Event-driven tracking of important sections of business processes

Christoph Heinz
Software AG
Darmstadt, Germany
christoph.heinz@softwareag.com

Abstract: Business processes are crucial in today's companies. They define for the different departments of a company the execution order of tasks as well as their interaction. Typically specific sections within a business process are essential for its successful completion and therefore deserve special attention. Such a section is typically associated with a completion constraint, e.g., a credit application has to be processed within 24 hours in order to meet SLAs agreed with the customer. The business user is highly interested in a real-time monitoring of those sections and constraints as it allows him to immediately tackle potential problems during process execution. We present a novel approach for tracking important sections of a business process using Complex Event Processing coupled with an Event-Driven Architecture. While the business process is executed, we continuously monitor its processing status to determine whether such a section is completed and its associated constraints have been fulfilled or violated. To allow for a more proactive reaction of the business user, we also present an early warning system which raises an alert if a not yet completed section runs the risk of violating a constraint.

1 Introduction

In modern companies business processes play a vital role as they clearly define how tasks in daily business are connected to each other and how they are processed. Business processes are used in virtually every industry and each department of a company. For example, there are processes that define the tasks for an approval of an insurance claim, an invitation of a job applicant for an interview, the shipment of an order to the customer, the authorization of a business travel. Each of those business processes consists of several tasks which are connected to each other, thereby modeling the basic sequence of steps from start to end of the process.

A business process can be defined in terms of a graphical model consisting of symbols and connections in between them, using for example the well-established BPMN modeling standard. The latter also defines how a model can be transferred into a process execution language. There are also other approaches available for defining an execution of business processes. Thus, business processes are available as models and transformed into executable instances. The execution of those process instances can be monitored in order to inform the business user about the current status of business processes he is interested in.

In the area of business process monitoring tools Software AG has recently introduced the notion of stages and associated milestones. A milestone is defined as one particular point in a process occurring at a specific point in time. For example, an order has been received at 15:25 or an email has been sent at 10:13. A stage is defined by a start and an end milestone and a condition how these two milestones relate to each other. Most prominent type of such stages is a temporal stage. The corresponding temporal condition demands that the timespan between start and end milestone has to be either greater or less than a predefined timespan. An exemplary stage could define that the timespan between receiving an order, which denotes the start milestone, and shipping the goods, which denotes the end milestone, has to be less than 12 hours. Thus, a stage allows the user to formally define the most important sections of a business process and corresponding completion criteria.

In this work we tackle a real-time tracking of those stages. Prerequisite is that the IT component executing the process instances emits status information in case of process instance changes, e.g., the order step has been started at 5:00. By evaluating this process status information in a continuous fashion, a corresponding process stage tracking component can directly inform the user that a stage has been started, breached, or completed. Using real-time processing capabilities, this stage information can be derived with minimum latency, e.g., the user is notified immediately if a stage has been breached. This enables the user to monitor crucial sections of his business processes.

We introduce the notion of process stage tracking with a focus on the proactive handling of stage violations. As described before the user is notified when a stage finally has breached or completed. A further extension is to provide early warnings when a not yet completed stage potentially might be violated.

The following enumeration sketches core problems a process stage tracking component has to deal with and our proposed solutions:

Detection of stages at risk: Recent process tracking capabilities allow to send notifications of stages being started, breached or completed in a real-time manner. Thus, the user knows immediately when a process stage has breached for instance. He can react to this situation and trigger consecutive actions. However, the user can no more take corrective actions to proactively avoid that a stage breaches in the near future. Therefore, the proposal is to provide an early warning to the user that a successful stage completion is at risk. While the process is executing and the current execution of a process instance is in between start and end milestone of a stage, the probability that the remaining steps until the end milestone take longer than the remaining time to stage completion is determined. If that probability is too high, the corresponding stage is classified as being at risk and an alert is sent.

Definition of potential stage violations: The detection of potential stage violations requires to estimate the remaining time from the current position in the process execution to the end milestone. For that purpose each step in the process has an associated statistical model which models the timespan this step typically takes. These models are continuously updated by incorporating recently measured step cycle times. By combining these step cycle models we estimate the remaining time to the end milestone. These estimates provide the user means to assess a potential violation of the stage.

Real-time detection of potential violations: In order to ensure a maximum transparency for the user potential stage violations are computed in a real-time manner. Provided that the process execution engine reports process instance changes directly, Complex Event Processing technology coupled with an Event-Driven Architecture [EN10], [SC09] can be leveraged to determine stage violations in real-time. Each time a process instance change occurs, the probability for violating the process stage is directly computed. Such a result can then be forwarded to the user. For example, a corresponding visual cockpit can highlight for a selected process model the corresponding stages being at risk. Additionally, an email can be sent or other follow-up actions can be triggered.

Software AG offers a platform for Intelligent Business Operations (IBO) [Ope14] which already includes parts of the proposed stage tracking functionality for business processes.

The paper is organized as follows. Section 2 discusses related work. Section 3 introduces stages and milestones for business processes. Section 4 gives an overview of our solution, while Section 5 presents its implementation. Section 6 addresses future extensions of our approach. Finally, we conclude with a summary in Section 7.

2 Related Work

Some IT vendors offer BPMS functionality that relates to some extent to the notion of milestones and stages for business processes.

With Business Monitor IBM provides a tool to monitor business process instances, including their notion of milestones. A milestone is a collection of so-called monitoring contexts. These milestones are used as single steps in a virtual process diagram. The user can specify under which condition a milestone gets a different fill color or labeling in a corresponding visual cockpit.

Oracle Business Process Management uses the notion of Guided Business Processes to organize process activities in milestones. This approach concentrates on interactive activities in a business process. The notion of milestones is focused on business processes involving human interaction. Milestones can be equipped with expiration dates or times and subsequent actions are triggered when milestones are breached.

The Appian BPM Suite for Designers provides capabilities for Intelligent BPMS Analytics as well as Complex Event Processing. Intelligent BPMS Analytics combines particularly BAM and Predictive Analytics. Business processes can be monitored using real-time performance metrics. With the help of predictive analytics this real-time information can be used to predict the process behavior.

SAP also provides capabilities to monitor milestones in a process context. The monitoring process can use rules to decide whether a relevant exception situation has occurred.

[MFE12] discusses CEP- and statistics-based monitoring for the transport and logistics domain, more precisely the standardized Cargo 2000 process. While this process contains fixed milestones and seems to be linear, our approach works for arbitrary business processes from arbitrary domains. Additionally we support more complex processes in-

cluding branches and iterations etc. as well as arbitrary process milestones.

[BDR12] addresses prediction and root cause analysis of events against the background of Business Activity Monitoring. Key performance indicators (KPI) and associated rules are used for the analysis. Predictions are created for rules associated with KPIs. When the associated prediction engine has finished a training phase, the predictions can be used to forecast rule violations and provide warnings. This work concentrates on KPIs and rules, while we address process stages and milestones. While databases for maintaining relevant data as well as polling intervals for the evaluation of the current status are used, we use Complex Event Processing technology in order to support real-time processing of large volumes of process data.

[CK07] proposes a model-driven approach to enhance already present business performance models with predictive modeling. So-called metamodels describe languages which can be used to express models. These metamodels are extended by a notion of time in order to allow for the prediction of metrics. Further trigger conditions are provided that define when and how the predictive analytics tasks are executed. The tasks themselves are provided as a service of a dedicated component, which gets the relevant data from a data warehouse and also persists the forecasted metrics into the warehouse. This work is a conceptual approach describing in a model-driven fashion how monitoring metrics can be extended towards forecasting and how the resulting models are generated and processed. It does not discuss notions of process stages and milestones as well as of an associated real-time monitoring based on Complex Event Processing.

[CFLZ13] discusses prediction of metrics based on events. Within a training phase selected events are stored in an intermediate storage. Then they are grouped and aggregated with summary statistics. These summary statistics are used to detect patterns within the events. A function selector then uses this information to select a suitable prediction model which is finally used to predict metrics. This approach delivers a framework for un-supervised prediction using events. The specific problem of monitoring stages and milestones of a business process and raising early warnings in case of potential stage violations is not addressed.

[BBH+14] addresses the monitoring of business processes with Complex Event Processing. A general framework is presented that describes how a business process environment can be extended by a Complex Event Processing engine so that business processes can be monitored. The focus of this paper is the discussion of the different components of such a framework and the basic interaction patterns.

[WZM+11] discusses an event-based monitoring of process execution violations. While we focus on violations of stages, these execution violations refer to deviations from the control-flow during execution of a process, i.e., is the current execution sequence compliant with the associated normative process model.

In comparison to these related approaches, our novel approach combines the following benefits: Our framework provides the business user means to monitor business processes in a very fine-granular manner as it allows him to concentrate on the crucial parts of the process and define meaningful monitoring criteria for these parts. This is encapsulated within the concept of stages and milestones being tracked. Additionally the user is alerted

proactively in case the process parts being monitored run the risk of failing the specified criteria. In order to provide maximum transparency to the user, the framework is based on EDA and CEP principles. This approach guarantees instantaneous results as well as scalability for the case of massive data volumes.

3 Preliminaries

In the subsequent discussions we use the following running example: An insurance company uses business processes to handle the processing of incoming insurance claims. The execution of such a process consists of automated steps, like sending the client a rejection or triggering the loss payment, as well as of manual steps requiring human interaction, like reviewing material documenting the damage or approving a payment estimate. Similar process steps can also be found in many other domains, e.g. handling calls in a call center or processing credit applications.

Such business processes are to be monitored by tracking associated milestones/stages. For this business process a process model is available that formally defines the tasks and their execution orders. The business process is executed by one or more process instances in a process execution environment. Within that execution environment sensors are used to monitor the process instances so that changes of a process instance are captured and directly populated to follow-up consumers for further analysis. For that purpose an Event-Driven Architecture (EDA) can be leveraged. Each time a process instance change takes place, a corresponding process instance change event is built and published. Such an event carries the information that a process instance change has taken place at a given point in time, e.g., the task 'Approve claim' of process instance 4711 has been completed at 14:23. Thus, during process execution all running process instances continuously emit information about their latest changes. For publishing and consuming those events the Event Bus within the EDA is used. That Event Bus allows multiple participants to publish events on event channels and also consume events from event channels they have subscribed to.

The process instance change events can be used to monitor running process instances by evaluating the status of stage and milestone completion. A milestone refers to one specific position within a process, e.g., the task 'Review damage documentation' has been started. A stage consists of a start and an end milestone and a stage condition. The stage defines that the steps between completion of start and end milestone have to meet the stage condition. A common class of stage conditions is of temporal nature. For example, the timespan between start of the task 'Review damage documentation' and start of the task 'Request additional information from client' has to be less than one day. Other stage conditions refer for example to the number of iterations of process steps.

Using the aforementioned process instance change events, which are continuously emitted, the stages can be continuously tracked. Such a tracking requires sending, analyzing, and receiving of corresponding process events in a real-time fashion.

4 Overview of Stage Tracking Framework

This overview introduces a framework for tracking stages of running business process instances, followed by a detailed discussion of the implementation in the next section. The framework comprises the following layers and components:

Stage configuration layer: On that layer the user can configure process stages for a given business process model. This includes defining the start and end milestone of the stage as well as the associated stage condition. The user can also define whether potential stage violations are to be checked. If that functionality is enabled, the user can optionally provide input for the initial setup of the statistical models used to determine stage violations. Once the stage plus its monitoring options has been defined, it can be uploaded for execution, i.e., the stage is actively monitored for currently running process instances.

Process data acquisition layer: On that layer the relevant data for evaluating stage completion status as well as potential stage violations is acquired. The process engine executing the process instances is equipped with sensors. These sensors trigger a process instance change event each time a process instance changes its status. These streams of process runtime events are directly forwarded to subsequent consumers using the EDA.

Stage analysis layer: That layer hosts the main logic to determine stage completion status and potential stage violations. While process instance change events are streaming in, the completion of stages is continuously monitored using a Complex Event Processing engine. For each incoming event all stages defined for the associated process model are determined. For each of those stages it can be derived when the milestones of a stage have been completed. That information is then used to check whether the stage condition is fulfilled. Depending on the result corresponding result events are published that indicate whether the stage has been breached or completed. Additionally the stage analysis layer determines potential stage violations. For that purpose it maintains for each process model statistical models of the process steps involved. Such a statistical model models the cycle time of a process step based on knowledge from previous and current process instance executions. Additionally the analysis layer is connected to the process data acquisition layer and thus receives the streams of process instance change events in a real-time fashion. Each time a new process instance change event arrives, the statistical model of the associated process step is updated. Given the current step of the process and the process model, all potential paths from that step to the end milestone are determined. For each of those paths the probability that the time from current step to end milestone takes longer than the remaining time to expected stage completion is determined. For the computation of that probability the statistical models are exploited. Then the computed probability for potential stage violations is forwarded as result event for further consumption.

Result processing layer: The result processing layer continuously receives events from the stage analysis layer describing completion status of stages as well as potential stage violations. These events include details like the process instance id, the stage id, and the violation probability. This information is combined with the process model and process execution in a visual cockpit so that the user can quickly assess the impact of stage completions and potential violations on his current business.

The following figure 1 shows an example stage that has been successfully completed. Evaluating the completion of the start milestone and the completion of the end milestone reveals that the stage condition has been met and thus that the stage has been successfully completed for that process instance.

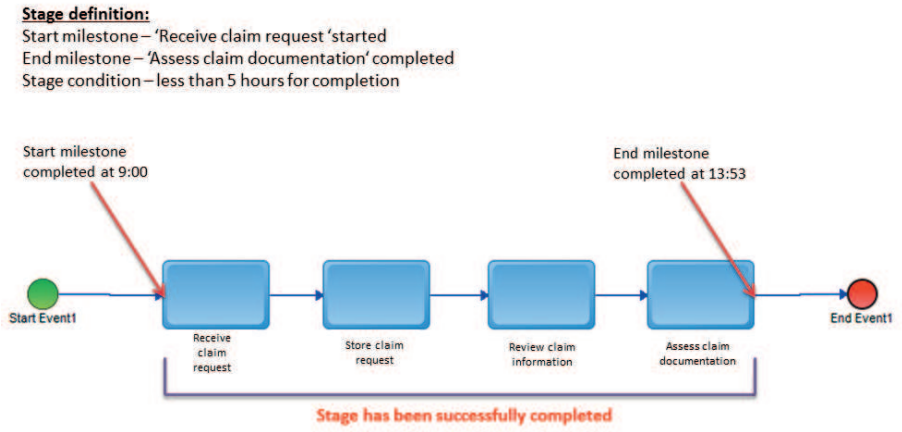

Figure 1: Example for successful completion of a stage

The following figure 2 illustrates the main idea of stage violation monitoring with an example. Using the estimated time till completion of the end milestone and comparing it to the time left for successful stage completion reveals that in this case the risk for a stage violation is very high.

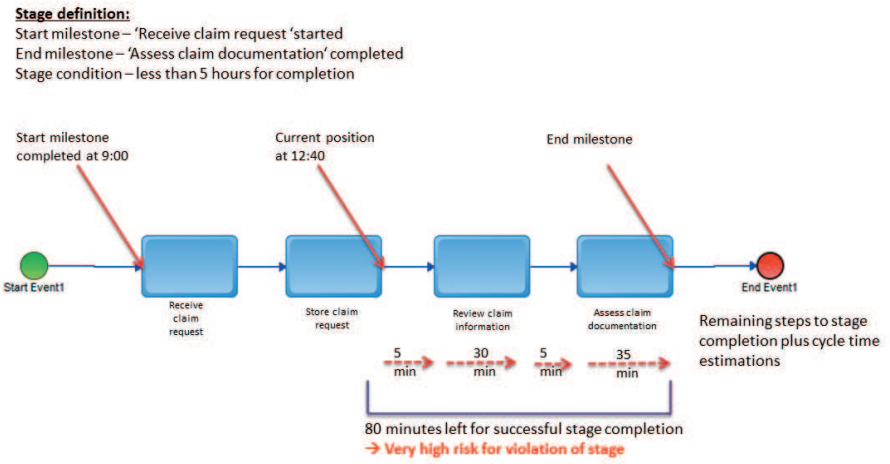

Figure 2: Example for potential violation of a stage

5 Implementation of Stage Tracking Framework

Based on the previous overview this section discusses how the corresponding layers can be implemented. Software AG offers in its platform for Intelligent Business Operations already the tracking of stages and milestones.

5.1 Stage Configuration Layer

The user utilizes the stage configuration layer to define and configure process stages for processes. As already described the user can select a process model and define start and end milestone of a stage as well as the associated stage condition. Next he can activate the monitoring of potential stage violations. This monitoring component relies on statistical models of the process steps, which describe their cycle times. More precisely, such a statistical model consists of a probability density function which describes the distribution of the cycle time. These models can either be learned automatically using process runtime data or can be specified by the user, thereby leveraging the expert knowledge of the user. For the latter option the configuration layer offers different parameterized statistical models. The user selects the model that fits best and then specifies the corresponding parameters of that model. Examples for such parameterized models are:

- Discrete single point distribution: Only one value has to be provided, which has a probability of 100% to occur.
- Discrete multiple point distribution: A set of values has to be provided, each value having assigned a probability. A special case is the discrete uniform distribution where each of the possible values has the same probability.
- Continuous uniform distribution: This distribution consists of two values, which are minimum and maximum. Between minimum and maximum the probability is uniform and 0 otherwise.
- Normal distribution: The normal distribution is identified by mean and variance.

If the user enables automatic learning of the statistical models, the models are learned using process runtime data. If the process instances have been freshly started, these models are not yet available. Therefore the user can either select to wait or to specify and use intermediate models until the learned models are available. For these intermediate models, again, the user can select from a list of parameterized models and set the corresponding parameter values.

Additionally the user can define different visualization options and follow-up actions depending on the severity of the stage violation. For example, if the probability is above 90%, the corresponding visual cockpit highlights that stage, e.g., by filling it with color red, and additionally an email is sent or a follow-up process is triggered. If the probability is below 30%, the stage is marked as green.

As a final step the process models, the defined stages as well as the configured monitoring options are uploaded for execution.

In the Software AG suite the Process Designer is used to specify process models and stages/milestones. Within that modeling tool the user can build and upload a process model for execution once the modeling and configuration of the model has been completed. During that upload step the Process Engine transforms the incoming process models into executable processes. Additionally the stage definitions are written to a database.

Figure 3 summarizes the main steps for the configuration of stage violation monitoring.

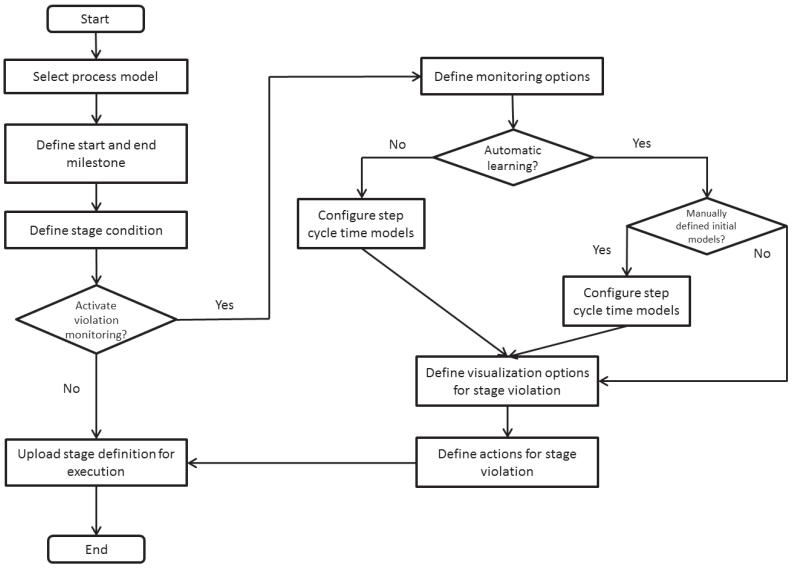

Figure 3: Main steps for configuration of stages

5.2 Process Data Acquisition Layer

Process Engine executes the process instances for uploaded process models. In the execution environment of these process instances internal triggers are registered, which listen for process initializations and step transitions. The registration of these internal triggers is done automatically; the user does not need to configure how and where process instance information is acquired during execution. No additional steps are required other than those already done in the process modeling step; Process Engine configures the internal triggers within the model upload and execution step. During runtime Process Engine reports each status change of a process instance and generates an internal document with detail information like process instance id, process model id, step id, point in time when the change took place etc. Each process model has a corresponding trigger associated, which listens to those generated documents with execution details of instances of that model. Therefore

the corresponding trigger executes every time Process Engine reports a status change of a process instance. This approach allows for a fine-granular monitoring of a process model as every transition of all process instances is reported.

After a trigger has received such an internal document describing a process instance change, that information is extracted and encapsulated into a new process instance change event. This event is to be published so that follow-up consumers, e.g., a visual cockpit or the aforementioned process tracking component, can be informed about recent changes of process instances. For that reason Process Engine is EDA-enabled, i.e., it is connected to the Event Bus, which is the general transport layer for arbitrary events in the Event-Driven Architecture. Process Engine can therefore be publisher as well as consumer within the EDA. In this context, Process Engine directly emits the newly built process instance change event to a dedicated channel of the Event Bus. Over that channel all process instance change events are published, i.e., that design also allows to capture information of multiple process engines. A corresponding tracking component subscribes to that channel and gets the events in an event-driven manner. This approach ensures a minimum time between change of a process instance and analysis of that information with respect to affected stages.

The results provided by the stage tracking component are processed in the same manner. Directly when new insights have been derived, a new result event is built and published on the Event Bus so that subsequent consumers like visual cockpits can process that information directly. Figure 4 summarizes the main flow of events.

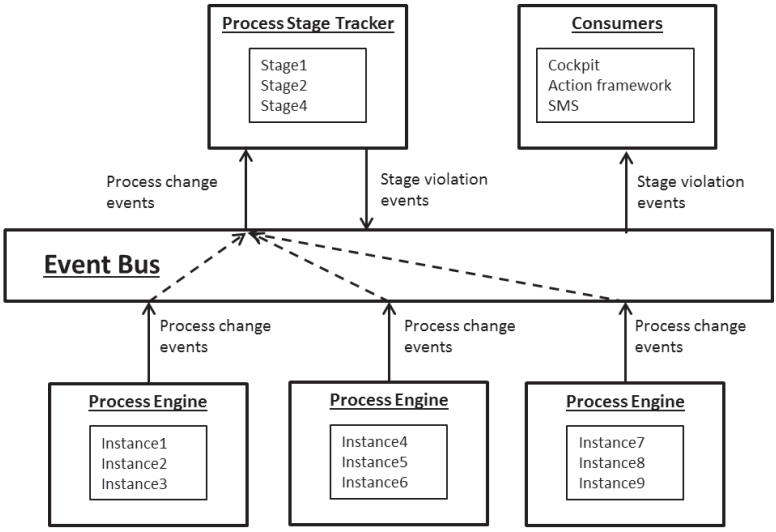

Figure 4: Event flow

Enabling such a monitoring of process instances for a process engine introduces some overhead as process instance change information has to be captured and published as an

event. That overhead will be manageable in most cases as the Event Bus bears the main burden of processing and publishing these events to follow-up consumers and the CEP engine bears the burden of analyzing the events. However, in cases where huge numbers of process instances change with a high frequency, the additional step of generating and sending process instance change events can adversely affect the performance of the process engine. Such a problem can be alleviated by distributing the process execution load over multiple process engines.

5.3 Stage Analysis Layer

The stage analysis layer is responsible for receiving process change events, determining status and potential violations of registered stages, and publishing the results to follow-up consumers. The main components of that layer are summarized in Figure 5.

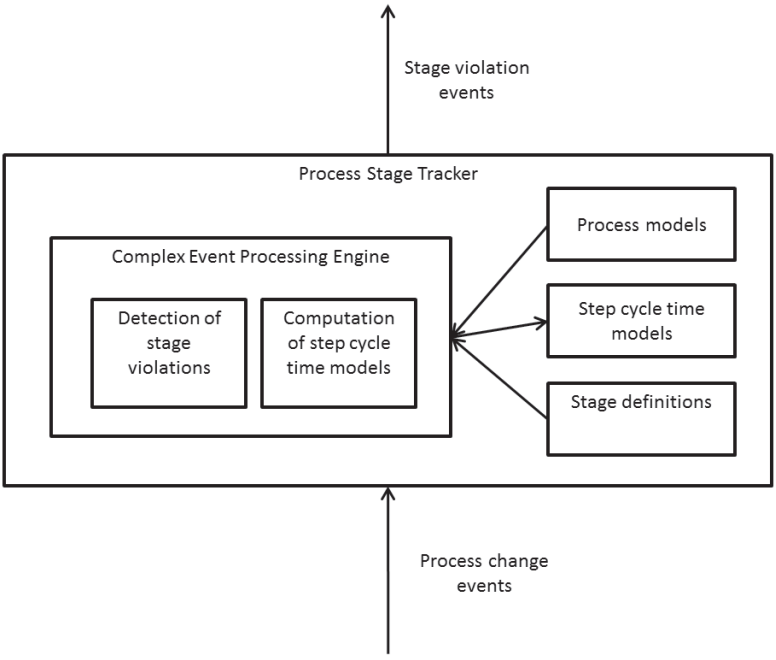

Figure 5: Main components of stage analysis layer

5.3.1 Processing of Incoming Events

The process stage tracker continuously receives process instance change events from one or more process execution engines. A process instance change event denotes for a particular process instance that either a process step has changed or a process transition has taken

place. A process step can change its status from started to completed. Besides started and completed more complex cases are possible like expired, waiting, interrupted etc. In the course of this work we focus on the case of status started changing to status completed. A process transition is the transition from one step being completed to the start of the subsequent step.

For each those incoming events the following actions are executed: First, the event is used to determine stages being registered for the associated process model. More precisely, the event is joined with the database that stores the stage definitions. Second, the stage status is derived. For each of those stages, it is checked whether start or end milestone of that stage is completed. If the latter is the case, the stage condition is checked. If the condition has been met or failed, a result event is published on a dedicated event channel. This functionality is implemented by means of Complex Event Processing queries to ensure a minimum processing latency. For example, a continuous query joins the incoming events against the stages database. The third step is to detect potential stage violations and publish corresponding stage violation events. The process instance change event is utilized to update the statistical model for the associated process step or transition. Combining these statistical models of the process cycle times for the different process steps and transitions, the probability for failing the stage condition can be computed, as will be discussed next.

5.3.2 Statistical Modeling of Cycle Times

Given a process model a statistical model for the cycle time of each step and transition is computed. This cycle time is defined as the elapsed time between start of a step/transition and completion of that step/transition. According to the previous considerations the cycle time can be computed by evaluating the streams of process transition events. For that purpose a continuously running CEP query can be used which joins the start of a process step/transition with its completion. Using the temporal information of these two events the cycle time can be derived. Thus, the query emits a stream of cycle times of steps/transitions of the running process instances.

This information is used to derive a statistical model. The cycle time of a step/transition is modeled as a univariate random variable X. Thus, X has a probability density function f which defines the distribution of the cycle time. This probability density function is unknown. Therefore, the measured cycle times are interpreted as a random sample of independent and identically distributed random variables following X. This $iid.$ sample can then be used to estimate the probability density function of X. The resulting density estimate $\hat{f}_X$ can be utilized to determine the characteristics of X. More specifically, the probability that the cycle time is in a given range $[a, b]$ can be estimated by

$$P(X \in [a, b]) \approx \int_a^b \hat{f}_X dx$$

For instance such a statistical model can be used to answer questions like:

- What is the probability that the cycle time of 'Receive claim request' is between 10 and 15 minutes?

- What is the probability that the transition from 'Store claim request' to 'Review claim information' takes more than 5 hours?
- What is the probability that 'Approve claim' completes in less than 2 hours?

5.3.3 Estimation of Probability Density Function

For the estimation of the probability density function using a sample mathematical statistics offers various techniques. Most prominent categories are parametric and nonparametric techniques. Parametric techniques assume that the density belongs to a class of parametrized density functions. Then the sample is used to estimate these parameters. A commonly used parametric density estimate assumes a normal distribution, which is parametrized by mean and standard deviation. Thus, the sample is used to estimate those two parameters. Nonparametric techniques make no assumption on the underlying distribution. A commonly used nonparametric density estimation technique are Kernel Density Estimators [Sil86].

Given the stream of cycle time events the density estimates for the different steps and transitions of the processes are continuously computed. For that purpose CEP queries are used. For example, if the statistical model is based on the normal distribution, the CEP query computes mean and standard deviation of the cycle times. These values are then used to derive the density estimate. If the statistical model is based on Kernel Density Estimators, the CEP query has to maintain the Kernel centers and the bandwidth. In both cases the queries refer to a sliding time window, i.e., only the events in the current time window are used to compute the density estimates. This approach ensures that the estimates focus on recent trends in the development of the cycle times. Additionally, it confines the memory required for maintaining internal states. For the case of very large windows, e.g. weeks or months, Cluster Kernels [HS08] can be used for kernel density estimation as they allow the online computation of density estimates over streaming data. As mean and standard deviation require for an online computation only a constant amount of memory, also the normal distribution can be computed for very large windows.

Overall, the CEP queries continuously derive statistical models of the associated cycle time distributions from the stream of measured step and transition cycle times. The resulting statistical models for each transition and step are maintained in a model database/cache. Besides the latest model, that layer also can store histories of the models in order to allow for comparisons between recent and past behavior. This history is particularly relevant for storing significant changes in the cycle time distributions which occur only seldom or periodically. For example, in summer season the number of insurance claims related to lightning strokes is typically higher. These claims are more difficult to process as the payment amount is typically high, on-site investigations of the damages are necessary, etc. Therefore the cycle times for processing these claims are higher.

5.3.4 Statistical Models for Two Process Changes

So far we have only discussed the cycle time distribution for a single step/transition. Additionally, a model is required for a sequence of subsequent steps and transitions. Given a step and a subsequent transition, the question is how long the combined cycle time for those two is. According to the previous discussion we have for both a separate statistical model. Let X_1 be the random variable underlying the step cycle time and X_2 the random variable underlying the transition cycle time. Then the combined cycle time can be modeled as the distribution of the sum of X_1 and X_2. Given the densities for X_1 and X_2 the density of that sum is defined as

$$\hat{f}_{X_1+X_2}(x) = \int_{-\infty}^{\infty} \hat{f}_1(y)\hat{f}_2(x-y)dy$$

Thus, the density estimates for X_1 and X_2 can be used to derive an estimate for the sum of those two random variables. Regarding evaluation of that formula, in some cases this integral can be resolved with a closed formula, e.g., for the sum of two normal distributed variables. If that is not the case, numerical integration techniques can be used for an approximate closed formula.

Thus, this approach can be utilized to derive a statistical model for the cycle time of a step followed by a transition. The same holds for a transition followed by a step.

5.3.5 Statistical Models for Multiple Process Changes

Next case is that multiple transitions, not only one, end in one step. In that case the maximum of the transition cycle time distributions is considered first. The statistical model for that maximum distribution can be derived by using the product of the density estimates of the contributing transition cycle times as overall density estimate. That density estimate for the transitions is combined with the density estimate for the subsequent step. The overall cycle time is the sum of the maximum cycle time of the transitions plus the cycle time of the step. Again the two density estimates for the transitions and the step are combined with the above formula to derive a density estimate for the overall cycle time of multiple transitions ending in one step.

Overall, these mechanisms allow to estimate the cycle time of two subsequent process changes. This can be generalized so that the cycle time for more than two subsequent process changes can be estimated. This in turn provides the means to estimate a process stage violation. Given two arbitrary positions A and B in a process, the models can be used to determine the probability that getting from A to B requires a certain amount of time, e.g., the probability that the time from A to B is between 15 and 20 minutes is 73%. Figure 6 illustrates that approach for the cycle time from 'Receive claim request' being completed to 'Assess claim documentation' being started.

The database/cache for the statistical models has an additional access layer for the latest statistical models. It basically offers the following query functionality:

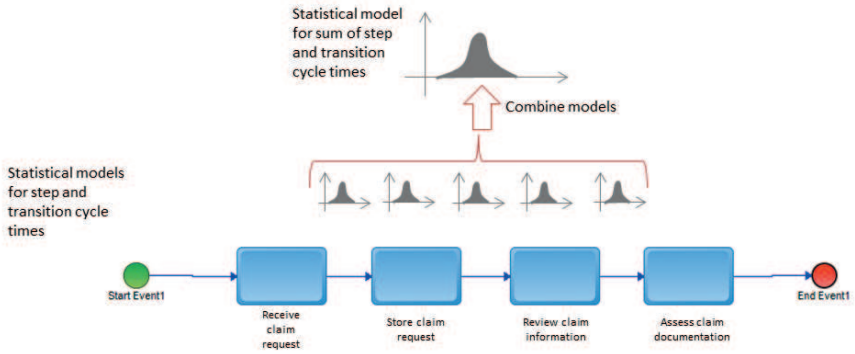

Figure 6: Computation of combined statistical model

- Input: A process model id, two positions A and B in the process, a time interval $[a, b]$
- Output: All paths from A to B plus the probability for each path that the time for the path traversal is in the time interval $[a, b]$

Regarding the actual computation of the statistical models while process instances are being executed, different phases are distinguished. When the process instances are initially started, the learning of the step cycle time models using measured cycle times as previously described requires some time. Therefore as discussed in the configuration chapter the user can specify initial statistical models based on his expert knowledge. In the initial learning phase these models are used. Once the statistical models have been learned, the system starts to use them instead of the user-specified models, provided the user has activated automatic model learning. For the learned statistical models different statistical techniques have been presented to determine the corresponding density estimates. By default a normal distribution is assumed for the following three reasons: First, the required parameters mean and standard deviation can be efficiently computed by means of simple Complex Event Processing queries using aggregates. Second, the parameters can be stored with minimum effort. Third, the sum of these statistical models can be computed with a closed formula.

Even though a considerable amount of real-world phenomena can be suitably modeled with a normal distribution, in many cases it does not fit. For example, the insurance company receives a significantly higher number of insurance claims due to a recent hailstorm. To deal with that overload also teams so far not familiar with that class of claims are assigned to these claims. Most likely, these team members will need more time for processing a claim than members familiar with those claims. So the cycle times of corresponding process steps more likely have a bimodal instead of a unimodal distribution. Similar effects can arise after sending some team members to trainings to improve their skills in handling more complex claims. In order to deal with these cases and to prevent decisions being made on the basis of an inadequate statistical model, the suitability of this normal distribution assumption is periodically checked by the system. A statistical test, e.g. the

Kolmogorov-Smirnov-Test, is used to test whether the measured cycle times, on which the model is based, are normal distributed. If that is not the case, the system switches to a nonparametric statistical model, which comes at the expense of higher complexity for model computation and combination of models, but fits better to the actual data.

5.3.6 Detection of Potential Stage Violations

The continuously computed statistical models are leveraged to detect potential stage violations. For each incoming process instance change event it is checked whether associated stages may be violated. Thus, for an incoming event the first task is to retrieve all stages that are registered for the associated process model. From these stages only those are selected where the current process state lies between start and end milestone. For each of those stages the start and end milestone as well as the associated temporal condition are determined. The process instance change event gives information on the current step or transition as well as when the change took place. This point in time is combined with the stage information to determine the remaining time to reach the end milestone. This requires additionally the time when the start milestone has been completed. Given the start milestone completion time, the current time of the event, and the temporal condition, the remaining time to stage completion can be derived. In the example in Figure 2 the stage definition requires that the time from the start of 'Receive claim request' to the completion of 'Assess claim documentation' is less than 5 hours. The start milestone, namely 'Receive claim request' has been completed, was reached at 9:00. The current event states that 'Store claim request' has been started at 12:40. Thus, the end milestone, completion of 'Assess claim documentation', has to be reached within 80 minutes in order to not violate the stage. The remaining time to end milestone completion as well as the current process state are then combined with the statistical models. As described in the previous section the probability that the time from current process state to end milestone completion is less than the remaining stage completion time is computed. This delivers the probability for violating that stage. For example, a process instance change event stating that process instance 4711 has completed 'Store claim request' at 12:40 may result in two events stating that Stage1 is violated with 70% probability and Stage2 with 17% probability.

Up to now stages with 'less than' conditions have been described, i.e., the stage condition demands a completion in less then a specified timespan. Analogously the same approach can be used to detect violations of 'greater than' stages, i.e., the stages have to take at minimum the specified amount of time.

Regarding the implementation of that functionality, again CEP queries are used. Each time a new process instance change event arrives, events describing potential stage violations are computed and directly published to follow-up consumers. Such a stage violation event provides information about

- Process model
- Process instance
- Current process step

- Stage
- List of paths to the end milestone
  - Steps of each path
  - Violation probability for each path

5.3.7 Result Processing Layer

The final layer processes the results produced by the stage analysis layer. It receives continuously events describing stage status as well as potential violations. These results are presented to the user in a suitable UI. This UI is real-time enabled, i.e., each time a new violation event is received, the corresponding components of the UI are directly updated. This is implemented by means of the underlying Event-Driven Architecture.

There are different ways to visualize stage violations. Some of them are exemplary discussed. A table can visualize all recently received stage violations. In order to limit the size of that table, events can be discarded depending on the severity of the violation, e.g., all violations with probability below 20% are discarded. Additionally that table can sort and highlight stage violations with respect to their severity. A stage violation event may comprise different paths to the end milestone with each path having a violation probability. In that case the maximum of those probabilities is used to determine the overall severity of the stage violation. The user can select an entry in the table to get more details. The complete information of that stage violation is then displayed, including in particular the paths to the end milestone. Additionally the process model is visualized. If the user selects a path to the end milestone, this path is highlighted in the process model, i.e., the current process position is highlighted and the subsequent steps to the end milestone. For each of those steps and transitions the user can also examine the associated statistical models to gain more insights into the actual behavior of the process steps.

The following example screenshot in figure 7 combines these components. The user selects a stage violation and gets details on the selected stage, the different paths to the end milestone, the associated process model with the selected path being highlighted, and the statistical models for each step on that path.

Besides visualizing the stage violations other actions can be triggered for an incoming event. For example, a follow-up process is triggered, a call is executed, or a SMS is sent. Again, these actions can be triggered dependent on the severity of the stage violation.

6 Extensions

The presented framework for tracking important sections of business processes can be extended in different areas:

Inclusion of intra-process dependencies in cycle time models: In the previous discussions on modeling the cycle times a separate model has been established for each step,

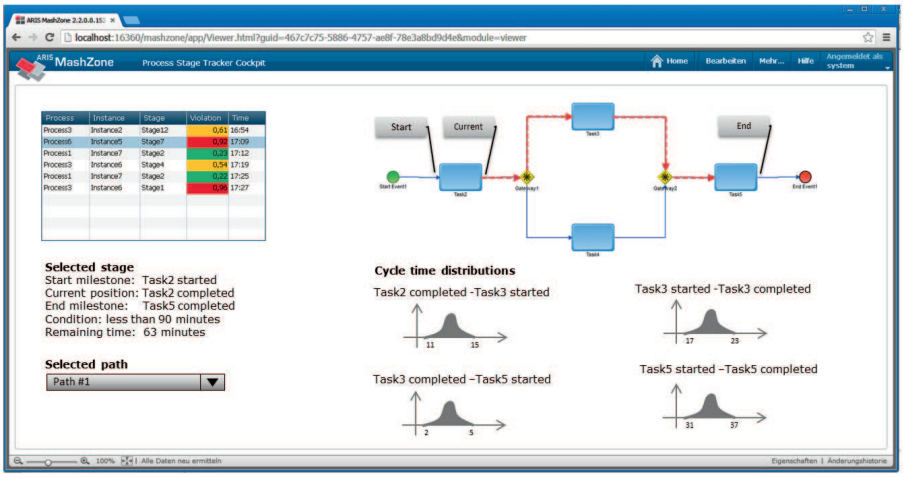

Figure 7: Example for a Process Stage Tracker Cockpit

with the assumption that the cycle time of one step does not interfere with the cycle time of the previous step. If we skip this independence assumption, a conditional cycle time model has to be used that copes with the dependencies of step executions.

Inclusion of inter-process dependencies in cycle time models: So far we have discussed cycle time distributions for a process step being estimated using data from instances of the corresponding process. A more complex model would include also the dependencies between different processes and their effects on the cycle time distributions. These inter-process dependencies correlate the information of process instances from multiple processes.

Notification of exceptionally long cycle times: The statistical model of a step cycle time provides a well-defined estimate of the distribution. This model can be used to detect exceptionally long cycle times. By comparing a measured cycle time with the latest model, the probability for the occurrence of that cycle time can be derived. If this probability is very low, that value can be marked as outlier. The user is then informed about that outlier as it affects the estimation of the remaining time to end milestone, which will most likely be higher due to that outlier.

Other tracking targets: Besides using stages associated with temporal conditions, other types of stages can be examined in a similar fashion. These stages could for example be based on the number of process step iterations. Besides the case of process steps changing from status started to status completed, the framework can also be adapted to deal with more complex statuses like waiting, interrupted, expired etc. Also other important exceptions in the execution of process instances can be implemented within the presented framework, e.g., detection of out-of-sequence events, process timeouts.

7 Conclusions

This work discusses an extension of business process management and execution with respect to a fine-granular real-time monitoring of crucial parts of business processes. A novel framework for tracking stages of business processes in an event-driven manner is introduced. This framework allows the business user to track in real-time whether constraints associated with those stages are fulfilled or violated. The framework relies on an Event-Driven Architecture (EDA) coupled with a Complex Event Processing (CEP) engine. The process engine executing the business process instances emits process status events within the EDA. The CEP engine receives those events and uses queries to determine whether registered stage constraints are met or not. Additionally, statistical models of the cycle times of process steps and transitions are continuously determined. These models are used to derive the probability that a stage constraint is violated, i.e., that the successful completion of a stage is at risk. This combined stage tracking functionality delivers the business user real-time insights into the execution of his business processes, with a focus on the most important sections so that he can directly or even proactively act in case of problems.

In our future work we will continue to incorporate more sophisticated business process monitoring functionality within the Intelligent Business Operations platform of Software AG. We will investigate the effectiveness and efficiency as well as the scalability of our approach in different application domains consisting of a multitude of concurrently running process instances. In the course of this investigation we will also assess the application of our process tracking solution within the context of iPRODICT [iPR14], an ongoing German research project which tackles a forecast-based adaptation of business processes in real-time.

References

[BBH+14] Susanne Bülow, Michael Backmann, Nico Herzberg, Thomas Hille, Andreas Meyer, Benjamin Ulm, Tsun Yin Wong, and Mathias Weske. Monitoring of Business Processes with Complex Event Processing. In Niels Lohmann, Minseok Song, and Petia Wohed, editors, *Business Process Management Workshops*, volume 171 of *Lecture Notes in Business Information Processing*, pages 277–290. Springer International Publishing, 2014.

[BDR12] Geoff Bullen, Richard M. Dickson, and Warren C. Roberts. Systems and/or methods for prediction and/or root cause analysis of events based on business activity monitoring related data. Patent US8140454 B2, 2012.

[CFLZ13] Hung-Yang Chang, Joachim H. Frank, Christoph Lingenfelder, and Liangzhao Zeng. Non-intrusive event-driven prediction. Patent US8468107 B2, 2013.

[CK07] Pawan Chowdhary and Shubir Kapoor. System and method for applying predictive metric analysis for a business monitoring subsystem. Patent US20070244738 A1, 2007.

[EN10] Opher Etzion and Peter Niblett. *Event Processing in Action*. Manning Publications Co., 2010.

[HS08] Christoph Heinz and Bernhard Seeger. Cluster Kernels: Resource-Aware Kernel Density Estimators over Streaming Data. *IEEE Transactions on Knowledge and Data Engineering*, 20(7):880–893, 2008.

[iPR14] iPRODICT. http://www.software-cluster.org/de/forschung/projekte/partnerprojekte/iprodict, 2014.

[MFE12] Andreas Metzger, Rod Franklin, and Yagil Engel. Predictive Monitoring of Heterogeneous Service-Oriented Business Networks: The Transport and Logistics Case. In Ralph Badinelli, Freimut Bodendorf, Simon Towers, Sharad Singhal, and Manish Gupta, editors, *Proceedings of the 2012 Annual SRII Global Conference*, pages 313–322. IEEE Computer Society, 2012.

[Ope14] Software AG Intelligent Business Operations. http://www.softwareag.com/corporate/products/bigdata/ibo/overview/, 2014.

[SC09] Roy Schulte and Mani Chandy. *Event Processing: Designing IT Systems for Agile Companies*. McGraw-Hill Osborne Media, 2009.

[Sil86] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[WZM+11] Matthias Weidlich, Holger Ziekow, Jan Mendling, Oliver Günther, Mathias Weske, and Nirmit Desai. Event-Based Monitoring of Process Execution Violations. In Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors, *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 182–198. Springer Berlin Heidelberg, 2011.