e.g., *Create offer*) represent exactly the same activities across all process components (modeled via call activities in BPMN). Execution-wise, after finishing *Create offer*, either the corresponding path in Fig. 1a or the path in Fig. 1c is taken. The actual decision will be computed at run-time.

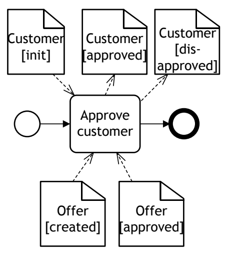
Optional Procedures. If the customer, requesting the quote, is new to the travel agency, she needs to be approved before the approved offer can be sent. This additional step can take place at different times during run-time, modeled in Fig. 1b.

Final Procedures. Furthermore, some process components are able to end (terminate) the process. We make use of the BPMN termination end event to represent this case. The example process of the travel agency ends after the offer has been sent, shown in Fig. 1d.

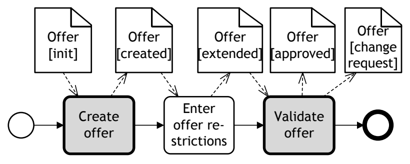
Global Procedures. Finally, we sometimes find activities that should be always possible. In the case of the travel agency, this is the cancellation of an offer if it is no longer required, which can happen at any time. Nevertheless, the cancellation must be documented and the process terminated. We show a corresponding component in Fig. 1e.



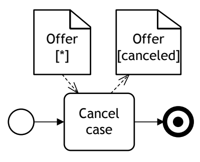
(a) Component p1.



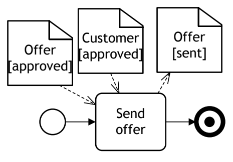
(b) Component p2.



(c) Component p3.



(d) Component p4.



(e) Component p5.

Fig. 2. Refined process components representing the request for quote business process of a travel agency.

B. Refined Process Components of the Travel Agency

The process components shown in the previous subsection represent an informal overview of the different procedures required for an offer process in a travel agency. The corresponding synchronization of the different components might only be assumed by the context or be written underneath. While this is usually a good start for capturing the different procedures of a business process with the domain experts of a customer, we need to add more refined constraints to the process components.

The refined constraints for synchronization requirements are represented by adding BPMN data objects with state labels as pre- and postconditions, shown in Fig. 2. The process components act upon multiple data objects with each activity being able to read (input) or write (output) them. A set of data objects read from an activity is the precondition which must hold to allow execution. A set of data objects written from an activity are the postconditions expected to hold after activity execution, i.e., the activity has to ensure that the data objects exist in the requested states. An activity having multiple data objects with the same label as input (or output) is required to only read (or write) one of them as they are exclusive and present multiple options for execution. Data objects with different labels are conjunctive.

For illustration purposes, we introduce two different data objects for the travel agency scenario. The first is the *Offer* that is created and passed between the different activities and components as transient data. The second one is a *Customer* data object that relates to master data stored in a customer

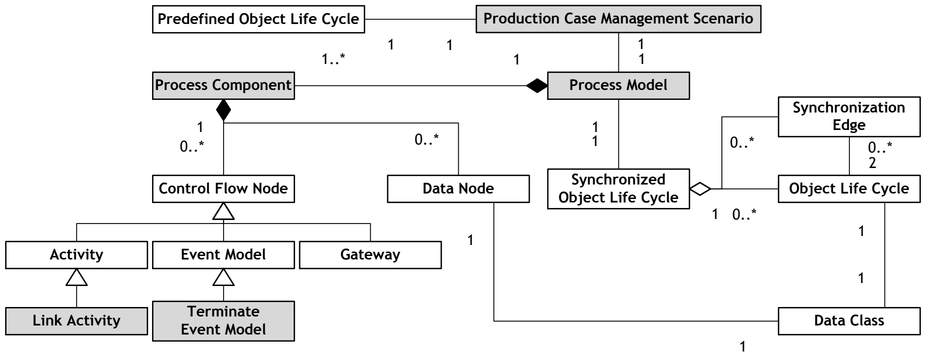


Fig. 3. Terminology overview of key modeling concepts as UML class diagram with the gray shaded classes representing the fundamental paradigm changes.

database.

C. Sample Execution of the Travel Agency's process

Concluding the example section, we would like to discuss how an agent of a travel agency might work with the implemented process components.

Starting the process. An agent has the option to start all process components that have a data object in the correct state attached to it; if the object is only used within one process, then this state is *init*. In case of the *Customer*, the correct state might also be *approved* as she might have been approved in some earlier execution. Regarding our example, if the customer is new, these are components p1 and p3, both allowing to execute the same activity *Create offer*.

Selecting different procedures. After *Create offer* has been executed, either *Validate offer* from component p1 or *Enter offer restrictions* from component p3 can be selected for execution. Additionally, the agent can decide to cancel the case with selecting the activity *Cancel case* from component p5. If she decides to execute the activity *Validate offer*, one of the two postconditions might set the state of the *Offer* to *change request*, which in turn enables the execution of *Change offer*.

Approving a customer. During the time an *Offer* is in state *created*, the agent has the option of executing the activity *Approve customer* from process component p2, if both preconditions are true (an *Offer* in state *created* AND a *Customer* in *init*).

Sending the offer. Finally, when the *Offer* as well as the *Customer* data objects are in the states *approved*, the agent can send the offer to the customer via selecting the activity *Send offer* for execution.

While this small example only discusses a small fraction of the possible states a real business process can go through, the introduced concepts scale to larger processes too, as discussed in Section VI. More details about the execution semantics follow in Section V. Next, we continue with the formal terminology and the general approach.

III. Terminology and General Approach

To introduce an implementation framework for Production Case Management, we re-interpret three fundamental paradigms used within the business process management community:

(P1) Process components instead of diagrams. A process instance is not correlated to a single process diagram but to a set of process components collectively specifying the corresponding business process model.

(P2) No visual end–to–end paths. The execution path taken to achieve the business process goal is not visual but emerges step-by-step during the actual execution.

(P3) Stepwise goal refinement. Referring to various process components representing the process model and the execution path which is gradually revealed by taking several process components into account, a large variety of paths through the process might be taken upon process instantiation, while towards the end the number of paths becomes limited due to the decisions taken.

Fig. 3 summarizes the key modeling concepts as UML class diagram [17] and relates them to each other. Gray shaded classes represent the aforementioned fundamental paradigm changes. Additionally, we will explain further concepts that are required for completeness reasons and which closely relate to one of the key concepts. Each *production case management scenario* is a business process represented by one *process model* that consists of multiple *process components*, each describing some part of the business process. A process component contains multiple *control flow nodes*, which specify the partial ordering of *activities*, one specific control flow node. This relation is named *control flow*. Further control flow nodes are *event models* and *gateways* where the latter is typed for representing interleaving semantics (AND) and exclusive choice (XOR) while the former may be of type plain start or plain end event model. These modeling constructs comprise the subset of BPMN [2] we support within the framework. Further, we introduce a specific activity named *link activity* which represents a set of activities occurring in different components but performing the same work. We also introduce a specific event model named *terminate event model* which indicates a final process state from the control flow point of view. Secondly, a component contains *data nodes* that specify the pre- and postconditions of activities with respect to their enablement and termination. Each data node references one *data class* that describes the structure of data objects in terms of attributes and possible states as enumeration. *Data objects* are instances of data classes and are represented by data nodes. A *data state* of an object describes a specific situation of interest that is described by a unique set of attributes with each one holding a value. Thus, the existence and non-existence of values identifies the current state of a data object during process execution.

A data node may be read, written, or both (modified) by an activity. We refer to the relation comprising all these associations between data nodes and activities as *data flow*. Thereby, each read specifies a precondition meaning that an activity only gets enabled if all preconditions are met, while a write specifies an expected result of the activity as postcondition, i.e., the data object in a specific state. Conditions including various data nodes combine the ones referencing the same data class by disjunctions and the ones referencing different data classes by conjunctions. Considering Fig. 2a, activity *Validate offer* requests object *Offer* in state *created* for enablement and expects the same object being either in state *approved* or in state *change request* after activity termination. Each data class references one *object life cycle (OLC)*, a state transition diagram that describes the behavior of data objects, i.e., the partial ordering of states of the corresponding class. Within a process model, multiple data classes with some inter-dependencies may be utilized. For each such class, one OLC exists. The inter-dependencies are represented by one *synchronized object life cycle* [15], which uses directed *synchronization edges* consisting of a source and a target to connect two states of different OLCs and to show the dependency direction. Each process model refers to one synchronized OLC.

Traditionally, the execution of a business process is represented by a process instance that is controlled by exactly one process model. In contrast, in our approach, a process instance is collectively controlled by a combination of process components, which are collected while the process instance runs. For the process model, a termination condition needs to be specified to indicate the achievement of the business process goal, i.e., the termination of one process instance. This condition subsumes control flow and data flow requirements for which at least one has to be defined. On the one hand, a termination condition refers to a set of data objects in specific data states, logically combined via conjunctions and disjunctions. On the other hand, *termination event models* characterize situations where the process model may terminate from the control flow point of view. If these conditions hold true, the process model is terminated.

Finally, a *predefined object life cycle* exists which defines the superset of all data states of all data classes including their allowed state transitions which are probably involved in the production case management scenario and the synchronization dependencies between these classes.

IV. Methodology

The framework consists of five steps as shown in Fig. 4: (i) business process creation by modeling several process components, (ii) transformation of data nodes used in the components into OLCs, (iii) extending the OLCs with additional states, state transitions, and synchronization edges, (iv) transforming the extensions made to the OLCs into separate components, and (v) executing the business process based on *all* process components, the created ones as well as the derived ones. Steps (i) to (iv) can be repeated multiple times to allow iterative process component creation.

We assume that object life cycles may only be extended as mentioned in step (iii). Deletions are not allowed. Adding

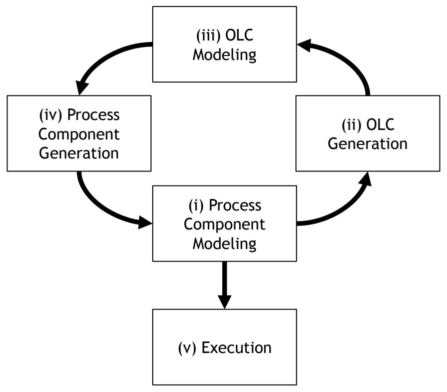

(iii) OLC Modeling
(iv) Process Component Generation
(ii) OLC Generation
(i) Process Component Modeling
(v) Execution

Fig. 4. Methodology overview.

synchronizations needs special handling as these may contradict to existing process components. In this paper, we restrict the addition of synchronizations to those that align with the current business process description. The predefined object life cycle shall be used as guidance for the specification of data manipulations (reads and writes) in process components. All state transitions of the data object's life cycle need to be represented by the activities used for defining the process components of the business process. These process components need to be sound from the control flow point of view, i.e., each process component instance that starts in the initial state may eventually reach its final state if no other component interferes. For the isolated execution of each component, reaching the final state implies that no tokens are left in the net, and each activity of the process component can contribute to the process component instance [3]. Applying the concept of weak conformance [18] to each component with respect to the OLCs that correspond to the utilized data classes enables the identification of inconsistencies if, for instance, the component requires a data state transition for some object which is not covered by the predefined OLC.

A. Process Component Modeling

We propose to first model the "happy paths" of the business process to build a basic understanding and to set the scope. A "happy path" describes the most likely process behavior ignoring special cases, exception handling, and complicated decision taking. Multiple process participants are involved in process execution but they may have different views on the overall process. Each participant models her own "happy path", where each is regarded during run-time (cf. Section V). Thereby, only control flow is modeled in the first place before data objects are associated to these activities as pre- and postconditions. To ease process understanding, the activities executed in such a "happy path" may not be modeled in a single process diagram but in a set of process components executed in a specific order. Further, such decomposition into components allows their reuse to describe further paths through the process model in later iterations. Consider *p1* and *p5* in Fig. 2 representing the "happy path", both could be modeled in a single process diagram. As activity *Send offer* needs to be reused at the end of *p3*, outsourcing it to *p5* reduces the number of modeling constructs.

In subsequent iterations, additional paths through the process model are added in single process components. Adding optional activities shall be done by copying the corresponding "happy path" process component and changing this by adding the respecting activities (cf. activity *Enter offer restrictions* in Fig. 2c). In this case, some activities also get reused. We support this by linked activities (gray shaded activities in the process components), which ensure that multiple appearances of the same activity are always treated the same way. From a technical point of view, it is not necessary to label linked activities similarly in different components. However, for stakeholders, it is easier to get the connection between linked activities if they are labeled equally. Alternative and parallel paths do not have to be modeled in separate process components by using the corresponding gateways. This also holds for loop behavior in the business process.

An activity that is executable at multiple occasions should be modeled in a single process component by adding the alternative preconditions. If execution is allowed at all times, the corresponding data object in this component gets assigned an asterisk * as data state acting as a placeholder for each state in the object's life cycle. Referring to the process component in Fig. 2d, the case can always be canceled independently from the current state of object *Offer*. Furthermore, this capability is often used for exception handling.

B. Object Life Cycle Generation and Modeling

Based on those process components, the relations between the data nodes are analyzed for the process model by generating single OLCs for each data class connected by synchronization edges. For generation, we use the approach presented in [15]. This is necessary, as data nodes depend on each other, e.g., the offer can only be sent to a customer if she is approved. Applying the OLC generation step to the process components shown in Fig. 2 will produce the synchronized OLC shown in Fig. 5 excluding the three dashed edges. In Fig. 5, a solid edge represents a transition between two states of one OLC, a dotted edge represents a dependency between two states of different OLCs, and a dashed edge represents a newly added transition or inter-dependency. This depends on whether the edge connects two states of one OLC or two states between different OLCs. The data objects used in this process model are *Customer* (upper part) and *Offer* (lower part). This OLC describes the state transitions (solid edges) and dependencies (dotted edges) modeled in process components *p1* to *p5*. Generation of the synchronized OLC requires a satisfied preceding weak

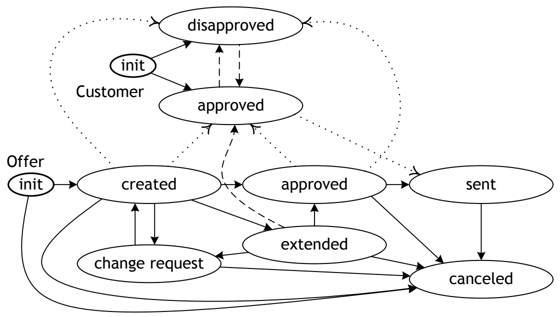


Fig. 5. Object life cycles generated from the process components in Fig. 2.

conformance check to ensure that only allowed state transitions are used in the components.

The process components can be transformed into the synchronized object life cycle at all times, allowing a view on the data manipulations specified. We recommend utilizing this view to (i) check whether changes that are supposed to be done to data objects comply to the expectations and to (ii) add state transitions determining object manipulations not yet modeled. In Fig. 5, we added, for instance, transitions to change the state of a *Customer* from *approved* to *disapproved* and vice versa (dashed edges).

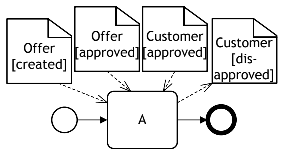
Maybe not all necessary state transitions are represented in the OLC yet, because these dependencies are not modeled in the process components. Thus, it is possible to add further states, state transitions, and synchronization edges to the OLC; this is expressed with dashed lines in Fig. 5. All of these changes must be additions that do not contradict with the specifications in the single process components. To ensure these consistencies, several verifications could be applied that are not subject to this paper. Satisfaction of the concept of weak conformance to the utilized OLCs must be ensured again after additions.

As shown in Fig. 5 and according to the process components described in Section II, a *Customer* may be in state *init* when she is new to the travel agency or in states *approved* or *disapproved* after customer approval in component p2. Customer re-approval is not modeled in a process component but is part of the business process. Therefore, the dashed transitions between states *disapproved* and *approved* are added to the OLC. Re-approval might also result in the same state as before but for complexity reasons, we abstain from this option in this paper. The first state of object *Offer* is *init* as well. Afterwards, the offer gets *created* before it is approved either directly or via state *extended* which refers to additional offer restrictions (cf. Fig. 2c). After approval, the offer is sent to the customer. Additionally, at all times, an offer might also be *canceled*. As both data objects interact with each other, some constraints apply to selected state transitions. A customer can only be approved or disapproved respectively if the corresponding offer is in state *created* or in state *approved*. Due to the added synchronization edge, a customer may also be approved if the offer is in state *extended*. Finally, an offer can only be sent to an approved customer.

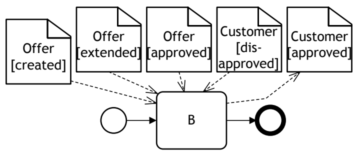
To start with modeling the OLCs and their inter-dependencies is another option in this iterative approach, especially if the process flow follows from objects instead of activities. However, generating the process components will result in a very fine-grained set of components, each having only one activity that may need to be consolidated manually to get meaningful ones. Nevertheless, the consolidation is optional and the approach presented in this paper also works with such minimal process components.

C. Process Component Generation

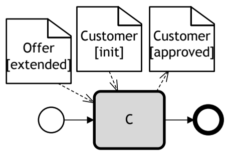
To incorporate the changes made to the synchronized OLC, each added edge is transformed into a separate process component, again using the transformation approach from [15] applied to the single edges instead of adapting the existing components based on the additions. After transforming the adapted synchronized object life cycle back to the set of



(a) Component p6 that allows to disapprove a customer.



(b) Component p7 that allows to re-approve a customer.



(c) Component p8 that allows customer approvals.

Fig. 6. Generated process components.

process components, the newly created components are assigned meaningful labels instead of placeholders as in Fig. 6.

For both of the dashed transitions between states *disapproved* and *approved* of object *Customer*, process components *p6* and *p7* shown in Fig. 6 are generated with p6 setting the state to *disapproved* and p7 setting the state to *approved*. Fig. 6c shows the process component resulting from the added synchronization edge between states *extended* and *approved* of objects *Offer* and *Customer* respectively. The labeling of the corresponding activities is arbitrary and can be changed by the process designer during the next iteration of process component modeling (step 1). Activity *C* of component p8 encapsulates the same operation as activity *Approve customer* from component p2 in Fig. 2b. Therefore, the process designer links both activities via a *link activity*.

As multiple participants contribute to process modeling, organization-internal modeling guidelines are required to ensure consistent granularity of the process components. In this paper, we abstract from such guidelines and assume they exist and are followed.

V. Execution Semantics

The modeled process components are managed in a run-time repository, where they get interpreted during run-time. This enables changing, adding, and removing process components during run-time, which allows adaptable task execution. The concept of optional tasks imposed during process execution, can be handled by adding a corresponding process component to the run-time repository even during process instance execution and executing following this path. Before the execution of a business process may start, OLC conformance between the process components and the predefined object life cycle must be ensured to avoid deadlocks [18]. If all components satisfy this property, the process model also does.

A. Process Execution Rule Set

The operational semantics to execute business processes consisting of multiple process components as introduced above utilizes token play that follows five rules.

(R1) Instantiation. Instantiation of a process model instantiates all comprised process components, i.e., all start events get a token. This means, there does not exist an explicit ordering of process components but an ordering of activities imposed by data flow.

(R2) Activity enablement. An activity gets enabled if control flow as well as data conditions hold. Thereby, we decouple both conditions such that control flow is based on the token flow through the process components and data objects are required to exist in the expected states (cf., for instance, [16]). To enable link activities, control flow and data conditions are required to hold for any activity being element of that link activity.

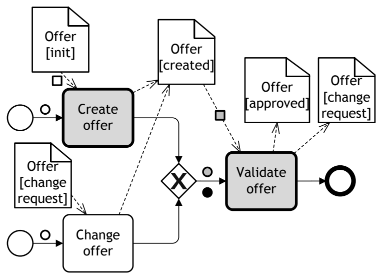
(R3) Activity execution. Execution of an activity pushes forward the control flow by enabling the control flow edge originating from this activity and therefore enabling the succeeding activity from the control flow perspective; control flow execution follows a token game as described in [18]. XOR gateways limit the number of enabled activities with respect to decisions taken. The options, i.e., conditions, need to be non-overlapping and complete such that always exactly one path can be chosen. In case of link activities, all activities that are part of that link activity and have been enabled from control flow and data conditions get pushed forward by putting a token on the control flow edges originating from that activity. Additionally, each further activity belonging to the link activity gets pushed forward if it is control-flow-enabled and if it shares the data output conditions with the activity being actually executed.

(R4) Component termination. Termination of a single process component leads to a re-initialization of that component as our approach allows to reuse process components multiple times throughout business process execution.

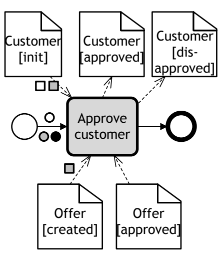
(R5) Process termination. A business process is terminated if the termination condition is matched. Termination of the business process removes all tokens from the process components.

These rules are applied in parallel to all process components potentially resulting in multiple activities being enabled at the same point in time. Following the concepts of production case management, the process participant gets presented all these activities – for instance, as *task list* such that the user can decide which one to execute next; link activities are presented only once. The execution semantics introduced above ensure that only valid activities are provided, meaning that the process participant follows one specific path through the process model, which is not laid out at process instantiation. The described task list solution is one implementation option but could easily be replaced by, for instance, multiple forms, tabs, or buttons describing upcoming work. However, these concepts can be traced back to task lists and only act as another representation. Usually, in the beginning of a business process, multiple paths are probable but with each decision taken to execute a specific activity, the number of remaining paths is reduced.

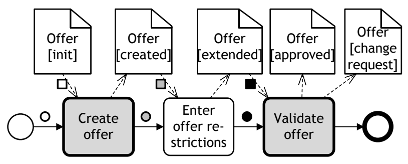
Below, we describe the execution semantics and the path reduction using the introduced travel agency example, which is presented with annotated tokens in Fig. 7. The tokens of circle and square shape present activity enablement at three different points in time during business process execution which are differentiated by the colors white, gray, and black. Circles denote control flow tokens while squares denote data flow tokens. Referring to rule (iii), an activity is enabled if data flow tokens are available on dashed edges indicating a read of a data node such that each corresponding data object class is referenced at least once and whether a control flow token is available on the corresponding control flow edge.



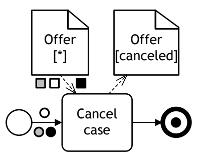
(a) Component p1.



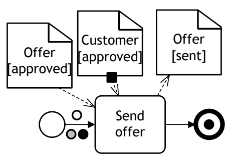
(b) Component p2.



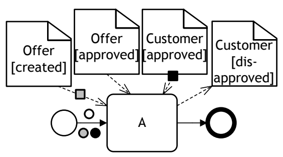
(c) Component p3.



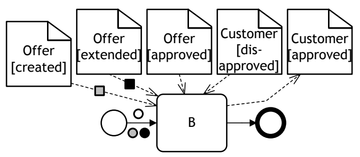
(d) Component p4.



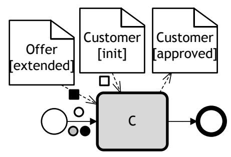
(e) Component p5.



(f) Component p6.



(g) Component p7.



(h) Component p8.

Fig. 7. Process components belonging to the travel agency business process with three different token distributions differentiated by white, gray, and black color. Circles on continuous edges targeting activities denote their enablement from the control flow point of view while squares on dashed edges denote the targeted activity's enablement from the data flow point of view.

B. Example: Token Play

For this process model, we assume that the conformance check was successful and that the termination condition indicates a proper termination if object *Offer* is in data state *sent* or *canceled* and the corresponding termination events in the respecting process components *p4* and *p5* are executed. Assuming the process components in Fig. 7 completely describe a process model, the instantiation of this process model instantiates all eight components by putting tokens into the eight start events, indicated by the white circles put on the edges originating from a start event. Both utilized data objects are in the initial states for this process instance such that data flow edges in components p1, p2, p3, p4, and p8 are assigned a white square, indicating that the corresponding data objects exist in the states specified in the data nodes. While control flow would enable each activity directly preceding a start event, data objects only enable the link activity *Create offer* and activity *Cancel case*, which are presented to the process participant for execution. For all other activities, at least one of the data objects is not in the state as defined by the respecting data nodes (cf. white squares).

Executing the cancellation results in a proper termination of the business process, because state *canceled* for object *Offer* is reached and the termination event is executed. Executing link activity *Create offer* enables activities *Validate offer* in p1, *Enter offer restrictions* in p3, and *Approve customer* in p2 in addition to the still enabled activity *Cancel case* in p4. The process participant cannot differentiate which variant of the link activity is actually executed as both have identical data inputs and outputs. Internally, the execution semantics ensure that the appropriate one is chosen. Following rule (iv) from above, both variants enable the control flow edge originating from them.

Next, the process participant may decide to execute activity *Validate offer* – again a link activity, but this time only the variant in p1 is enabled because the control flow enablement for the one in p3 is missing. Therefore, the mentioned variant is executed. Depending on the actual execution, object *Offer* is either transitioned to state *approved* or to state *change request*. In this example, the latter takes place leading to two enabled activities: *Cancel case* and *Change offer*. As p1 was terminated by reaching the end event, it gets re-instantiated (see rule (v)) and because of the data condition, the mentioned activity is enabled and ready for execution. In component p3, the control flow still allows enabling activity *Enter offer restrictions*, but as the data has changed, the enablement is refused. The same holds for process component p2. Changing the offer now results in the enablement situation as represented by the gray shaded tokens in Fig. 7. Executing activity *Approve customer* completes process component p2, re-instantiates it, and transitions object *Customer* to state *approved*. At the same time, the other variant of this link activity (activity C in p8) is not pushed forward as the output is not identical to the one from activity *Approve customer*. As the customer cannot be set into state *init* again (cf. Fig. 5), this link activity will not get enabled again in this process instance, meaning both process components will not contribute to the process goal anymore. With that, the number of probable paths through the process model is reduced step by step (cf. paradigm change P3). The remaining three tasks of the previous task list remain enabled as these utilize object *Offer* and are not affected by changes of object *Customer*. Additionally, object *Customer* in state *approved* fully enables activity *A* in process component p6 (cf. black tokens in Fig. 7).

Executing activity *Enter offer restrictions* enables activity *Validate offer* in process component p3 instead of p1. Besides, activity *Cancel case* of p4 is still enabled. Along this procedure, activity *Validate offer* of p3 is executed to approve the offer that then enables the execution of activity *Send offer* of p5 that transitions object offer into state *sent*. The offer in state *sent* with the termination event being executed represents a valid termination of the process instance.

In the described process instance, no activity of process component p8 is enabled at any point in time as the process participant decided for a different path through the process model. While in the beginning, p8 could have been part of the instance, the approval of the customer (see p2) before entering offer restrictions to the offer revealed that a path via process component p8 is not possible for the given process instance.

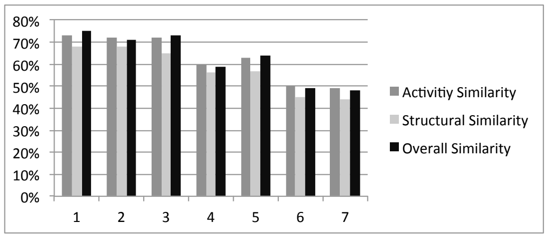
VI. Evaluation

We evaluated the presented approach in a customer project with a tourism corporation, where we focused on the methodology and user acceptance. The customer's team was set up with one process modeling expert, experienced in traditional BPMN and EPC modeling, and three business experts from different branches of the company. The investigated process of the customer focused on request handling from agencies to the back-offices. The existing process was handled manually via email, phone, and letters. Due to the high workload and the many resources required – in the company, up to 120 agents worked concurrently to process approximately 75.000 process instances a month – the introduction of a business process management system was investigated.

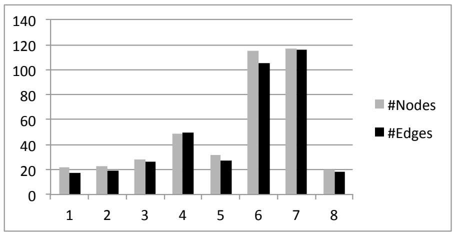
Our analysis of the existing process showed that the work of the employees focused on one single "happy path" that was used in approximately 60 to 70 variants. Due to the inability to capture all variants in process models, the customer decided to focus on the seven topmost variants and a *generic* happy path variant that should resemble most of the manual handling enhanced with certain logic and traceability. Further, the customer had the requirement that an employee should have the possibility to *move* from the execution of a certain variant to the generic "happy path" and vice versa at any time.

Since the resulting all-in-one process model would become very large, we introduced the concepts shown in this paper to capture the requirements. We started by modeling process components such as different variants that could trigger a new process (left out in this paper) and two concrete variants to create a conceptual understanding at the customer. Interestingly, the customer's experts had no problem modeling the concrete variants, whereas they were unsure about the generic process ("Is it too generic? Is it already too specific?"). We then introduced the required data modeling as well as the object life cycle to these variants and added *dashed transitions* to the life cycle to denote generic issues such as closing a case at any time without modeling this in any activity. After the conceptual understanding was established, we created a generic "happy path" process component that matched both existing variants. Finally, the five other variants have been modeled as process components based on the "happy path" component.

We compared the similarity and complexity of all seven process components that represent variants of the "happy path" component with the "happy path" component itself. Fig. 8a shows the similarity for (1) syntactical, semantic, and attribute-related similarity of the components (activity similarity), (2) structural similarity including contextual information and node ids, and (3) a combined, overall similarity measured according to the algorithms used in [19]. Fig. 8b shows the complexity of the components without data objects and associations. Both



(a) Similarity compared to "happy path".



(b) Complexity (no.8 is "happy path").

Fig. 8. Process component similarity and complexity for the seven topmost variants.

figures show that a correlation exists between the complexity of the process components and their similarity to the "happy path" component. Interestingly, the complex components no.6 and no.7 still have a similarity of around 45 to 49 percent. A closer investigation of these components showed that the customer introduced copies of the basic activities found in the "happy path" component that had only been changed slightly, e.g., from *call agency* to *contact agency* or *write email to agency*. Due to the applied Wordnet-based semantic-similarity measures for activities [20] and the structural equivalence checks via contexts based on refined process structure trees (RPST) [21], the components are detected as similar even while the absolute number of activities has been multiplied. Referring to the paradigm changes explained in Section III, i.e., (P1) a process instance is composed of different process component instances, (P2) the complete execution path is not visual, and (P3) *alternation* between components should be possible, we found a good acceptance at the customer side. In particular, the customer was able to work on the five remaining process components with only limited coaching/reviewing from our side.

Finally, we qualitatively assess the understandability of the given process model. Splitting complex process models into multiple components following the methodology presented in Section IV eases understanding of the process behavior. Naturally, each of the smaller components is easier to understand than the fully-pledged complex process model, but the combination of all components might not be as the process flow is distributed. However, as presented, each process component consists of one specific case, e.g., the happy path, an exception handling path, or a user-specific need for the distinction of customer classes. Thus, assuming that each component presents such an isolated part of the overall process, we can safely assume that the set of process components is easier to understand in terms of process behavior than the large, combined process model.

VII. Related Work

There are multiple approaches around dealing with flexibility in process models from different points of view. Declarative process modeling [22] is activity-centric and allows to forbid certain behavior by constraints, easing the specification of flexible processes by avoiding to model all probable paths. Based thereon, only allowed and executable activities are presented to the process participant who then decides about activity execution. Although it is activity-driven, data can be considered by adding it as a constraint to the activity. However, it seems more natural to model what steps have to be undertaken to achieve a business goal rather then specifying what must not be done. Case handling [12] deals with the data perspective, only allowing to start execution of an activity based on data dependencies. If a preceding activity, for instance, provides the information required to start the succeeding one, execution may start although the preceding one is not yet finished. Proclets [23] focus on the communication between lightweight process snippets which partition the process into many parts which are executed after another or interactive. Thereby, they allow late binding of the communication partners and provide flexibility with respect to the participating parties. These snippets relate to our process components in the process model but do not provide alternative execution behavior except from creating large processes models with many paths showing each variant.

Coping with this issue, the Provop approach [6] provides a flexible solution to manage variants by utilizing context-aware change operations that are specified with respect to a reference process. Conceptually, each variant is a set of change operations specifically catered to a certain use case. Amongst others, the authors describe policies to create the reference process and to make sure that existing components do not get invalidated while the reference process evolves. Thereby, the process model contains control flow nodes only. The reference process can be referred to the "happy path" component in our approach but with different focus as we concentrate on data and control flow. Further, we model the remaining variants not as a set of change operations, but as process components which allow the variant to be directly seen and ease the modeling approach, because the user does not need to learn new concepts. However, both approaches complement each other such that the concepts can be merged for a holistic approach.

Process variants provide flexibility of processes during design time with respect to variability. Similar to the Provop approach, change patterns [7] help to classify the changes to process models and allow the retracing of changes done to a process model to ensure process correctness and robustness. These change patterns are used to evaluate the ability of process-aware information systems to handle process changes on the process model as well as the instance level. [24] elaborates extensively on the need of such systems. Flexibility of processes is needed for exception handling, variability management, and to cope with more unstructured processes resulting in four flexibility needs: variability, looseness, adaptation, and evolution. Our approach covers all of them. We support multiple variants with process models that contain several process components whose combination during run-time describes a production case where the components get executed as needed.

The oclet approach [10] with its extension towards data support [25] comes closest to our framework. In fact, each oclet represents one specific part of the overall process and at run-time, based on data and control flow dependencies, a run, i.e., a trace, through the set of all scenarios is calculated. However, oclets are presented as Petri nets which positively add to the analyzability through the formal character but also increase the usability barrier as practitioners rarely apply formal methods. They require a more abstract view. Here, our focus on BPMN comes into play, which is widely understood and also utilized by practitioners in everyday working life. Additionally, the utilization of Petri nets implies implicit modeling of data objects and states which possibly lead to ambiguities; e.g., it is not clear which object an action manipulates. Explicit modeling of data objects as in our component approach helps in this regard. An explicit data view, as provided with the OLCs, eases the modeling of data dependencies and the specification of data manipulations. These aspects are completely out of scope in [10]. The data extension does not provide such a data view nor explicit data state modeling, but it allows the addition of process instance data to the process model which is currently out of scope for our Production Case Management framework.

In practice, usually large numbers of process variants respectively process schemes must be handled. [8] addresses this issue by introducing a flexible process design, combining generic process templates and business rules. A process schema including control flow, resources, and data is generated from a template by applying business rules to it. The generation takes place upon process instantiation and cannot be changed during run-time. Therefore, the flexibility is limited to the design-time instead of both as in the presented framework. [9] motivates the need to individualize reference models at design-time. Unlike the Provop approach, configurable models showing the different variants as models handle the individualization. Again, run-time flexibility is not provided.

Utilizing data dependencies is not new. Thereby, the control flow is leading and upon the start of an activity, the data dependencies are checked. The processing of the activity is set on hold if the data dependencies are not met. On the other side of the spectrum, object-centric approaches [26]–[29] exist which led to the case management model and notation standard [30] positioned as adaptive case management. Here, the existence of objects in specific states decides about execution. Control flow plays a very minor role and is mostly given implicitly. In this paper, we combine the data and control flow perspectives; both being similarly important for process execution. Further, we introduce flexibility by allowing to *alternate* between different process components to cope with current challenges, moving traditional process modeling with BPMN into the field of production case management [1], the pre-stage towards adaptive case management.

The transformation of process models into OLCs or vice versa is discussed in various approaches, e.g., [13]–[15], [31]. The challenge within our Production Case Management framework is to deal with dependencies between multiple OLCs. For instance, the offer may only be *sent* to the customer, if she was *approved* before. Most existing approaches including [13], [14], [31] cannot deal with these inter-dependencies. [15] can and thus we decided to utilize these algorithms for the transformation between process models and object life cycles.

VIII. Conclusion

In this paper, we presented an implementation framework, integrating production case management concepts into activity-centric process modeling by specifying a modeling and an execution perspective leading to flexible process management at both: design-time and run-time. This framework allows the representation of a complex business process in smaller and easier to understand components that are interconnected by control and data flow, rather than one large process model as common in configurable or traditional process modeling. Thereby, the approach is based on a run-time repository, where the process components are interpreted during run-time. Hence, the components can be changed, added, or removed even while executing instances. This paper focuses on presenting the concepts and leaves tooling to future work. Thereby, tooling especially needs to handle the consistency of distributed modeled components and their complexity; e.g., process model abstraction is a proper technique for tackling complexity.

In detail, we represent a process model in terms of process components that collectively describe the process behavior. During process execution, the process participant may alternate between several components, deciding at run-time which path to follow through the process. At design-time, the various components allow the modeling of the "happy path" on the one side and exceptional behavior and additional details on the other side. We discussed the application of this framework in an industry project with a large tourism corporation and showed the very positive impact of such an integrated framework.

In future work, we will present the formalization in terms of a Petri net mapping to support the methodology and execution semantics presented here. Thereby, we will evaluate whether the oclet [25] approach might be utilized to build on existing work and reuse existing analyzing techniques as, for instance, soundness checking to ensure deadlock freedom. In addition, we will work on formal verification, including data conformance between the process components and the object life cycles as well as validity of the termination condition. Further, we will allow more operations within the generated object life cycles, e.g., adding synchronization edges without constraints or deleting edges of any type.

References

[1] K. D. Swenson, "State of the art in case management," Fujitsu, Tech. Rep., March 2013.

[2] OMG, "Business Process Model and Notation (BPMN), Version 2.0," 2011.

[3] M. Weske, *Business Process Management: Concepts, Languages, Architectures. Second Edition*. Springer, 2012.

[4] K. D. Swenson, *Mastering The Unpredictable: How Adaptive Case Management Will Revolutionize The Way That Knowledge Workers Get Things Do*. Meghan-Kiffer Press, 2010.

[5] J. B. Hill, "The case for case management solutions," Gartner, 2012.

[6] A. Hallerbach, T. Bauer, and M. Reichert, "Capturing variability in business process models: the provop approach," *Journal of Software Maintenance and Evolution*, vol. 22, no. 6-7, pp. 519–546, 2010.

[7] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features–enhancing flexibility in process-aware information systems," *Data & Knowledge Engineering*, vol. 66, no. 3, pp. 438–466, 2008.

[8] A. Kumar and W. Yao, "Design and management of flexible process variants using templates and rules," *Computers in Industry*, vol. 63, no. 2, pp. 112–130, 2012.

[9] M. Rosemann and W. M. van der Aalst, "A configurable reference modelling language," *Information Systems*, vol. 32, no. 1, pp. 1–23, 2007.

[10] D. Fahland, "From Scenarios to Components," Ph.D. dissertation, Humboldt-Universität zu Berlin, 2010.

[11] M. Reichert and P. Dadam, "Enabling adaptive process-aware information systems with adept2," in *Handbook of Research on Business Process Modeling*. Information Science Reference, 2009, pp. 173–203.

[12] W. M. P. van der Aalst, M. Weske, and D. Grünbauer, "Case Handling: A New Paradigm for Business Process Support," *Data & Knowledge Engineering*, vol. 53, no. 2, pp. 129–162, 2005.

[13] J. Küster, K. Ryndina, and H. Gall, "Generation of Business Process Models for Object Life Cycle Compliance," in *Business Process Management*. Springer, 2007, pp. 165–181.

[14] K. Ryndina, J. Küster, and H. Gall, "Consistency of Business Process Models and Object Life Cycles," in *MoDELS Workshops*. Springer, 2006, pp. 80–90.

[15] A. Meyer and M. Weske, "Activity-centric and Artifact-centric Process Model Roundtrip," in *Business Process Management Workshops*. Springer, 2013, pp. 167–181.

[16] A. Meyer, L. Pufahl, D. Fahland, and M. Weske, "Modeling and Enacting Complex Data Dependencies in Business Processes," in *Business Process Management*. Springer, 2013, pp. 171–186.

[17] OMG, "Unified Modeling Language (UML), Version 2.4.1," 2011.

[18] A. Meyer, A. Polyvyanyy, and M. Weske, "Weak Conformance of Process Models with respect to Data Objects," in *Services and their Composition (ZEUS)*, 2012, pp. 74–80.

[19] F. Friedrich, J. Mendling, and F. Puhlmann, "Process model generation from natural language text," in *CAiSE*. Springer, 2011, pp. 482–496.

[20] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[21] J. Vanhatalo, H. Völzer, and J. Koehler, "The Refined Process Structure Tree," *Data & Knowledge Engineering*, vol. 68, no. 9, pp. 793–818, 2009.

[22] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg, "Declarative workflows: Balancing between flexibility and support," *Computer Science-Research and Development*, vol. 23, no. 2, pp. 99–113, 2009.

[23] W. M. P. van der Aalst, P. Barthelmess, C. A. Ellis, and J. Wainer, "Proclets: A Framework for Lightweight Interacting Workflow Processes," *Int. J. Cooperative Inf. Syst.*, vol. 10, no. 4, pp. 443–481, 2001.

[24] M. U. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer, 2012.

[25] D. Fahland and R. Prüfer, "Data and abstraction for scenario-based modeling with petri nets," in *Application and Theory of Petri Nets*. Springer, 2012, pp. 168–187.

[26] V. Künzle and M. Reichert, "PHILharmonicFlows: Towards a Framework for Object-aware Process Management," *Journal of Software Maintenance and Evolution*, vol. 23, no. 4, pp. 205–244, 2011.

[27] D. Cohn and R. Hull, "Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes," *IEEE Data Engineering Bulletin*, vol. 32, no. 3, pp. 3–9, 2009.

[28] G. Redding, M. Dumas, A. H. ter Hofstede, and A. Iordachescu, "A flexible, object-centric approach for business process modelling," *SOCA*, vol. 4, no. 3, pp. 191–201, 2010.

[29] D. Dori, *Object Process Methodology: A Holistic Systems Paradigm*. Springer, 2002.

[30] OMG, "Case Management Model and Notation (CMMN)," January 2013.

[31] R. Liu, F. Y. Wu, and S. Kumaran, "Transforming Activity-Centric Business Process Models into Information-Centric Models for SOA Solutions," *Journal of Database Management*, vol. 21, no. 4, pp. 14–34, 2010.