# A One-Dimensional Kalman Filter for Real-Time Progress Prediction in Object Lifecycle Processes

Lisa Arnold
*Institute of Databases and Information Systems*
*Ulm University, Germany*
lisa.arnold@uni-ulm.de

Marius Breitmayer
*Institute of Databases and Information Systems*
*Ulm University, Germany*
marius.breitmayer@uni-ulm.de

Manfred Reichert
*Institute of Databases and Information Systems*
*Ulm University, Germany*
manfred.reichert@uni-ulm.de

***Abstract*—Real-time monitoring of business processes offers promising perspectives to discover problems and optimisation potentials. Early detection is a key part in this endeavour. One crucial aspect of real-time monitoring is to determine the current progress of a running business process. This is particularly challenging for business processes that consist of a multitude of loosely coupled, smaller processes that interact with each other, like object lifecycle processes in data-centric approaches to business process management. In this paper, an approach to predict the remaining portion of the process path to be still executed in relation to the overall process is proposed. This prediction is based on a one-dimensional Kalman Filter. As a major benefit of this approach, real-time progress determination can start directly with the first run of the process, i.e., without need for comprehensive event log data. This becomes possible due to the procedure applied by the Kalman Filter, which requires no log data. A quantitative study with 250 progress estimations for large object lifecycle processes results in a deviation of the average estimated progress from the real progress, calculated after the completion of the process, of about 5%. This emphasises that reasonable progress predictions are possible even in the absence of an event log, as it is the case when deploying new or changed processes to the run-time system.**

***Index Terms*—Business process monitoring, object lifecycle process, real-time progress, prediction, one-Dimensional Kalman Filter**

## I. Introduction

To stay competitive, companies need to continuously improve and evolve their business processes. In this context, monitoring is crucial to discover problems and optimisation potentials of business processes. Additionally, real-time monitoring can support process participants during run-time, for example, through the provision of an overview about the current progress of a process as well as potential delays or errors. Real-time monitoring can display the current state of the business process and indicate which activities or sub-processes are on time, at risk or overdue. When triggering an alarm for an activity (or the entire business process); e.g., in case a sub-process is assessed as being at *"on risk"*, it becomes possible to take timely actions to prevent this assessment changing to *"overdue"*. This kind of monitoring has already been implemented in several Business Process Management (BPM) tools. Another fundamental approach is to monitor and display the current progress of the business process to involved stakeholders.

Currently, however, there are only a few BPM tools with capabilities for real-time process monitoring and progress measurement is usually not supported. To remedy this drawback, we developed a method for measuring process progress in object-aware BPM and implemented this method in PHILharmonicFlows [16]. In principle, the approach introduced in this paper can also be used to predict progress in activity-centric BPM tools. Note that there are predictive monitoring approaches [5] that can be used to determine the progress of processes with multiple execution paths. However, these approaches presume the existence of an event log, but are unable to determine the progress for processes without log data [5]. Note that there are business processes that need more than one year to collect a representative log data set. Thus, for all business processes with low transaction rates, progress monitoring is not possible when deploying a new or changed process. To deal with this situation, an approach for predictive monitoring and progress estimation is needed that can be used without log data.

In this paper, we introduce an approach that is solely based on the schema of the business processes. This approach uses graph theory to determine all possible paths from the current state to an end state of a business process. With these possible paths, the estimation of a one-dimensional Kalman Filter [6], [7], [10], [13], [22] is used to predict the remaining path of a business process without utilising any log data. With this estimation, in turn, the current progress of a lifecycle can be determined.

Object-aware business processes consist of multiple interacting objects whose relations are defined in a directed relational process structure [15]. Each object is defined by its attributes and an object lifecycle process describing its run-time behaviour. In contrast a coordination process controls the interactions between multiple lifecycle processes, i.e., the overall business process [19]. In the following, an *object lifecycle process* is denoted as a *lifecycle process*. Fig. 1 illustrates the structure of an object-aware business process with simple lifecycle processes in their state-based view. Taking this process structure, the procedure of determining the progress can be mapped to few tasks. First, we must be able to determine the progress of a single lifecycle process (cf. Fig. 1) with its state-based view. Second, we need to refine progress measurement of a single state within a lifecycle process taking the data-driven execution of lifecycle processes into account (i.e. data-driven state transition). Third, we must investigate progress determination of multiple, interacting (i.e., interrelated) lifecycle processes and, fourth, the way the coordination process (see dependencies in Fig. 1) of an object-aware business process influences its progress [2]. In this paper, the first and second challenge of determining the progress of a single lifecycle process are addressed. In a linear lifecycle or a lifecycle with equally long paths, like the simple lifecycles depicted in Fig. 1, progress determination is trivial. However, for lifecycles with a multitude of possible paths of different lengths, progress determination is challenging. Particularly, for large lifecycle processes, progress determination depends on accurate path predictions.

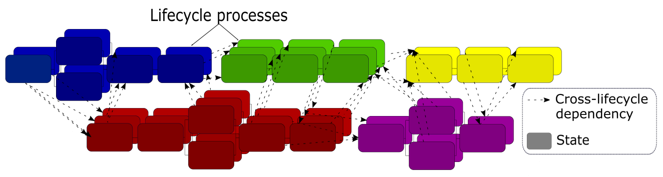

Fig. 1: Abstract example of the structure of an object-aware business process with its interacting lifecycle processes (in their state-based view).

One major contribution of this paper is to define an approach for estimating the current progress of running lifecycle processes without need for event log data. In this context, progress determination methods are defined in terms of pseudo code to define the Kalman Filter approach. Additionally, an implementation based on this pseudo code is introduced for the object-aware BPM system PHILHARMONICFLOWS [16]. Finally, the implementation is evaluated with 250 progress predictions from several object lifecycles.

The remainder of this paper is structured as follows. Section II gives an overview on (A) the object-aware BPM paradigm with its most fundamental components and, (B) the procedure of the one-dimensional Kalman Filter. Section III shoes how to calculate the required input values for the Kalman Filter to apply the latter to lifecycle processes. In Section IV progress determination methods are presented based on the results from Section III. Section V evaluates the approach. First, the design of the evaluation, which comprises a qualitative as well as a quantitative study, is presented. Second, the results of 250 progress estimations performed in the context of different business process are discussed. Finally, Section VII concludes the paper.

## II. Fundamentals

### A. Object-Aware Business Process Lifecycle

Object-aware business processes consist of interacting objects whose relations are defined in a directed relational process structure [15]. The relation between objects and their lifecycle processes, respectively, induces a hierarchy within a relational process structure with its lower- and higher-level relations between objects. Cardinality constraints for these objects are 1:n or n:m [17]. Each object of a relational process structure is defined by its attributes and lifecycle process, with the latter describing object behaviour at run-time [15]. Finally, a coordination process controls the interactions between the multiple lifecycle processes forming the overall business process [19].

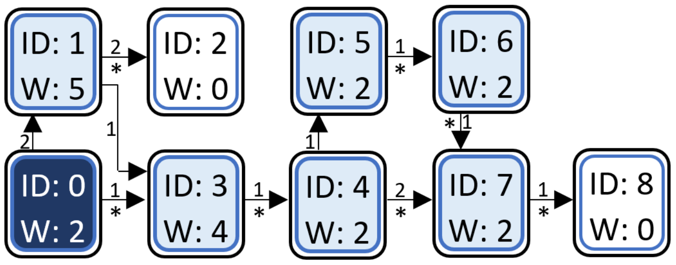

Fig. 2: Structure of a lifecycle with its state-based view. The dark blue state is the start state, white states are end states, and light blue states are intermediate ones. ID is the unique identifier of the state and W represents its weight (i.e., the effort required to process the state). The number next to a transition arrow represents the priority, whereas * marks the most frequently used path.

Each lifecycle process of an object-aware business process comprises states (including exactly one start and at least one end state) as well as state transitions. In Fig. 2, a lifecycle in its state-based view is shown and used as running example in the following. During run-time, for each instance of an object, a lifecycle process instance (formally superscript$^I$) is created. During the execution of the latter, to each state one of the markings described in Table I is assigned. A weight between $1$ and $5$ can be assigned to each state, except the end states that receive $0$ as weight. The weights are assigned at design-time with a default weight of $3$ (i.e., the median). A modeler decides whether a state requires more or less effort than the average and then adapts the weight accordingly.

| **Marking** | **Description** |
|---|---|
| Waiting | The state has not been executed yet. A predecessor state is activated. |
| Pending | The activation of the state is blocked due to an unfulfilled coordination constraint (derived from the relation to other running lifecycles processes). |
| Activated | The state is currently executing. |
| Confirmed | The state has been successfully executed. A successor state is activated. |
| Skipped | The state can no longer be executed. A state on an alternative execution branch was chosen. |

TABLE I: Possible state markings at run-time [18]

At run-time, exactly one active state exists per lifecycle process instance. Consequently, no parallel execution within a particular lifecycle process instance is possible. However, several lifecycle process instances maybe executed in parallel enabling concurrency on the business process level. The execution of a lifecycle process instance begins with its instantiation and the marking of the unique start state as *"Activated"*. It ends with marking one of the end states as *"Activated"*. Throughout the execution of a business process instance all associated lifecycle process instance sole present and will not be discarded, even if these are in a end state. Formally, each end state must be a silent state (i.e., a state without any action). When marking the end state as *"Activated"*, the execution of the lifecycle process instance terminates. A single state, in turn, is defined by a number of steps, of which each is related to an object's attributes whose values it needs to see when executing the state.

In addition, forms enabling the user input (i.e. the writing of the attributes) are generated dynamically for each state using the information on its steps (cf. Fig. 3). Moreover, backward transition between two states may be defined at design-time. Based on a backward transition, jumping back from a given state to a previous state becomes possible such that the attributes of target state may be updated. In case of a backward jump, the data previously filled in the form of the state is re-displayed.

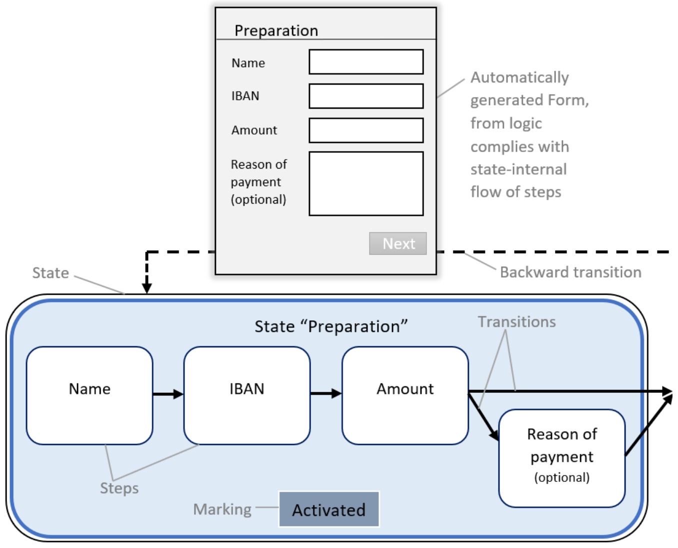
Preparation
Name
IBAN
Amount
Reason of payment (optional)
Next
Automatically generated Form, from logic complies with state-internal flow of steps
State
Backward transition
State "Preparation"
Transitions
Name
IBAN
Amount
Reason of payment (optional)
Steps
Marking
Activated

Fig. 3: The start state *"Preparation"* from a banking process with its step-based view and its automatically generated form.

For a given state the automated generation of its form is accomplished based on its steps and their order. As discussed, the latter allow refining a state. Fig. 3 shows state *"Preparation"* from a banking process with its step-based view. Each step represents an attribute of the object, or more precisely an update operation on this attribute. As for the structure of the states, the steps are connected with transitions (but no backward transitions, which are solely allowed between the states of a lifecycle process). The execution mode is defined by step markings with no parallel execution being possible. With a predicate step a decision can be defined to realise conditional branches. Finally, for all outgoing transitions a priority must be set, which represents the standard path.

## *B. One-Dimensional Kalman Filter*

In a non-linear lifecycle, calculating progress at run-time is challenging as the still to be executed (i.e. remaining) path needs to be predicted. One possible approach for calculating the progress in a running lifecycle process is the use of a Kalman Filter. The latter constitutes an iterative mathematical process that can estimate the remaining weight of the path that still needs to be executed to complete lifecycle execution. In general, Kalman Filter can estimate the real value. In this context, the remaining weight is estimated in order to determine the progress in relation to the total weight. This estimation is based on the measured value of the unexpected error (e.g. measuring error) with a set of equations and consecutive data inputs (measurements). Kalman Filter eliminates these errors or variations from the data measurements and projects the measurements onto the estimate [6], [10]. A model of the procedure of a Kalman Filter is depicted in Fig. 4. It can be divided into three parts of which each is defined by an equation given in (1), (2), and (3) in the following.

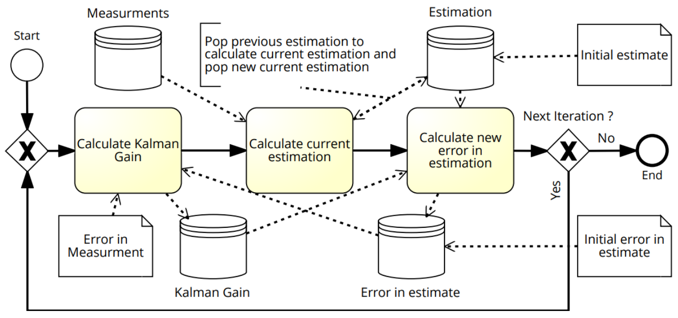
Start
Measurments
Pop previous estimation to calculate current estimation and pop new current estimation
Estimation
Initial estimate
Calculate Kalman Gain
Calculate current estimation
Calculate new error in estimation
Next Iteration ?
No
End
Yes
Error in Measurment
Kalman Gain
Error in estimate
Initial error in estimate

Fig. 4: Procedure of the Kalman Filter.

The first activity of the Kalman Filter procedure, shown in Fig. 4, calculates Kalman Gain (KG), i.e., the percentage the estimation error contributes to the total error (error in estimate plus measurement). The calculated value is used to minimise the unexpected error or variation of the estimate. KG is defined by Formula (1). For its calculation, Error in Estimate ($E_{EST} \in \mathbb{R}^{+}$) and Error in Measurement ($E_{MEA} \in \mathbb{R}^{+}$) are required, with Error in Measurement being a fixed value. Error in Estimate as well as KG are recalculated in each iteration. Note that the number of iterations corresponds to the number of Measurements. KG specifies the proportion between the estimation error and the total error.

$$KG = \frac{E_{EST}}{E_{EST} + E_{MEA}} \tag{1}$$
$$\text{with } KG \in \{x \mid x \in \mathbb{R},\ 0 < x \leq 1\}$$

The closer KG is to 1, the bigger the Error in Estimate is and the smaller Error in Measurement. In contrast, the closer KG is to 0, the smaller Error in Estimate is and the bigger Error in Measurement. KG is recalculated in each measurement. The current estimation error is recalculated in the third activity of Kalman Filter procedure (cf. Fig. 4).

The second activity of the Kalman Filter procedure, (cf. Fig. 4) calculates the Current Estimate ($EST_t \in \mathbb{R}^+$). In the first iteration ($t = 1$, $t$ marks count of iteration) of calculating Kalman Filter, an initial estimate as well as the first Measurement ($MEA \in \mathbb{R}^+$) are needed to calculate the current estimation. In case of progress determination of a single lifecycle process, the average weight of all possible paths in the lifecycle is used as this initial estimate. Additionally, KG as calculated by the first activity of Kalman Filter is required. In the first iteration of calculating Kalman Filter the value of the initial estimate is used as Previous Estimate ($EST_{t-1} \in \mathbb{R}^+$). Accordingly, all further iterations use the result of the preceding calculation of Formula (2).

$$EST_t = \underbrace{EST_{t-1}}_{Part1} + \underbrace{KG \cdot \big(MEA - EST_{t-1}\big)}_{Part2} \qquad (2)$$

Formula (2) determines Current Estimate. For this purpose, the estimation is recalculated with the next data input (Measurement). More precisely, the second part of the equation proportionately computes the difference of Measurement and Previous Estimate with KG. In this context, multiplying the difference between Measurement and Previous Estimate with KG determines the percentage of the modification from the previous estimate. The impact on Current Estimate is minimal with a large Error in Measurement and a small Error in Estimate, as KG is close to zero. Depending on whether Previous Estimate is bigger or smaller than Measurement, Current Estimate decreases or increases.

The third activity of the Kalman Filter procedure, (cf. Fig. 4) recalculates the estimate error for the next iteration. New Error in Estimate is denoted as $E_{EST_t} \in \mathbb{R}^+$ and the previous Error in Estimate, which is used in the current iteration, is as $E_{EST_{t-1}} \in \mathbb{R}^+$. Therefore, the Error in Estimate is multiplied with the proportion of Error in Measurement and Total Error [6], [7]. This calculation is given by Formula (3).

$$E_{EST_t} = \big(1 - KG\big) \cdot E_{EST_{t-1}} \qquad (3)$$

Altogether, Kalman Filter needs an Initial Estimate and Initial Error in Estimate as input, and an immutable Error in Measurement (unchanging). Additionally, in each iteration the data input (i.e. Measurement) is required to calculate the estimation of the remaining still to be executed path.

## III. Lifecycle Progress Prediction with Kalman Filter

Kalman Filter is now applied for predicting the progress of a running lifecycle process (cf. Section II). More precisely, Kalman Filter is applied every time a state changes its marking to *"Activated"* (cf. Table I) in order to determine the current progress of the running lifecycles process. Therefore, the sum of weights of all confirmed states (cf. Table I) in relation to the sum of weights of the executed path (confirmed states plus active state plus unknown states still to be executed) are calculated. At run-time, the path to be executed is not known. For example, if state 3 of the lifecycle from Fig. 2 is the current active state, two possible paths exit. The lower one (states: 3-4-7-8) with a total weight of 8 and the upper one (states: 3-4-5-6-7-8) with a total weight of 12. To predict the remaining weight for any from the active state 3 to an end state, Kalman Filter is applied.

The data inputs of Kalman Filter are various measurements based on all possible paths from a given state to the end states. Due to the fact that the measurements are independent of the execution of the lifecycle process as well as the Kalman Filter procedure, they can be already calculated at design-time for each lifecycle type. At run-time, the precalculated measurements can be used to allow for a real-time progress calculation. For each state, an array comprising the following five measurements is calculated:

| | |
|---|---|
| **[0]** | Average weight of all possible paths |
| **[1]** | Highest weight of all possible paths |
| **[2]** | Lowest weight of all possible paths |
| **[3]** | Weight of the priority path |
| **[4]** | Weight of the most frequently used path |

The most frequently used (MFU) path may only be considered after several runs of the lifecycle process. At run-time, this additional measurement is realised with a check on whether there exists a sufficient number of runs.

### A. Initialisation of Kalman Filter

For the first application of Kalman Filter, initial values for Error in Estimation, Error in Measurement, and Initial Estimate need to be provided. Using Kalman Filter in a physical scenario, for example, a measurement of a distance sensor might have Error in Measurement. The graph algorithms, used to calculate the array have, in general, no Error in Measurement. For this reason, Error in Measurement is set to a number near zero (zero is not possible). The biggest possible Error in Estimation is the difference of remaining weights between the path with the highest and lowest weight. Consequently, this difference is chosen as initial Error in Estimation. Initial Estimate, in turn, is set to the first measurement (data input). After initialising the Kalman Filter, the multiple measurements from the array are used as input.

### B. Handover parameter for Kalman Filter

For each state, Kalman Filter has the five defined measurements as input: average, highest, lowest, priority and most frequently used weight. These input parameters are stored in a two-dimensional array and are calculated by Algorithms 1 and 2 before run-time. For each state $S$, Algorithm 1 determines a list that stores the sum of weights for each possible path from $S$ to an end state. For this purpose, all list elements from all successor states are added with the current state weight and are appended to the list of State $S$ (cf. line 11-12). To ensure that all successor state have a complete list, a topological sorting (cf. Line 7) starting with the last element is applied. [12].

**Algorithm 1:** AllPaths($\Theta^I$)

```
Input  : One lifecycle $\Theta^I$ that contains all states $\Sigma^I$.
Output: A list (for each state) of lists containing all
        weights of possible paths for this state.
1  begin
2      $topSort \leftarrow topSorting(\Sigma^I)$
3      $i \leftarrow |\Sigma^I|$
4      initialise $list$
5
6      while $topSort \ != empty$ do
7          $\sigma^I \leftarrow topSort.last$
8          if $\sigma^I.next\ = empty$ then
9              add $\sigma^I.weight$ to $list[\sigma^I.ID]$
10         else
11             forall $\sigma^I.next$ of $\sigma^I$ do
12                 add
                   $\sigma^I.next.list[\sigma^I.ID].elem+\sigma^I.weight$
                   to $list[\sigma^I.ID]$
13         remove $topSort.last$
14     return $list$
```

Algorithm 2 is used to calculate the five defined measurements for each state. To store these measurements for all states, a two-dimensional array with size $5 \times i$ (or $4 \times i$ if MFU is not known) is initialised (cf. Line 4) with $i$ being the number of states of the given lifecycle. Concerning the running example (cf. Fig. 2), the two-dimensional array calculated by Algorithm 2 is shown in Table II. Note that the calculation of the first three measurements – AVG, MAX and MIN – is trivial (cf. Lines 7-9).

| MEA | State | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 (AVG) | 13.8 | 12.33 | 0 | 11 | 7 | 8 | 5 | 2 | 0 |
| 1 (MAX) | 21 | 19 | 0 | 14 | 10 | 8 | 5 | 2 | 0 |
| 2 (MIN) | 7 | 5 | 0 | 8 | 4 | 8 | 5 | 2 | 0 |
| 3 (PRIO) | 16 | 19 | 0 | 14 | 10 | 8 | 5 | 2 | 0 |
| 4 (MFU) | 10 | 5 | 0 | 8 | 4 | 8 | 5 | 2 | 0 |

TABLE II: Array resulting from the application of Algorithm 2 to the running example.

For calculating the weight of the priority path (cf. Lines 11-17), a loop is used. The result $sumWeight$ is the weight corresponds to the current state as well as all weights of subsequent states that are connected with a transition marked with priority 1. The weight measurement of the most frequently used path can only be calculated if a sufficient number of runs of the lifecycle process exists (cf. Line 19). The calculation is based on the same principle as the one of the priority path by following the path marked as the most frequently used one (cf. Lines 19-26). The result of Algorithm 2 is used as the measurements for the second activity of Kalman Filter procedure (cf. Fig. 4).

**Algorithm 2:** 2DimensionalArray($\Theta^I$)

```
Input: One lifecycle $\Theta^I$ that contains all states $\Sigma^I$.
Output: two-dimensional array $i \times j$. $i$ being the
        number of states, in turn $j = 0$ defines the
        average weight, $j = 1$ the highest weight,
        $j = 2$ the lowest weight, $j = 3$ the weight of
        the priority path, and $j = 4$ the weight of the
        path most frequently used.
1  begin
2      $list \leftarrow AllPaths(\Theta^I)$
3      $count \leftarrow |\Sigma^I|$
       /* w. MFU size 5, w/o. size 4    */
4      initialise $array[5][i]$
5
6      for $0 \le i < count$ do
7          $array[0][i] \leftarrow AVG(list[i])$
8          $array[1][i] \leftarrow MAX(list[i])$
9          $array[2][i] \leftarrow MIN(list[i])$
10
           /* Calculate prio weight      */
11         $sumWeight \leftarrow 0$
12         $\sigma^I \leftarrow \sigma^I\ with\ (ID == i)$
13         do
14             $sumWeight \leftarrow sumWeight + \sigma^I.weight$
15             $\sigma^I \leftarrow \sigma^I.next\ with\ (prio == 1)$
16         while $\sigma^I.next\ != empty$
17         $array[3][i] \leftarrow sumWeight$
18
           /* Calculate MFU weight       */
19         if $MFU\ exists$ then
20             $sumWeight \leftarrow 0$
21             $\sigma^I \leftarrow \sigma^I\ with\ (ID == i)$
22             do
23                 $sumWeight \leftarrow$
                     $sumWeight + \sigma^I.weight$
24                 $\sigma^I \leftarrow \sigma^I.next\ with\ MFU\ set$
25             while $\sigma^I.next\ != empty$
26             $array[4][i] \leftarrow sumWeight$
27
28     return $array$
```

## IV. Determining the progress of object lifecycles

This section shows how the progress of a lifecycle process can be determined based on the Kalman Filter estimation.

In the **first step**, Algorithms 1 and 2 are executed once before running any instance of the lifecycle process. The algorithms return a two-dimensional array that captures the required measurements (average, highest, lowest, priority, and most frequently used weight) for each state.

In the **second step**, the Kalman Filter is applied to calculate the estimated weight starting from the current lifecycle process state to a possible end state. At run-time, this calculation

**Algorithm 3:** LPD($\Theta^I$)

```
Input: One lifecycle $\Theta^I$ that contains all states $\Sigma^I$.
begin
    $list_i \leftarrow AllPaths(\Theta^I)$
    $array[][] \leftarrow 2DimensionalArray(list_i)$
    $progress \leftarrow 0$

    do
        if $\sigma^I_A.next\ != \emptyset$ then
            $progress \leftarrow 100$
            output progress
        else
            $\sigma_A \leftarrow activeState(\Theta^I)$
            $est \leftarrow kalmanFilter(array[][\sigma_A])$

            $doneWeights \leftarrow 0$
            forall $\sigma^I$ marked as Confirmed do
                $doneWeights \leftarrow doneWeights + \sigma^I.weight$

            $sumWeights \leftarrow est + doneWeights$

            $progress \leftarrow \frac{doneWeights}{sumWeights} * 100$

            $progressMax \leftarrow \frac{doneWeights + \sigma^I_A.weight}{sumWeights} * 100$

            /* Algorithm 4 */
            $refineProgress(progress, progressMax, \sigma_A)$

    while process run
```

is performed each time a state becomes activated taking the results of Algorithms 1 and 2 as input as well.

In the **third step**, the current progress can be calculated. This calculation is based on Algorithm 3, which we denotes as LPD (**L**ifecycle **P**rogress **D**etermination). Algorithm 3 receives the lifecycle instance for which the current progress shall be determined as input. The first part of Algorithm 3 calculates the two-dimensional array by invoking Algorithms 1 and 2 (cf. Lines 2-3). Then the current progress is calculated continuously during run-time. This is realised by a loop (cf. Lines 6-25), which terminates when reaching an end state of the lifecycle process. If the current state corresponds to an end state, the progress is set to 100 percent (cf. Line 8). Otherwise, the estimated weight of the remaining process is calculated with the Kalman Filter (cf. Line 12). To determine the current progress the weights of all confirmed states (cf. Table I) are summed up (cf. Lines 14-16). With this sum and the estimation provided by Kalman Filter the estimated total weight can be calculated (cf. Line 18). This result describes, the estimated progress at the point in time switching to the current state (cf. Line 20). For the calculation of the upper progress boundary, the sum of the weights from all confirmed states as well the weight of the current state are added (cf. Line 22). Accordingly, the size of this interval depends on the weights of the states. The realisation with a left-closed and right-open interval ensures that the progress of a state is non-overlapping with any previous or succeeding state. In this approach, a percentage of 100 is not possible, as a right-open progress interval not includes 100. When marking an end state of the lifecycle as *"Activated"*, therefore, 100 is assigned (cf. Line 8) directly. The end states of a lifecycle process are silent, i.e., they have no associated actions and steps respectively. Regarding the running example (cf. Fig. 2), path ( *0, 1, 3, 4, 7 ,8* ) is taken. For the latter, the resulting interval is shown in Table III.

| State | Progress in Percent |
|---|---|
| 0 | [ 0, 13.3 ) |
| 1 | [ 13.3, 46.7 ) |
| 3 | [ 46.7, 73.3 ) |
| 4 | [ 73.3, 86.7 ) |
| 7 | [ 86.7, 100 ) |
| 8 | 100 |

TABLE III: Progress distribution of path ( *0, 1, 3, 4, 7 ,8* ) from the running example.

The progress interval of the current state can be refined with Algorithm 4, which is invoked by Algorithm 3 (cf. Line 24). Algorithm 4 calculates the current progress until the next state becomes activated (cf. Lines 3-5). Remember that every state of a lifecycle process is refined by its steps (cf. Fig. 3), based on which user form for processing the respective state is automatically created. In this context, steps with marking *"Assigned"* the corresponding attribute has already set with the created user form. To determine the progress of the current state itself, first, the number of the steps, (i.e. steps marked as *"Assigned"*) executed in the current state is computed. Second, the current progress is determined within the provided state interval (difference between maximum and minimum, cf. Line 7) in relation between assigned steps and total steps of a state.

In principle, the described Kalman Filter approach can be adopted to activity-centric business processes as well. In this case, amongst others, loops must be addressed as well. The functionality of a loop in activity-centric BPM is not comparable with a backward transition in object-aware BPM. In the latter, the input data of a previous state remain available in the case of a backward jump. An actual repetitions as in a loop in the activity-centric BPM is not possible. Moreover, an activity-centric process should have one start event and at least one end event to meet the preconditions of the introduced algorithms. Finally, no weights exist in activity-centric business processes. Therefore, either to all activities and events the same weight of one should be assigned or a specific solution for introducing weights in activity-centric business processes needs to be developed.

**Algorithm 4:** refineProgress($min, max, \sigma_A$)

```
Input: Possible progress interval (min, max) of an
       active state and the active state itself.
Output: Returned maximum of progress interval.
        Output current progress.
1  begin
2      do
3          if $\sigma^I.next$ = $Activated$ then  ▷ End State
4              output $max$
5              break
6          else
7              $progress \leftarrow$
                 $min + (max - min) * \frac{|\Gamma^I.assigned|}{|\Gamma^I|}$
8              output $progress$
9
10     while $process$ run
```

## V. Evaluation

### A. Design of the evaluation

In the evaluation, first of all, the improvement of the Kalman Filter estimation compared to the initial progress determination, which always uses the highest weighted path, shall be demonstrated. Second, Kalman Filter estimations with and without the most frequently used (MFU) path are evaluated and compared with each other. The overall aim of the evaluation is to identify the most suitable approach for determining progress.

We have implemented the object-aware process management paradigm in the PHILharmonicFlows tool [1], [16]. The latter enables the modeling of object-aware business processes, including their relational process structure, lifecycles and coordination processes. Additionally, the tool provides an execution engine that controls lifecycle processes and coordination their interactions at run-time. To integrate the Kalman Filter approach with PHILharmonicFlows an extension using the JSON-based process export of PHILharmonicFlows was implemented. The export contains all information needed to reconstruct a business process and its components. This includes information on lifecycle process. For each lifecycle process, its states as well as their weights and successor states are stored. This information is supplemented by the presented priorities and the most frequently used paths (cf. Section III). Algorithms 2-4 as well as the Kalman Filter function have been implemented and integrated with the tool.

In detail, the evaluation of the progress estimation with the Kalman Filter comprises two parts, i.e., a qualitative and a quantitative study:

1. The qualitative study evaluates two paths of the lifecycle process from the running example (cf. Fig. 2). For each state on these paths, the estimated progress is calculated and compared to the real progress.
2. The quantitative study evaluates the application of the presented approach to five large lifecycles with 44 different execution paths in total. This results in 250 progress estimations[1]. Each estimation is calculated with and without the weight of the most frequently used path. The results are then compared with the real progress as well as the highest possible weight used in the first implementation of progress determination.

To visualise the calculated progress we propose diagrams as the one depicted in Fig.: 6. The y-axis shows the progress that may range between 0 and 100 percent. The x-axis shows all states on the evaluated path. Concerning the diagram from Fig. 6, path ( *0, 1, 3, 4, 7 ,8* ) was taken. For this evaluation, only the Kalman Filter estimations are analysed. Given an active state with a given progress interval and the proportion of assigned and unassigned steps, determining the progress in processing this state is trivial. Therefore, we focus on determining the possible progress at the transition between two states. For each state, the corresponding progress values always refer to the state after its complete processing. Consider Fig. 5. For each state, its progress values (estimation with and without Kalman Filter as well as real progress) are displayed. The span coloured in black represents the progress range from the path with the lowest weight (maximum progress) to the one with the highest weight (minimum progress). The red line, in turn, indicates the real progress which can not be precalculated, but only becomes known after having actually executed the path. Note that this value is created to compare Kalman Filter values with the actual progress. The green line shows the progress estimated with the Kalman Filter without the weight of the most frequently used path (MFU). This corresponds the progress calculation, if no data on the most frequently path exits. Finally, the blue line represents the progress estimated with the Kalman Filter with the weight of the path most frequently used (MFU). This corresponds to the progress calculation after multiple runs, i.e., when having knowledge of the weight of the most frequently used (MFU) path.

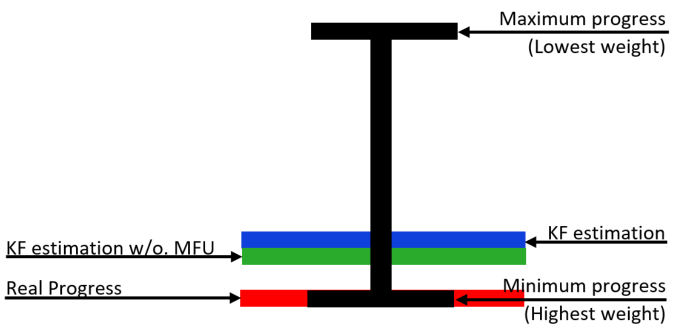

Fig. 5: Legend of evaluation diagram.

[1]The 250 progress estimation as well as the 5 lifecycles are available via the following link to Researchgate: https://www.researchgate.net/project/Progress-determination

## B. Evaluation of the Kalman Filter

**Qualitative Study**. We use the longest path to determine progress, as in this case no backward jumps of the progress occur. Consider, for example, Fig. 7; when using the shortest path to calculate the current progress of a lifecycle process, the course of the progress would result in a backward jump between State 1 and 3 (see black maximum lines).

When using the Kalman Filter, backward jumps in the progress scale might occur in principle. However, for all considered lifecycles and their various paths, this was not the case. Backward jumps occur, when the blue line (or green line when using the estimation without the most frequently used path) of a state is lower than the one of the previous state.

In the first evaluation step we consider for the progress estimation for the running example (cf. Fig. 2 in more detail. Here, five different execution paths are possible as summarised in Table IV.

| Path ID | Path Sequence |
|---|---|
| 1 | 0 - 1 - 2 |
| 2 | 0 - 1 - 3 - 4 - 5 - 6 - 7 - 8 |
| 3 | 0 - 1 - 2 - 3 - 4 - 7 - 8 |
| 4 | 0 - 3 - 4 - 5 - 6 - 7 - 8 |
| 5 | 0 - 3 - 4 - 7 - 8 |

TABLE IV: Possible paths from start state to an end state

In the following, two paths are analysed in more detail. Fig. 6 shows the progress comparison of the estimated and real progress for Path 3. As can be seen, the progress estimation for State 1 is very close to the real progress. Comparing the different values of State 0, the progress presuming the path with the highest weight taken is 9.52% and progress presuming path with the lowest weight taken is 28.57%. This results in the progress difference of 19.05%.

In relation to this difference, the deviation of the real progress (13.33%) from the one estimated Kalman Filter (w/o. MFU) (13.64%) is only 0.34%. The approach using the path with the highest weight for estimation deviates 3.81% from the real progress, which is more than ten times worse.

Note that the different values of State 1 result in a very large range of 66.67% with the progress between 33.33% and 100%. This can be explained with an end state being a potential successor. The real progress is 46.67%. Both estimations (w. and w/o. MFU) deviating similar degree from the real progress (KF w. MFU 3.34%, KF w/o. MFU 3.81%). Considering the large range, both estimations are very close to the real progress. After State 3, the path with the lowest remaining weight is executed. This path reflects the worse case scenario of the approach using the highest weighted path for prediction resulting in a deviation of the prediction of more than 20% to the real progress.

In comparison, Kalman Filter (w. MFU) achieves an improvement of $8,73\%$ after State 3 and 10.33% after State 4. The improvement of Kalman Filter (w/o. MFU) is 2.72% better after State 3 and 9.34% better after State 4. After State 7 only one transition to an end state remains. For this reason, all calculated progress estimations result in 100%.

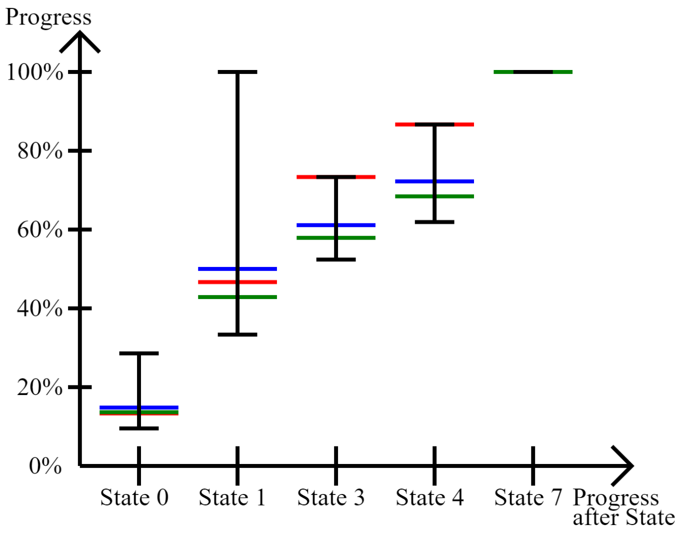

Fig. 6: Evaluating Path 3 of the lifecycle from Fig. 2.

Fig. 7 shows another example for evaluating the Kalman Filter approach. Here, the path with the highest weight of the lifecycle process (i.e. Path 2) is used. This is the best-case scenario for the first implementation, which is always used the highest weighted path for progress determination. For this reason, this path is analysed in more detail.

Table V shows the numerical difference between the estimations provided by the Kalman Filter (w. MFU and w/o. MFU) in relation to the real progress. Regarding, the path with the highest weight, the estimation without the most frequently path is significantly better (improvement of 7.15% for State 1). In this case, the input of the most frequently used path is not executed. For this reason, the estimation of Kalman Filter (w/o. MFU) is, on average, 3.84% more accurate than Kalman Filter (w. MFU). After the decision in State 4, only path ( *5, 6, 7 ,8* ) is left over. Therefore, all progress determinations are the same.

| Error Calculation | State | | | | ∅ |
|---|---|---|---|---|---|
| | 0 | 1 | 3 | 4 | |
| \|w/o. MFU - real progress\| in % | 4.12 | 9.53 | 5.52 | 6.52 | 6.42 |
| \|w. MFU - real progress\| in % | 5.3 | 16.68 | 8.73 | 10.33 | 10.26 |

TABLE V: Difference between the progress estimations obtained with Kalman Filter and the real progress (using Path 2 of the example of Fig. 2).

**Quantitative Study** we considered five lifecycle processes. These lifecycles are larger than most of the lifecycles from practice. Note that the larger the lifecycle process, the less accurate an estimation might be due to the potentially large differences in the weights of the different paths. In total, these five lifecycles have 44 different paths. In the following, the progress of each state on all 44 paths is evaluated. This results

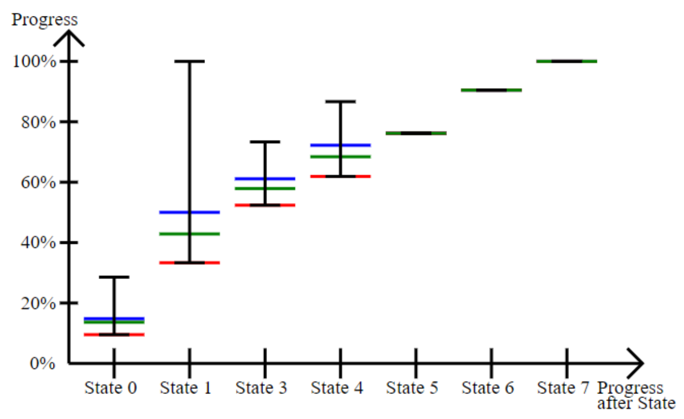

Fig. 7: Evaluation of path 2 of running lifecycle example from Fig. 2.

in a total number of 250 states among all five lifecycles. For each of them, the estimation of Kalman Filter with and without the most frequently used path is calculated. In addition, the result of the highest weighted path for progress determination is indicated. In the boxplot depicted in Fig. 8 the difference between the real progress, Kalman Filter with and without the most frequently used path, and the highest weighted path is visualised. The divergence between the real progress and the two Kalman Filter estimations as well as the divergence between the real progress and the highest weighted path are used to evaluate the approach.

First, the divergences of the two Kalman Filter progress estimations to the real progress are very similar. Consequently, the most frequently used path does not result in a significant improvement. In fact, Kalman Filter (w/o. MFU) is on average even better (0.24%) than the one with MFU. Thus, the most frequently used path need not be considered as an input for Kalman Filter. In particular, the progress calculation can be applied immediately after model creation without need for event log data. The average difference between the real progress and the progress estimation using the highest weighted path (5.69%) is significantly worse than the average difference between the real progress and the progress estimation of Kalman Filter (w/o. MFU) (7.46%). Additionally, the outlier points are significantly bigger (the bigger, the less accurate the estimate) and occur more frequent for the highest weighted path than for Kalman Filter estimation. The average error rate (i.e., the difference between the real progress and the progress estimated one using the highest weighted path) for any path not being the highest weighed path itself is $14,36\%$. This error is almost three times as large as in the context of the Kalman Filter (w/o. MFU) estimation. On average, Kalman Filter estimation deviates about $5\%$ (above or below) from the real progress. Considering that no log data is utilised, this error rate of $5\%$ constitutes a suitable first orientation towards progress, especially for newly deployed business processes estimation, which no log data is available at the beginning. If log data is collected over several runs of the process, other approaches (e.g. predictive monitoring [5]) can be further used to optimise this error rate.

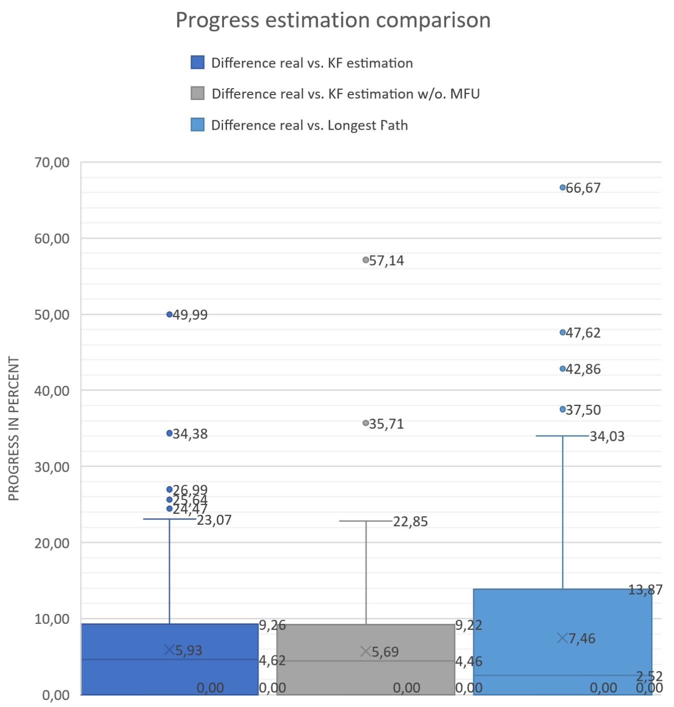

Fig. 8: Comparison of the 250 progress estimates in a boxplot with outlier points.

## VI. Related Work

Related to our work is the field of predictive process monitoring [4], [14], [20], [21]. According to [5], predictive monitoring approaches can be categorised into three different types. First, numerical prediction is applied to predict the remaining time of an ongoing execution [21] or its cost [20]. Second categorical predictions on the risk class of an ongoing process are supported be existing approaches [4]. Third, the sequence of the still remaining activities (their payload) can be predicted [14]. However, these approaches presume the existence of an event logs. For newly generated processes without any event log, as well as predictions are therefore not possible.

Real-time process monitoring, especially progress determination, is not well-supported in object-aware or object-centric approaches to BPM. A dashboard-based approach for offline (i.e. non-real-time) monitoring in object-aware BPM is described in [3]. Another approach to measure the progress of activity-centric business models is introduced in [11]. The latter included two measurement techniques for reference model-based business process modeling. For this purpose, both techniques determine the progress based on the compliance of activity labels [11]. Moreover, an approach for monitoring processes and prediction their progress with data state transition events is presented in [8]. In [9], progress in activity-centric processes is improved by utilising object state transition as described in [8].

In the research area of business processes monitoring, the Kalman Filter has not been applied yet. However, Kalman Filter can be found in many other disciplines to predict states. One of its most famous applications is the moon landing [13]. Even today, Kalman Filter is used extensively in state estimation of linear and non-linear systems such as in the inertial navigation system of aeroplanes [13], in the tracking method of autonomous vehicles [13], in early failure prediction [22], and many more applications [10], [13].

## VII. Summery and Outlook

We presented an approach for determining the progress of object lifecycle processes as known from data- or artifact-centric business processes. The challenge of determining progress in the given context is to estimate the weight of the still to be executed path of the lifecycle process originating from its current state to one of the end states.

To tackle this challenge, an existing estimation method, i.e., one-dimensional Kalman Filter, is used to predict the weight of this remaining path. As a major benefit of our approach, real-time progress determination becomes feasible already with the first runs of lifecycle process. As opposed to known predictive monitoring approaches, no log data is required. For large and complex lifecycle processes, the deviation of the determined median and average progress differs about $5\%$ from the real progress. Our quantitative study has also shown that the use of the weight of the most frequently used path results in slightly inferior progress prediction than without considering this most frequently used path. The latter case the estimation of Kalman Filter and the resulting progress are completely independent of previous executions of the business process.

In future work, we will consider progress determination multiple, interacting (i.e., interrelated) lifecycle processes. We want to investigate how a coordination process affects the progress of the overall (object-aware) business process [2]. Furthermore, the inaccuracy of about $5\%$ on average can be improved. Considering that no log data is used at all, the error rate of $5\%$ of Kalman Filter (w/o. MFU) constitutes a suitable first orientation towards progress determination, especially for newly deployed business processes for which no expressive log data is available at the beginning. Furthermore, backward transitions within a lifecycle process have not been considered in this paper. Jumping back to previous state can be interpreted in two ways. First, jumping back might decease current progress (as parts of the lifecycle process may have to be repeated) or, second, the progress remains the same as the data has already been set. Both options will be evaluated in an empirical study.

In principle, the basic idea of using Kalman Filter for progress determination is not restricted to object-aware business process, i.e., in activity-centric business processes this basic approach to calculate progress and to predict the remaining path can be applied as well. However, the robustness and accuracy of other modelling paradigms need to be investigated separately, as the structure and size of a single lifecycle in an object-aware business process is usually smaller than an activity-centric business process.

## Acknowledgment

This work is part of the ZAFH intralogistic project, funded by the European Regional Development Fund and the Ministry of Science, Research and Arts of Baden-Württemberg, Germany (F.No. 32-7545.24-17/12/1).

## References

[1] K. Andrews, S. Steinau, and M. Reichert, “A tool for supporting ad-hoc changes to object-aware processes,” in *2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW)*. IEEE, 2018, pp. 220–223.

[2] L. Arnold, M. Breitmayer, and M. Reichert, “Towards real-time progress determination of object-aware business processes,” *Central European Workshop on Services and their Composition*, pp. 14–18, 2021.

[3] M. Breitmayer, L. Arnold, and M. Reichert, “A dashboard-based approach for monitoring object-aware processes,” *Central European Workshop on Services and their Composition*, pp. 29–33, 2021.

[4] R. Conforti, S. Fink, J. Manderscheid, and M. Röglinger, “Prism–a predictive risk monitoring approach for business processes,” in *Int. Conf. on Business Process Management*, 2016, pp. 383–400.

[5] C. Di Francescomarino, C. Ghidini, F. M. Maggi, and F. Milani, “Predictive process monitoring methods: Which one suits me best?” in *Int. Conf. on Business Process Management*, 2018, pp. 462–479.

[6] G. Galanis and M. Anadranistakis, “A one-dimensional kalman filter for the correction of near surface temperature forecasts,” *Meteorological Applications*, vol. 9, no. 4, pp. 437–441, 2002.

[7] A. Gelb, *Applied optimal estimation*. MIT press, 1974.

[8] N. Herzberg and A. Meyer, “Improving process monitoring and progress prediction with data state transition events,” *Central European Workshop on Services and their Composition*, 2013.

[9] N. Herzberg, A. Meyer, and M. Weske, “Improving business process intelligence by observing object state transitions,” *Data & Knowledge Engineering*, vol. 98, pp. 144–164, 2015.

[10] Z. Khan, H. Bugti, and A. S. Bugti, “Single dimensional generalized kalman filter,” in *Int.Conf. on Computing., Electronic and Electrical Engineering*. IEEE, 2018, pp. 1–5.

[11] A. Koschmider, J. L. d. l. Vara, and J. Sánchez, “Measuring the progress of reference model-based business process modeling,” in *INFORMATIK 2010*, pp. 218–229.

[12] D. J. Lasser, “Topological ordering of a list of randomly-numbered elements of a network,” *Communications of the ACM*, vol. 4, no. 4, pp. 167–168, 1961.

[13] R. Marchthaler and S. Dingler, *Kalman-Filter*. Springer, 2017.

[14] M. Polato, A. Sperduti, A. Burattin, and M. de Leoni, “Time and activity sequence prediction of business process instances,” *Computing*, vol. 100, no. 9, pp. 1005–1031, 2018.

[15] S. Steinau, K. Andrews, and M. Reichert, “Executing lifecycle processes in object-aware process management,” in *Int. Symp. on Data-Driven Process Discovery and Analysis*. Springer, 2017, pp. 25–44.

[16] ——, “A modeling tool for philharmonicflows objects and lifecycle processes,” *BPM Demo Session*, 2017.

[17] ——, “The relational process structure,” in *Int. Conf. on Advanced Information Systems Engineering*. Springer, 2018, pp. 53–67.

[18] ——, “Enacting coordination processes,” *arXiv preprint arXiv:2012.08409*, 2020.

[19] S. Steinau, V. Künzle, K. Andrews, and M. Reichert, “Coordinating business processes using semantic relationships,” in *Conf. on Business Informatics*, vol. 1. IEEE, 2017, pp. 33–42.

[20] T. B. H. Tu and M. Song, “Analysis and prediction cost of manufacturing process based on process mining,” in *Int. Conf. on Industrial Engineering, Management Science and Application)*. IEEE, 2016, pp. 1–5.

[21] W. van der Aalst, M. H. Schonenberg, and M. Song, “Time prediction based on process mining,” *Information systems*, vol. 36, no. 2, pp. 450–475, 2011.

[22] S. Yang and T. Liu, “State estimation for predictive maintenance using kalman filter,” *Reliability Engineering & System Safety*, vol. 66, no. 1, pp. 29–39, 1999.