Preventing KPI Violations in Business Processes based on Decision Tree Learning and Proactive Runtime Adaptation

Branimir Wetzstein¹, Asli Zengin², Raman Kazhamiakin³, Annapaola Marconi², Marco Pistore², Dimka Karastoyanova¹, Frank Leymann¹

¹Institute of Architecture of Application Systems (IAAS), University of Stuttgart, Germany
branimir.wetzstein@iaas.uni-stuttgart.de
karastoyanova@iaas.uni-stuttgart.de
leymann@iaas.uni-stuttgart.de

²Fondazione Bruno Kessler IRST, Italy
zengin@fbk.eu
marconi@fbk.eu
pistore@fbk.eu

³SayService srl, Trento, Italy
raman@sayservice.it

Abstract: The performance of business processes is measured and monitored in terms of Key Performance Indicators (KPIs). If the monitoring results show that the KPI targets are violated, the underlying reasons have to be identified and the process should be adapted accordingly to address the violations. In this paper we propose an integrated monitoring, prediction and adaptation approach for preventing KPI violations of business process instances. KPIs are monitored continuously while the process is executed. Additionally, based on KPI measurements of historical process instances we use decision tree learning to construct classification models which are then used to predict the KPI value of an instance while it is still running. If a KPI violation is predicted, we identify adaptation requirements and adaptation strategies in order to prevent the violation.

Key words: Business Activity Monitoring, Business Process Intelligence, WS-BPEL, Decision Tree, Process Adaptation.

1. Introduction

In recent years, the industry experienced a wide adoption of the service-oriented architecture for the implementation of business processes [15]. Service-based applications realize such processes by modeling and deploying a complex, distributed and layered system, where the business model of an application is implemented through a service composition which orchestrates services running on a service infrastructure [12, 14]. To be effective, such applications should meet certain business goals, traditionally expressed as Key Performance Indicators (KPIs) of the business processes. These KPIs are typically continuously monitored at run-time using business activity monitoring techniques.

If monitoring shows that KPI targets are not reached, then it is necessary to identify the factors which strongly influence the KPI and cause KPI target violations most often. In complex business processes, the relations between the overall business process performance and lower-level influential factors and their combination are neither explicit nor easy to reveal. In addition to identifying the influential factors based on historical process executions, it is desirable to be able to predict for a running process instance whether it will reach the KPI target. This allows us to react in a timely fashion and possibly prevent a predicted KPI target violation by identifying an adaptation strategy which can potentially improve the performance of the running process instance.

In this paper we present an integrated monitoring, prediction and adaptation approach that aims to address the above problems. The execution of the business process is continuously monitored based on runtime events published by the process execution middleware. Based on monitoring data of historical process instances, we use decision tree algorithms in order to learn the dependencies between the KPI and the influencing lower-level metrics. The resulting KPI dependency tree is used for KPI prediction in future process instances. If for a running process instance, a KPI target violation is predicted, adaptation requirements are extracted from the decision tree specifying predicates on the
metric values that should be improved. In the next step, we identify adaptation strategies consisting of adaptation actions which should be performed in order to satisfy the adaptation requirements. In a subsequent step, we filter and rank adaptation strategies based on constraints and preferences model. Finally, the process instance is pro-actively adapted in order to prevent the KPI target violation. The presented work builds on the work presented in [16] which focused on monitoring and KPI dependency analysis, and extends, refines and evaluates our preliminary ideas presented in [7], where the overall monitoring, analysis and adaptation framework has been described on a higher level.

The paper is organized as follows. We begin with a motivating scenario that describes the problem and which we use in the rest of the paper to present our solution. Section 3 gives an overview of the approach by describing its lifecycle. Section 4 presents the different types of artifact models created at design time. Section 5 describes the runtime phases consisting of learning of KPI dependency trees, KPI prediction, and adaptation. Section 6 describes the implementation of the approach and presents results of an experimental evaluation. Finally, we give a summary of related work and conclude the paper together with the directions for future work.

2. Scenario and Motivation

In this section we introduce a scenario that we use in the following sections for explaining our approach. As shown in Figure 1, the scenario consists of a purchase order process implemented by a reseller who offers products to its customers and interacts with external supplier, banking, and shipment services for processing the order. Furthermore, the reseller uses warehouse and packaging services, which are internal to the organization.

The defined KPIs have to be measured and calculated based on executed process instances. If after a while the monitoring shows an unsatisfactory result, i.e., undesirable KPI classes (KPI violations) are reached for many instances (i.e. purchase orders are late, in our case), the reseller wants to find out the influential factors which lead to those KPI classes. Automatically identifying the influential factors is not trivial. Let us consider for instance our reference example: understanding the reasons why certain orders are delivered on time and others are not is a complex task, as the KPI depends on the...
combination of several factors such as ordered product types and amounts (input data of the process), duration and availability of the internal services, duration, reliability, SLA conformance of external services etc. After the KPI dependencies have been understood based on historical process instances, they can be used for predicting the KPI classes of future instances.

The next step is then to try to prevent the KPI violations in future. This can be done using runtime process instance adaptation. Therefore, we assume that there is a set of alternatives in the process execution which can be chosen dynamically to proactively adapt the process instance. Assume for example that there is a set of alternative services for a set of service types used in the process. For example, there might be several alternative shippers which offer different service levels via shipment options (e.g., standard, premium, overnight express); each of those options can be modeled as a candidate service with different quality of service characteristics (such as shipment delivery time, shipment cost, reputation, etc.) Based on the prediction, the goal is then to choose a shipper which will be likely to lead to a desirable KPI class.

3. Solution Overview and Methodology

In this section we give an overview of our approach by describing its lifecycle as shown in Figure 2.

![Fig. 2: Lifecycle of the Approach](image)

The supporting architecture and implementation are described in Section 6. The lifecycle consists of the following phases:

1. **Modeling:** In the modeling phase (not shown explicitly in Figure 2), several models are created: (i) application model, which in our case consists of a deployable process model implemented as a service orchestration, (ii) metrics model, defining process and QoS metrics which have to be measured, and the KPIs specifying targets on key metrics based on business goals (iii) adaptation action model, specifying the available adaptation actions, (iv) check point model, defining at which points in the process the prediction and potential adaptation should take place, (v) preferences and constraints model, needed for selection of an adaptation strategy.

2. **Monitoring:** In the monitoring phase, all metrics specified in the metrics model are monitored. That includes the KPIs but also lower-level metrics of the potential influential factors. As a result, metric values for a set of executed process instances are obtained.

3. **KPI Dependency Analysis:** After a certain number of executed process instances, for each KPI at each checkpoint a decision tree is trained which helps to understand the dependencies of that KPI on lower-level metrics. The resulting KPI dependency trees serve from now on as
classification models for future process instances and are used for KPI prediction. The learning of the decision trees is performed “offline” (in the background) based on history data and does not affect the execution of running process instances.

4. **KPI Prediction:** When a running process instance reaches a checkpoint, it halts its execution. The metric values which have been measured until the checkpoint for that instance are gathered and used as input to the classification model(s) learned in phase 3. The prediction result per KPI is (in the special case) a predicted KPI class (e.g., “good”, “medium” or “bad”) or (in the general case) an instance tree, i.e., a subtree of the original tree, which shows for a particular running process instance which metrics should be improved to reach a specific KPI class and serves thus as basis for adaptation.

5. **Identification of Adaptation Requirements and Adaptation Strategies:** Adaptation requirements are identified by extracting the metrics that should be improved from the instance tree(s). Based on the adaptation requirements, a set of alternative adaptation strategies is identified by taking into account available adaptation actions. An adaptation strategy thus consists of a set of adaptation actions that should be used in the process instance in order to reach a certain desired KPI class.

6. **Selection of an Adaptation Strategy:** The list of alternative adaptation strategies is filtered and ranked based on a constraints and preferences model. Constraints allow defining conditions which should never be violated, while preferences are specified as weights on different KPIs and metrics and lead to a strategy score number determined based on multiple attribute decision making.

7. **Adaptation Enactment:** The adaptation strategy with the best score is enacted by executing the adaptation actions. The process instance is unblocked and continues its execution while taking into account the performed adaptation.

After the steps 4-7 have been performed for a certain number of instances, the effectiveness of the adaptations can be evaluated by checking how many KPI violations have been prevented and how many instances still violate their KPIs. This might lead to (re-)adjustment of the models, e.g., adjustment of KPI targets, (re)moving or adding of checkpoints, and adjustment of the constraints and preferences model.

4. **Design Time: Modeling for Monitoring, Prediction and Adaptation**

In this section, we describe the different types of models created at design time: metrics model, adaptation actions model, check point model, and constraints and preferences model. These models are used as input to the runtime phases (Figure 2). An overview of the overall metamodel is shown in Figure 3.

![Fig. 3: Design-Time Artifacts Metamodel](image-url)
4.1 Metrics and KPIs

The metrics model contains (i) KPIs and the underlying KPI metrics, i.e. key metrics reflecting the time, cost, and quality dimensions of the process which help to assess the process performance, (ii) metrics which KPIs potentially depend on, i.e., lower level metrics used during KPI dependency analysis and prediction.

A metric definition contains in particular the following elements:

- **data domain**, i.e. all unique values the metric can contain (e.g., real numbers, nominal values)
- **entity** characterized by the metric (e.g., a process instance, activity instance, service endpoint)
- **measurement definition** which specifies how the **metric value** is to be obtained. It therefore uses one or more measurement mechanisms, e.g., a probe, or an event processing engine. This definition can be based on other metrics.

The metrics model is deployed on the monitoring infrastructure and is used to monitor process instances during their execution in order to obtain **metric values** for the defined metrics. The metric measurement can be realized based on diverse monitoring mechanisms. In our prototype, we use an event-based approach receiving process events from the process execution engine and the service infrastructure and correlating and aggregating those events based on complex event processing (see Section 6).

In our scenario, we define among others the metric **Order Fulfillment Time** to measure the duration of each process instance of the reseller process. The metric value is calculated by receiving and correlating two corresponding events of a reseller process instance (start of the activity “Receive PO” and end of the activity “Shipment”) and subtracting their timestamps. The correlation is performed based on a process instance ID which every corresponding process event contains. The corresponding metric values (one per process instance) are stored in a Metric Database.

In order to be able to assess the monitored metric values in respect to business goals, a set of KPIs is defined. A KPI is defined based on a metric (KPI metric) and maps value ranges of that metric to a set of nominal values (KPI classes) which allow evaluating whether that metric conforms to business goals.

A Key Performance Indicator (KPI) definition contains the following elements:

- the underlying **KPI metric**
- a set of nominal values (>=2) representing **KPI classes** (e.g., “good”, “medium”, “bad”)
- a **target value function** which maps values of the KPI metric to KPI classes

The KPI is itself a metric and is defined in the metrics model.

In the scenario, we use the metric Order Fulfillment Time as the KPI metric, specify three KPI classes “green”, “yellow”, “red”, and then define a target value function as follows: < 4 days → “green”, > 4 days and < 7 days → “yellow”, otherwise “red”. Note that in this case, the KPI class is evaluated per process instance as the underlying metric is evaluated per process instance. The KPI could however also be based on a metric that is calculated based on several process instances in a period (e.g., average order fulfillment time in the last month).

4.2 Adaptation Actions

The adaptation actions model defines (i) a set of adaptable entities in a process and (ii) a corresponding set of adaptation actions which implement alternative realizations of adaptable entities.

An adaptable entity definition contains the following elements:

- **entity** which can be adapted (e.g., in a BPEL process [13] that could be a particular partner link instance, activity instance, variable instance)
- a set of **metrics** which characterize this entity

In our scenario, we define the shipment and supplier partner links in the reseller process as adaptable entities, as there are alternative shipment and supplier services available. For the shipper partner link, for example, we define “shipment delivery time”, “shipment cost”, “shipper reputation” as characterizing metrics.

An adaptation action (AA) definition contains the following elements:

- adaptable entity targeted by this action
• adaptation specification which defines how the adaptable entity is to be adapted, e.g., substitution of another service, skipping of a process activity, process variable value change etc.

• a set of metric effects which specify the impact of the adaptation action on metric values of the adaptable entity. The impact is specified as a predicate on metric values (e.g., delivery time < 3 days). The metric effects can be derived from past measurements; if no such measurements are available then they have to be estimated or in some cases can be derived from SLAs (e.g., in case of service substitutions).

We have predefined three adaptation action types in our prototype which can be used for adapting a running BPEL process instance after it has been halted at a check point (Section 4.3): (i) WritePartnerLink allows changing the service EPR (endpoint reference as defined in WS-Addressing) property in a partner link in the BPEL process thus effectively performing service substitution; (ii) WriteProcessVariable allows changing process variable values, which can be used for example for changing the control flow in data-based branching activities (e.g., if-else); (iii) ChangeActivityState, which allows e.g. skipping of activities. Of course, this set of adaptation action types could be extended to include other types of adaptation such as infrastructural reconfiguration.

In our concrete scenario, we have assumed that there is a set of alternative shipment and supplier services with different QoS characteristics. For each alternative, we have created an adaptation action and specified its effects on metrics of the corresponding adaptable entity. Thereby, we assume that the effects can be derived from SLAs, or estimated based on experience if no measurement data is yet available on those services. For example, an AA which substitutes a new shipment service defines its effects on the shipment delivery time, shipment cost, and shipper reputation.

4.3 Check Points

For performing prediction and adaptation, one defines one or more checkpoints in the process.

A checkpoint definition contains the following elements:

• a trigger defined as a process runtime event (or derived event from a process runtime event) typically signaling the start or completion of an activity. The event is typically but not necessarily configured to be blocking, i.e. to stop process instance execution until prediction and potential adaptation are performed.

• a set of available metrics from the metrics model whose metric values are available at this checkpoint and which should be used as explanatory attributes for creating the classification model (one per KPI) for this checkpoint (see Section 5.1). The set of available metrics for a check point is created automatically (by analyzing the process model and deriving data and time metrics available at a check point) and provides suitable results in most cases but can be adjusted by an “advanced user” if he wants to influence the classification learning process (see [16] for a more general discussion on this topic).

• a set of available adaptation actions from the adaptation actions model which can be used to adapt the process at this checkpoint.

Obviously, the set of available metrics increases in size the later the checkpoint is defined in the process thus increasing prediction accuracy, however at the same time the set of available adaptation actions will decrease, and thus there will be fewer adaptation possibilities or it could even be too late for adaptation. Thus, there is a tradeoff between prediction accuracy and adaptation possibilities. In long-running processes where the prediction and adaptation only marginally influence the overall process execution time, one could define and use many different check points in a process model.

In our example, after the “check in stock” activity at the beginning of the process, available metrics are e.g. the ordered product types and amounts, the customer, the process duration until that activity, and whether the ordered items are in stock. Available adaptation actions are supplier and shipment service substitution.

4.4 Constraints and Preferences

When several alternative adaptation strategies are identified, we need to make a decision which of those alternatives is to be selected. Thereby, we address two aspects: (i) adaptation strategies might violate certain rules or thresholds which should always be avoided, (ii) adaptation strategies have
different effects on a set of competing KPIs and metrics (e.g., time vs. cost). The former aspect is addressed via constraints, the latter via preferences.

A constraint defines a boolean-valued predicate over one or more metrics measured for the running instance. If during the selection of a strategy a constraint evaluation results in the value “false” for a strategy, then that strategy is removed from the set of alternatives.

In our scenario, we use constraints for defining which KPI classes should be prevented in any case (KPI target violations), e.g., by specifying that the predicted class of a KPI ≠ “red”. We use constraints also on metrics of adaptable entities, e.g., maximal cost of supplier and shipper service < x.

Preferences are used for ranking of adaptation strategies according to a score represented by a number between 0 and 1. Therefore, we use Simple Additive Weighting as part of Multiple Attribute Decision Making [5]. At design time, the user has to assign a weight between 0 and 1 to each KPI and metric of an adaptable entity, whereby the sum of all weights should be 1. At runtime, then a score is calculated as discussed in Section 5.4.

In the scenario, where we have specified one KPI and three metrics for each of the two adaptable entities, we thus have to assign weights to seven metrics in total.

The constraints and preferences model can be used at design-time for creating a default configuration of the adaptable entities (i.e., provide them with initial values). In our case, we can select a combination of a supplier and shipper service which has the highest score according to the subset of preferences and constraints which can be evaluated at design time (e.g., KPI class is not known before runtime, however the delivery times of the shippers and suppliers are known at design time from the modeled metric effects and can be used to create a default configuration). This default configuration can then be changed at runtime based on KPI prediction results.

5. Runtime: KPI Dependency Tree Learning, Prediction and Adaptation

In this section, we present the runtime phases in detail. Figure 4 shows an overview of the artifacts created at runtime. After a set of process instances and corresponding metric values have been monitored as defined in the metrics model, a KPI Dependency Tree is learned for each KPI at each check point based on classification learning techniques. When a checkpoint is triggered for a running process instance thus creating a check point instance, the KPI Dependency Tree is used to predict the class of the corresponding KPI in the running process instance by inserting the available metrics values at that check point instance into the tree. The result is an instance tree which shows the predicted KPI class in relation to metrics which can be adapted. The instance tree is then used to derive adaptation requirements and corresponding adaptation strategies consisting of adaptation actions.

![Fig. 4: Runtime Artifacts Metamodel](image-url)
5.1 Creating KPI Dependency Trees based on Classification Learning

The KPI metric value and the corresponding KPI class depend typically on the combination of a set of influential factors (alt. influential metrics), e.g., input data to the process (ordered product types and amounts), service outputs (e.g., ordered products available in stock) and processing duration of used services (e.g., shipment delivery time).

In order to find out these dependencies, we use classification learning known from machine learning and data mining [17]. In a classification problem, a dataset is given consisting of a set of examples (a.k.a. instances) described in terms of a set of explanatory attributes (a.k.a. predictive variables) and a categorical target attribute. The explanatory attributes may be partly categorical and partly numerical. By using a learning algorithm, based on the example dataset (a.k.a. training set) a classification model is learned (a.k.a. supervised learning), whose purpose is to identify recurring relationships among the explanatory variables which describe the examples belonging to the same class of the target attribute. The so created classification model can be used to explain the dependencies in past instances but in particular also to predict the class of (future) instances for which only the values of the explanatory attributes are known.

We map the KPI Dependency Analysis to a classification problem by defining the KPI as the categorical target attribute with categorical values as KPI classes, and a set of lower-level metrics (potential influential factors) which serve as explanatory attributes for this KPI at the check point. Classification learning for a KPI is then performed for each check point separately as the set of available explanatory attributes is different for each check point. This set consists of two types of metrics: (i) metrics whose values are available at the check point; this set is part of the check point definition (ii) metrics whose values cannot be measured until the checkpoint but which are affected by the available adaptation actions of the check point. The latter group of metrics is important as we want to learn how the KPI class depends on metrics which are affected by adaptations. This will allow us to extract adaptation requirements from the tree (Section 5.3). If a tree would be trained only based on available metrics at a check point, then the prediction would yield the predicted KPI class as a result, however we would not know how to adapt the process in case a bad KPI class is predicted.

At process runtime, after a set of process instances have been executed, we construct a data set for a KPI at a check point as follows. For each instance, we create a data item consisting of (i) the metric values of the available metrics and “adaptable” metrics defined for that check point, (ii) the KPI class of the KPI metric value for this instance. Based on this data set resulting from monitoring, a classification problem consists now of identifying a classification model that can optimally describe the relationship between the metrics and the KPI class.

There are different types of algorithms for classification model learning and prediction, e.g., artificial neural networks, classification rules, and support vector machines [17]. We have decided to use decision trees because of their following advantages in our context: (i) They constitute a white box model as they show explicitly the relationships between explanatory attribute value ranges and categorical target attributes (i.e., KPI classes). Thus they are easy to understand and interpret for people and enable human support in the learning and adaptation phases. (ii) They support both explanation and prediction. (iii) In particular, they support extraction of adaptation requirements from the tree paths (Section 5.3). (iv) Furthermore, decision trees support both numeric (typically, time based metrics) and categorical explanatory attributes (typically, process data based metrics).
A decision tree algorithm works by splitting the instance set into subsets by selecting an explanatory attribute (new node in the tree) and corresponding splitting predicates on the values of that attribute (branches). This process is then repeated on each derived subset in a recursive manner until all instances of the subset at a node have the same value of the target attribute or when splitting does not improve the prediction accuracy. There are different types of decision tree algorithms. They differ, for example, in how they select predictive attributes for splitting (e.g., based on information entropy), or splitting predicates, e.g. whether the tree is binary, or can have more than two outgoing edges per node. The algorithm automatically performs a validation of the learned classification model (e.g., using cross-validation) and calculates quality metrics of the tree, in particular its accuracy, i.e. the percentage of correctly classified instances (based on a test set). A KPI dependency tree is learned automatically at runtime for each KPI per checkpoint. It can be configured after how many instances the tree should be learned and when it should again be retrained.

In our approach, we have used the popular J48 algorithm to generate the KPI dependency tree [17].

A KPI Dependency Tree (J48) consists of a (possibly empty) set of non-leaf nodes representing metrics and a non-empty set of leaf nodes representing KPI classes. Thereby, a particular metric or KPI class can be present in the tree zero to several times. An outgoing edge of a tree node defines a predicate on the values of the metric of that node. The metric values on outgoing edges of a node are disjoint. Each leaf node contains the number of instances which satisfy the path of this leaf to the root. Thus, by following the path from the root to a leaf node, we learn which metric values lead to a particular KPI class, and for how many instances that was the case in the past.

An example tree is shown in Figure 5. It has been generated for the Order Fulfillment KPI at the checkpoint defined right after the “check in stock” activity on the basis of 100 instances. The tree contains available metrics (order in stock, item quantity) and “adaptable” metrics at the checkpoint (shipment delivery time, supplier delivery time). It shows, for example, that for the combination “order in stock=true” and “item quantity <= 20” the KPI class “green” has always been reached in the past (which was the case for 30 instances). Overall, the tree shows that the order fulfillment time KPI mainly depends on whether the ordered items are available in stock. In the positive case, 45% (30+15) of all instances reached “green”, and 12% reached “red”. In the other case (order in stock = false), many KPI violations have occurred (36+23) and the KPI class mainly depends on the shipment delivery time and supplier delivery time.

5.2 Runtime Prediction based on KPI Dependency Trees

At process runtime, after a sufficiently large set of instances has been executed and monitored, based on the checkpoint definition, for each checkpoint a decision tree is learned. It explains how the KPI classes of those history instances depend on influential factor metrics.

---

Fig. 5: A KPI Dependency Tree for the Order Fulfillment Time KPI
In the next step the decision tree can be used for prediction. When the process instance execution reaches a checkpoint which is signaled by the specified event, the available metrics for that instance until the checkpoint are gathered and "inserted" into the decision tree for that checkpoint. Therefore we traverse the tree breadth-first; if the current node corresponds to an available metric, we follow the outgoing branch whose predicate is satisfied by the measured metric value and replace the current node with the target node of that branch; otherwise, if the metric is not available (but affected by available adaptation actions) we leave the node in the tree (and continue with its children until a leaf node is reached).

As the result we get a subtree of the original one (in the following denoted as instance tree) consisting either of (i) just one leaf representing the prediction of the corresponding KPI class (the special case); (ii) a tree containing one or more nodes which correspond to metrics which are affected by the available adaptation actions. In the latter (general) case, the KPI class is thus predicted in relation to the values of metrics which can be adapted.

![Instance Tree Diagram](image)

**Fig. 6: An Instance Tree of the above KPI Dependency Tree**

Figure 6 shows an instance tree created from the original tree shown in Figure 5 assuming that we have measured “order in stock=false”. This tree consists now only of metrics which are yet unknown but are affected by the available adaptation actions of this checkpoint. It is used for extraction of adaptation requirements as discussed in the next section.

### 5.3 Identification of Adaptation Requirements

At each check point, after obtaining an instance tree for each KPI, we have to decide whether adaptation is needed, and if yes, which metrics should be improved and how. An instance tree shows how the KPI class of the running instance depends on the metrics affected by available adaptation actions (adaptable metrics).

If the instance tree contains only one leaf denoting the KPI class, then the predicted KPI class is independent of the adaptable metrics and an adaptation would not lead to another KPI class (for this KPI). If the instance tree contains more than just one leaf (as the one in Figure 6), then the non-leaf nodes correspond to influential factor metrics which are adaptable by the predefined adaptation actions and the tree shows how we should adapt. For example, if we ensure a supplier delivery time below 3 days and a shipment delivery time below 2.2 days we will very likely (assuming that the classification model has a high accuracy, see Section 6) reach a “green” KPI class.

The idea towards adaptation is thus (i) to extract those paths and the corresponding metric predicates, which lead to desirable KPI classes, and then (ii) select adaptation actions, which will lead to satisfaction of those metric predicates. We call the paths of the instance tree which lead to desirable KPI classes safe paths. The user can configure which KPI classes are desirable for an instance (e.g., “green”, and “yellow” could be desirable, while “red” is to be avoided) via constraints in the constraints and preferences model. If we ensure one of the safe paths, then we avoid all of the undesirable paths, i.e. KPI target violations. Thus, eventually each safe path (consisting of a conjunction of metric predicates) is an alternative adaptation requirement for the corresponding KPI.
An adaptation requirement (AR) is extracted from a safe path as follows: from each branch on the path we extract the metric predicate and add it to the adaptation requirement. The predicates are combined by using logical conjunction, i.e., all predicates have to be true in order to satisfy the requirement. Finally, in the last step if there are predicates which are satisfied with semantically worse metric values (e.g., if a predicate is “supplier delivery time > 2 days”), then it can be ignored and removed from the requirement because the value does not have to be improved.

In case more than one KPI has been defined, alternative adaptation requirement sets are extracted from each instance tree separately and then combined by building a Cartesian product between them, whereby some of the resulting ARs can contain contradictory metric predicates and are then removed.

An adaptation requirement (AR) specifies
- the predicted desirable KPI class for one or more KPIs
- a conjunction of metric predicates which should be achieved in order to reach those desirable KPI classes

As a result we get a set of alternative adaptation requirements each consisting of a conjunction of predicates over adaptable metrics which have to be satisfied.

For the example instance tree (Figure 6), we can extract two adaptation requirements as shown in Table 1 (first two columns), one for the KPI class “green” and one for the KPI class “yellow”. We assume here that we have specified only one KPI, and that a constraint has been defined specifying the KPI class “red” to be undesirable.

### 5.4 Identification and Ranking of Adaptation Strategies

After the requirements have been identified, the next step is to identify adaptation strategies which can be used to satisfy the adaptation requirements. An adaptation strategy (AS) consists of a set of adaptation actions which satisfy the metric predicates of an adaptation requirement and the constraints in the constraints and preferences model.

A valid adaptation strategy (AS) consists of the following elements:
- **the adaptation requirement addressed by the strategy**
- **a set of adaptation actions** which should be enacted for this strategy, whereby all metric predicates of the corresponding adaptation requirement are satisfied by the metric effects of the adaptation actions and all constraints are satisfied
- **a score number**, calculated on the basis of the specified preferences

In the first step, for each adaptation requirement a set of alternative strategies is identified as follows:
(i) for each metric predicate of the AR we enumerate (alternative) adaptation actions which satisfy those predicates according to their metric effects; (ii) if there are adaptable entities where at least one of their metrics are not part of the AR, then all AAs for each such adaptable entity are also enumerated; (iii) the sets of AAs created in (i) and (ii) are combined using Cartesian product creating a set of alternative adaptation strategies for this AR. The result is a set of alternative adaptation strategies which would all according to the corresponding adaptation requirement are satisfied by the metric effects of the adaptable metrics and are then removed.

In the second step, that set is further filtered according to the constraints defined in the constraints and preferences model. If a constraint evaluation evaluates to “false” for a strategy, then that strategy is removed from the set. The result is set of alternative valid adaptation strategies.

In the third step, the strategies are finally ranked according to a score and the strategy with the highest score is enacted. The score of an adaptation strategy is calculated based on the preferences model which assigns weights to a metric set $M_w = \{m_1, m_2, \ldots, m_n\}$ consisting of KPIs and metrics of the adaptable entities (Section 4.4). For each adaptation strategy $x$ and metric $y$ in $M_w$ we can determine the value $v_{xy}$ (either from measurements or metric effects). Before applying the simple additive weighting (SAW) [5], we have to normalize these metric values to make the different metrics comparable. The normalized metric value $nv_{xy}$ can be calculated by using the division by maximum value method:

$$nv_{xy} = \begin{cases} \frac{v_{xy}}{\max_y(v_{1y}, \ldots, v_{py})} & \text{if "higher is better"} \\ \frac{v_{xy}^{-1}}{\max_y(v_{1y}^{-1}, \ldots, v_{py}^{-1})} & \text{if "lower is better"} \end{cases}$$
The normalized metric values $n_{xy}$ are in the range between 0 and 1, whereby the value 1 is always given to the best metric value. We thereby have to distinguish between metrics where a higher value is better (e.g., reputation) and metrics where a lower value is better (e.g., cost). Note that for KPIs and any other non-quantitative metrics, the categorical values have to be mapped to a cardinal scale to enable proper calculation of a normalized value; this mapping has to be provided by the user.

Finally, for each strategy we can calculate a score by summing up the weighted metric values:

$$\text{Score}_x = \frac{1}{P} \sum_{y=1}^{P} w_y \cdot n_{xy}$$

Finally, the best ranked strategy is selected and enacted.

### Tab. 1: Identification and Ranking of Adaptation Strategies

<table>
<thead>
<tr>
<th>Conditions</th>
<th>KPI</th>
<th>Strategy</th>
<th>Adaptation Strategies</th>
<th>Normalized Metric Values</th>
<th>Constraints</th>
<th>Score</th>
<th>Rank</th>
</tr>
</thead>
</table>
| Shipment Deliv. Time $< 2.2$  
+ Supplier Deliv. Time $< 3.0$ | green | Sh1 Premium + Su1 Premium | (1.0, 0.9, 0.2, 1.0, 1.0, 0.3, 0.9) | ok | 0.61 | 3 |
| | | Sh1 Premium + Su2 Premium | (1.0, 0.9, 0.2, 1.0, 0.8, 0.5, 0.8) | ok | 0.64 | 1 |
| | | Sh2 Premium + Su1 Premium | (1.0, 1.0, 0.1, 0.8, 1.0, 0.3, 0.9) | nok | 0.57 | - |
| Shipment Deliv. Time $< 2.2$  
+ Supplier Deliv. Time $< 7.5$ | yellow | Sh1 Premium + Su1 Standard | (0.5, 0.9, 0.2, 1.0, 0.8, 0.8, 0.8) | ok | 0.615 | 2 |
| | | Sh1 Premium + Su2 Standard | (0.5, 0.9, 0.2, 1.0, 0.7, 0.7, 0.6) | ok | 0.565 | 4 |
| | | Sh1 Premium + Su1 Premium | (0.5, 0.9, 0.2, 1.0, 1.0, 0.3, 0.9) | ok | 0.51 | 5 |

Table 1 shows the two adaptation requirements extracted from the instance tree (Figure 6) and the identified alternative strategies per requirement. Each strategy consists here of a combination of a shipper service and supplier service with different metric effects. For each strategy a normalized metric values vector is constructed containing the corresponding KPI class ("green" is mapped here to the value 1.0, while "yellow" is assigned 0.5), and the duration, cost, and reputation metrics for the shipper and the supplier, respectively. Based on the weight distribution (0.2, 0.05, 0.25, 0.1, 0.05, 0.25, 1.0) in the preferences model, the score for each strategy is calculated and used for ranking.

## 6. Prototype Implementation and Experimental Evaluation

We have implemented the approach as shown in Figure 7. Our prototype uses Apache ODE\(^1\) as the business process execution engine which executes BPEL processes [13]. The monitoring is performed based on the ESPER complex event processing (CEP) framework\(^2\) which calculates metrics based on events which are published by the process engine and a QoS monitor as already described in [16]. The classification model learner is based on the WEKA suite\(^3\) which provides decision tree algorithm implementations. For the implementation of check points and instance adaptation, we use a framework which extends the Apache ODE BPEL engine [8]. The check points are supported via blocking events which stop process instance execution until they are explicitly unblocked by a corresponding incoming event coming from our framework. The adaptation actions are supported by the same mechanism of incoming events whereby our framework populates the corresponding incoming event with the new partner link value, variable value, or the state of an activity and sends it to the process engine.

---

\(^1\) http://ode.apache.org/

\(^2\) http://esper.codehaus.org/

\(^3\) http://www.cs.waikato.ac.nz/ml/weka
6.1 Experimental Evaluation

We have implemented the scenario from Section 2 as a BPEL process interacting with six Web services. The Web services have been implemented in Java and for experimental purposes simulate certain influential factors (e.g., duration and output are made dependent on factors such as product types and amounts, and random behavior). For experimentation, we have deployed all these components on a single desktop PC. We define Order Fulfillment Lead Time as the KPI to be analyzed and define two checkpoints in the process (after “Check Stock” (i.e., both supplier and shipper can still be selected), and before “Shipment” thus allowing only the shipper to be selected). We create a set of overall 30 service candidates with different QoS characteristics (specified as mean values) and create a configuration which simulates the behavior of those services according to their QoS characteristics, but with deviations.

Tab. 2: Experimental Results

<table>
<thead>
<tr>
<th>Check Point</th>
<th>Learning Decision Tree Accuracy</th>
<th>Prediction and Prevention (200 instances per run)</th>
<th>KPI Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>No Need (predicted/measured)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Too late (predicted/meas.)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adapted (predicted/successful)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warehouse</td>
<td>88.2%</td>
<td>102/63</td>
<td>98/88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>108/105</td>
<td>92/90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>119/12/49</td>
</tr>
<tr>
<td>Shipment</td>
<td>94.7%</td>
<td>85/85</td>
<td>109/92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>105/103</td>
<td>90/88</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>157/20/23</td>
</tr>
</tbody>
</table>

KPI Dependency Tree Learning Phase: We trigger the execution of 500 process instances using a test client. For each of these instances we select the concrete supplier service and shipper service randomly in order to ensure that history data used for learning contains metrics data on each of these services and on most of their combinations. During process instance execution, the previously specified metrics are measured and saved in the metrics database. Then, for each checkpoint a decision tree is learned using the J48 algorithm [17]. The results and aspects to consider are as follows:

- **Duration:** The performance of the learning of a tree is about 15 seconds for 500 instances. As learning can be done in the background it does not affect the instance execution.
- **Accuracy:** The quality of the trained tree as a classification model can be assessed in terms of its accuracy, i.e., the percentage of correctly classified instances (from a test set) by the
model. This metric is provided by the decision tree algorithm after validation of the model (cross-validation) [17]. In general, the later the checkpoint is defined in the process the better the tree quality will be, because there are more known metrics that can be used for training the tree (see column tree accuracy in Table 2). In our approach, we assume that a classification model with a reasonably high accuracy has been created and do not take the accuracy into account in the following prediction and adaptation phases.

- **Historical Data**: The dependency tree is generated based on metric values resulting from historical process executions and corresponding adaptations. Metric effects resulting from adaptation actions which have never been used before are not reflected in the tree and would not later be considering during extraction of adaptation requirements. Thus, there should be a “bootstrapping” phase where several adaptations are performed either randomly (as in our experiments) or based on some other criteria in order to create historical instance data used for learning.

**Prediction and Adaptation Phase**: For the prediction and adaptation, we use two different constraints and preference models, one preferring lower cost, the other lower duration. For each model, we perform three experimental runs with 200 instances per run. The first run is performed using the default configuration (optimal according to the preferences model) without using the prediction and adaptation framework. In the other two runs, the prediction and (potential) prevention is performed at two different checkpoints. We evaluate for each instance what is predicted and whether the prediction has been correct (“measured”); this is done for the prediction types “No (Adaptation) Need” (predicted KPI class is “green” or “yellow”), “Too Late” (predicted KPI class is “red”), and “Adaptation Need” (instance tree has more than one leaf). The results are as follows (Table 2):

- **Duration and Cost of Adaptation**: The prediction and adaptation time together are below a second, thus making it only a performance impact factor for very short running processes. This duration metric and potentially other metrics reflecting the “cost of adaptation” could be modeled as “adaptable metrics” and given a weight in the preferences model. Then they would be taken into account during selection of an adaptation strategy.

- **Prevention Effectiveness**: The KPI performance (column “KPI Evaluation”) has been considerably improved by using our framework (run 2 and 3 outperform run 1). For example, for the first preference model the number of violations (KPI class = “red”) has been reduced from 64 to 49 and 23, respectively.

- **Effects of Preferences Model Settings**: The prevention effectiveness depends on settings in the preferences model and is in our case obviously much better when the preference is set on duration rather than cost. This is because substituted services are not always behaving as expected from their specified metric effects (i.e., not satisfying the corresponding AR predicates). Thus, when choosing services which just so satisfy the AR predicates, the risk of a violation of that AR is higher.

- **Effects of Check Point Positioning**: The later the checkpoint is chosen, the higher the prevention effectiveness as the prediction accuracy is higher. On the other hand, there is an increasing risk that it is too late to adapt (“Too Late” column). Of course, for even better performance, we could predict and adapt at both checkpoints for each process instance.

7. **Related Work**

In the area of process performance monitoring and analysis, most closely related to our approach is iBOM [4] which is a platform for monitoring and analysis of business processes based on machine learning techniques. It supports similar analysis mechanisms as in our approach such as decision trees, but does not deal with adaptation, i.e., extraction of adaptation requirements from the decision trees and derivation of adaptation strategies as in our approach. [18] presents an integrated KPI monitoring and prediction approach which uses machine learning techniques for prediction. It supports not only instance level KPI prediction as in our approach but also time series based prediction across process instances. It however does not deal with adaptation. We do not exploit information on process structure during dependency analysis, as the approach described in [2], but rely on machine learning algorithms to find those dependencies supporting not only numerical but also process data based metrics. [9] deals with prediction of numerical metric values based on artificial neural networks and introduces the concept of a checkpoint used for prediction which we have reused in our approach. Like us, [20] considers SBA layer dependencies for adaptation. While they support functional dependencies as well as the non-functional ones, they model the dependencies at design-time rather than extracting them through an analysis.
[10], [11] also cover the phases monitoring, prediction and adaptation as in our approach focusing on prevention of SLO violations by adapting the process via service substitution and fragment substitution, respectively. The best adaptation strategy is selected by performing a numerical KPI prediction for each adaptation strategy alternative separately and then selecting the one with the best prediction result. Our (analysis and) adaptation approach is different, as we use decision trees which as a white box classification model enable explicit extraction of adaptation requirements and strategies from the classification model. We in addition support adaptation in relation to several KPIs based on specified constraints and preferences.

There are several existing works in the context of QoS-aware service composition [6], [19] which describe how to create service compositions which conform to global and local QoS constraints taking into account process structure when aggregating QoS values of atomic services. We have reused concepts from those works when it comes to the definition of the constraints and preferences model and calculation of QoS scores [6]. Currently, we are simply enumerating all combinations of services when identifying adaptation strategies; that is only feasible for a small number of service types and could be optimized if needed as described, for example, in [6]. Furthermore, these approaches can be used for QoS-based adaptation by re-planning the service composition during monitoring [3]. In [1] the PAWS (Processes and Adaptive Web Services) framework is presented which takes into account local and global QoS constraints for selection of Web services at composition runtime. If at runtime a QoS requirement cannot be met, the framework chooses among a set of recovery actions such as retry, substitute, and compensate. Our approach is different in that we do not look at the process structure for prediction or for dependency analysis, but use machine learning techniques instead. This has the advantage that we support also process data-based metrics during analysis in addition to numerical metrics (such as duration and cost) and the approach is extensible towards taking account influential factors which go beyond process flow, e.g., infrastructure-level metrics.

8. Conclusions

In this paper we have presented an integrated monitoring, prediction and adaptation approach for preventing KPI violations in service compositions. At checkpoints, the KPI class of a running process instance is predicted based on the learned KPI dependency tree and metric data gathered for that instance until the checkpoint. In order to prevent KPI violations, adaptation requirements are extracted from the tree and then a set of alternative adaptation strategies is identified which can satisfy those requirements. The identified adaptation strategies are filtered and ranked according to constraints and preferences. The experimental evaluation of the approach has shown that the KPI violations are reduced and that the effectiveness in particular depends on the conformance of metric effect definitions (in the adaptation actions) to the adaptation requirement predicates and the related settings in the constraints and preferences model.

In our future work we will implement additional types of adaptation actions on different applications layers of a service-based application. In that context, one could think of infrastructural reconfigurations on the service layer. We will in particular address the cross-layer aspect by looking at how adaptation actions on different layers influence each other, e.g., a reconfiguration of the infrastructure has an impact on all services and process instances running on that infrastructure. That has to be taken into account during identification and selection of adaptation strategies.

Acknowledgments

The research leading to these results has received funding from the European Community’s 7th Framework Programme under the Network of Excellence S-Cube - Grant Agreement no. 215483.

References


This article should be cited as: