Distrib Parallel Databases
DOI 10.1007/s10619-013-7125-7

Data-driven and automated prediction of service level agreement violations in service compositions

Philipp Leitner · Johannes Ferner · Waldemar Hummer · Schahram Dustdar

Abstract Service Level Agreements (SLAs), i.e., contractually binding agreements between service providers and clients, are gaining momentum as the main discriminating factor between service implementations. For providers, SLA compliance is of utmost importance, as violations typically lead to penalty payments or reduced customer satisfaction. In this paper, we discuss approaches to predict violations a priori. This allows operators to take timely remedial actions, and prevent SLA violations before they have occurred. We discuss data-driven, statistical approaches for both, instance-level prediction (SLA compliance prediction for an ongoing business process instance) and forecasting (compliance prediction for future instances). We present an integrated framework, and numerically evaluate our approach based on a case study from the manufacturing domain.

Keywords Service composition · Service level agreements · Quality prediction

1 Introduction

Service-based applications and business process management have been blooming research areas for the last years, solving many fundamental problems of both, aca-

Communicated by Amit Sheth.

P. Leitner (✉) · J. Ferner · W. Hummer · S. Dustdar
Distributed Systems Group, Vienna University of Technology, Argentinierstrasse 8, Vienna, Austria
e-mail: leitner@infosys.tuwien.ac.at

J. Ferner
e-mail: johannes_ferner@yahoo.de

W. Hummer
e-mail: hummer@infosys.tuwien.ac.at

S. Dustdar
e-mail: sd@infosys.tuwien.ac.at

Published online: 04 April 2013

demic and industrial interest [37, 49]. Going forward, global trends like Everything-as-a-Service (XaaS) or Cloud Computing will further increase the spotlight put on services engineering and related disciplines [7]. In the wake of these developments, non-functional service aspects and Quality-of-Service (QoS) are becoming more relevant to industrial practice, where QoS promises are typically defined as legally binding Service Level Agreements (SLAs) [25]. SLAs are described using languages such as WSLA [10] or OGFs WS-Agreement [2], and contain so-called Service Level Objectives (SLOs), numerical QoS objectives, which the service needs to fulfill. For cases of violations, SLAs often define monetary penalties, e.g., a discount that the provider needs to grant to the client. Furthermore, frequent SLA violations detect from the service client's experience, and are damaging to the image of the provider. Hence, providers have a strong interest in reducing the number of SLA violations for their services.

Essentially, there are two approaches to achieve this. On the one hand, it is possible to use post mortem analysis and optimization [5, 51], i.e., analyzing historical cases of violation and modifying the service composition [12] (or the business process that the composition implements), so that the same type of violation is less likely to happen again. On the other hand, one can aim to predict violations in advance, before they have actually happened. This is more promising, as it allows providers to not only learn from past failures, but actually prevent them in the first place. In this paper, the main contribution is an extensive discussion of an approach to achieve the latter, runtime prediction of SLA violations. We distinguish between two types of SLOs, instance-level SLOs (which can be evaluated for each business process instance in isolation) and aggregated SLOs (which can only be evaluated over a defined period of time), and present data-driven statistical prediction techniques for both types of SLO. This paper is an extension of previously published work [28]. This earlier work introduced our approach for machine learning based prediction, while the current paper provides a more extensive discussion of the end-to-end framework, coverage of quantitative and aggregated SLOs, and numerical evaluation of the approach.

The remainder of this paper is structured as follows. Firstly, in Sect. 2, we introduce the domain of SLA management. Afterwards, in Sect. 3, the case study used in the remainder of the paper, is introduced. The main contributions of this paper are contained in Sect. 4 and Sect. 5. Section 4 introduces the general framework for predicting violations, while Sect. 5 details the used statistical models. Afterwards, Sect. 6 discusses a prototypical implementation of our techniques based on an existing infrastructure for service-oriented architectures (SOAs) [35], which forms the basis of the numerical evaluation discussed in Sect. 7. Section 8 gives an overview of earlier research, and puts our work into context with regard to the research area at large. Finally, we conclude the paper in Sect. 9, summarizing the main points of our contribution and outlining future research.

2 Background

In the following section, we introduce some important background and notation with regard to the concept of SLAs. SLAs [25] are a formalization and contractual arrangement of QoS for composed services. Instead of assuming that services provide

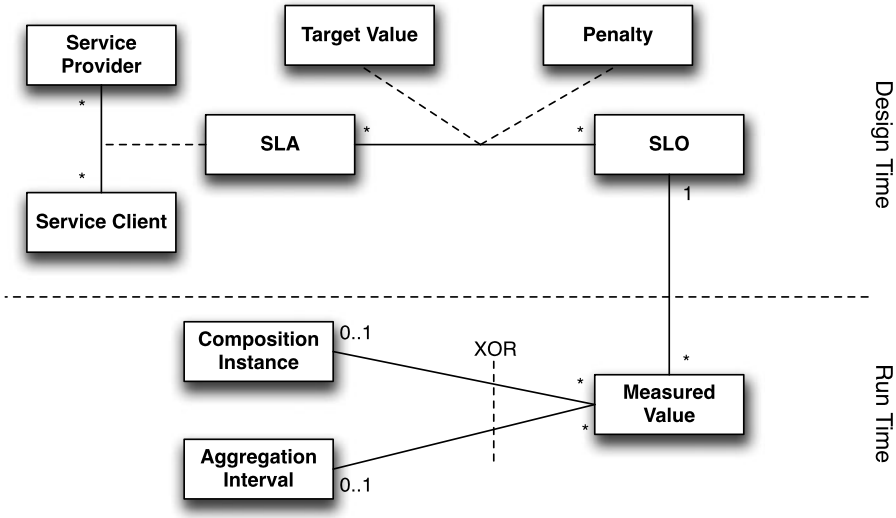

Fig. 1 Simplified SLA model

the highest quality they can on a best-effort basis, SLAs fix the minimally promised quality in various dimensions. SLAs are often seen as legally binding contracts between one or more service clients and a service provider. SLAs are a collection of many SLOs. An SLO is an observable quality dimension of a service. Additionally, SLAs define penalties for non-achievement (violation) of SLOs. Both, penalties and target values, can be different for every SLA in which an SLO is used. At runtime, concrete values for SLOs can be monitored. Based on the type of SLO (see below), this measured value can be generated either per composition instance or per aggregation interval. For clarity, the domain model of SLAs, as they are understood in this paper, is depicted in Fig. 1.

Some different languages have been proposed to model SLAs, including WSLA [10, 25], WS-Agreement [2] and SLAng [46]. These models do not differ so much in their expressiveness, but more in the environment they live in. For instance, WSLA specifies a monitoring and accounting infrastructure along with the basic language [10]. It is important to note that the work in this paper is agnostic with regard to the used SLA language, as long as it fits the basic model specified in Fig. 1.

SLOs come in different flavors. In this paper, two distinctions are of relevance. Firstly, one can differentiate between nominal and continuous SLOs. For nominal SLOs, the measured value can be one of a finite number of potential values. The target range (i.e., the domain of valid values as per the agreement) is a subset of the set of potential values. Metric SLOs, which are more prevalent, can take an infinite number of values. Target values are defined as thresholds on the metric. Secondly, one can distinguish SLOs on composition instance level and aggregated SLOs. For composition instance level SLOs, a decision of whether an SLA violation has happened can be made for every single composition instance individually. Aggregated SLOs are defined over an aggregation interval, for instance a number of composi-

tion instances or a time interval. Decisions can be made only looking at the whole aggregation interval, e.g., all composition instances of a month.

3 Illustrative use case

In the remainder of this paper, the case of a reseller of built-on-demand heavy-duty industry robots (ACMEBOT) will be used. We originally introduced ACMEBOT in [26], and the following discussion will mostly follow the same basic notions.

The business model of ACMEBOT is as follows. Customers request a quote (request for quotation, RFQ) for a specific robot in a specific configuration. ACMEBOT plans the steps necessary for the requested configuration, checks whether necessary parts are not in the internal warehouse (these parts need to be ordered from external suppliers), and generates an offer. The customer can then either cancel the process or place the order. If the order is placed, ACMEBOT starts by getting all missing parts from external suppliers and waits for these parts to arrive. As soon as all parts are available, the product is scheduled for assembling. After assembling is finished, the product is subject to a quality control step, and, if successful, shipped to the customer. In parallel to the shipping of the product, the customer is billed and an invoice is sent. ACMEBOT's core business process is depicted in Fig. 2.

For reasons of brevity, we concentrate on the two roles "Customer" and "ACMEBOT Assembly Service" in the figure, even though ACMEBOT interacts with many different external partners (e.g., suppliers of parts, shippers, credit card companies) to implement the described functionality. ACMEBOT's IT relies on the notion of SOA, i.e., the order process is implemented as a service composition. Hence, activities in the process are mapped to one or more invocations of (Web) services.

Generally, customers of ACMEBOT are larger business partners, typically resellers (i.e., ACMEBOT does not interact with end customers directly). For each reseller, there are pre-established SLAs, which typically consist of a subset of a number of often-seen SLOs. Some examples are depicted in Table 1, along with target values and penalties. Evidently, these are illustrative example values only, without claims of generality. Penalty payments are captured as discount on the current or future purchases. Time periods are given in working days. The SLO "Quality Control Positive" is of particular interest, as it is the only nominal SLO in the case study. The possible values of this SLO are simply `{Yes, No}`, with the target being `Yes`.

4 Data-driven prediction of SLA violations

In this section, we discuss our integrated, data-driven approach for predicting different types of SLOs in advance.

4.1 Framework overview

We sketch our overall framework for predicting SLA violations in Fig. 3. Essentially, three separate phases can be distinguished. Firstly, in the **monitoring phase**,

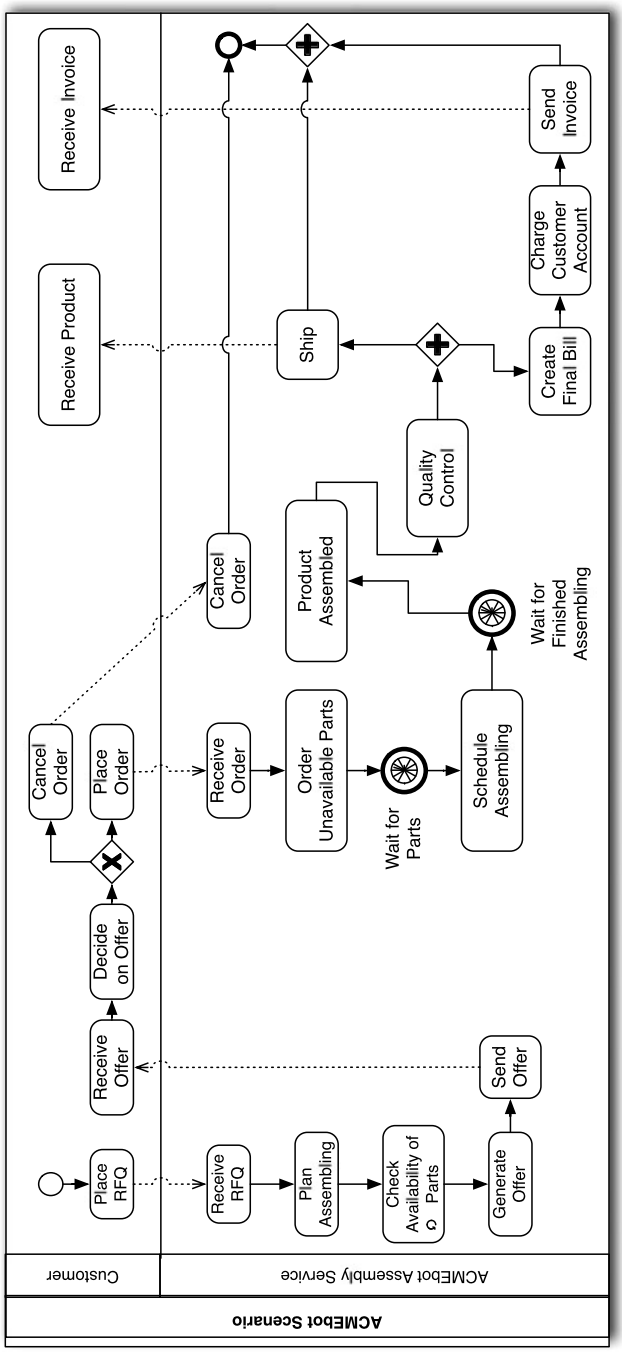

Fig. 2 Use case illustration (from [26])

Table 1 Example SLOs in ACMEBOT

Type of SLO		SLO	Target	Penalty	Interval
Instance-Level	Metric	Time To Offer	≤2 days	5 % discount	n/a
		Delivery Time	≤3 days	10 % discount	n/a
		End-To-End Duration	≤7 days	10 % discount	n/a
	Nominal	Quality Control Positive	Yes	80 % discount, or correction at provider's expense	n/a
Aggregated	Metric	Rate of Failed Service Requests	<1 %	10 % discount on next purchase	Weekly
		Service Availability	>99.99 %	10 % discount on next purchase	Monthly

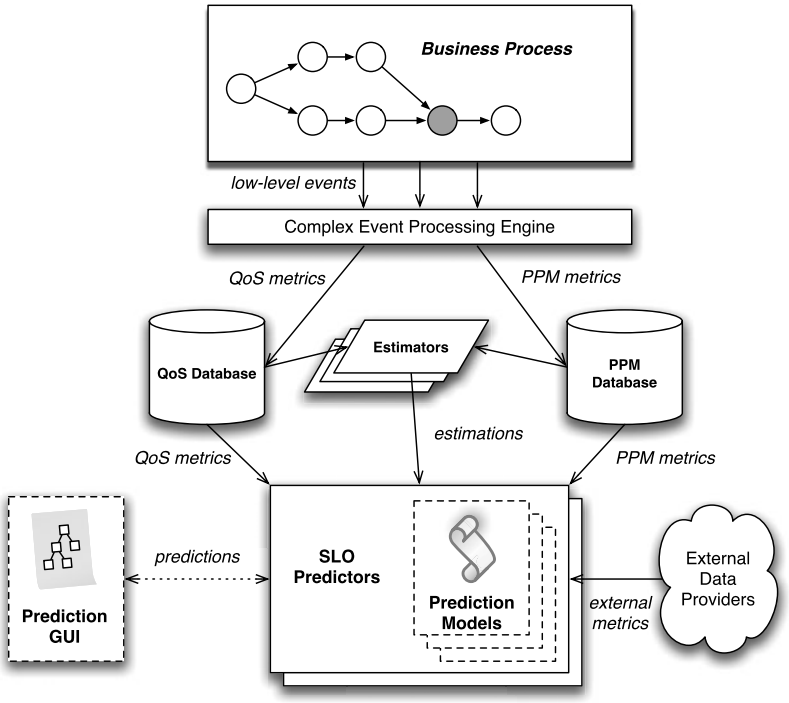

Fig. 3 Prediction approach overview

raw data is gathered from the running service composition, and transformed into so-called metrics (both, QoS and Process Performance Metrics, or PPMs, as will be explained below). We use the notion of complex event processing [30] (CEP) to extract higher-level knowledge from low-level monitoring events. These metrics are stored in databases, decoupling the monitoring parts of the system from the actual analysis. Secondly, in the **prediction phase**, the gathered high-level knowledge is enriched with external data, if available, as well as with so-called estimations. All

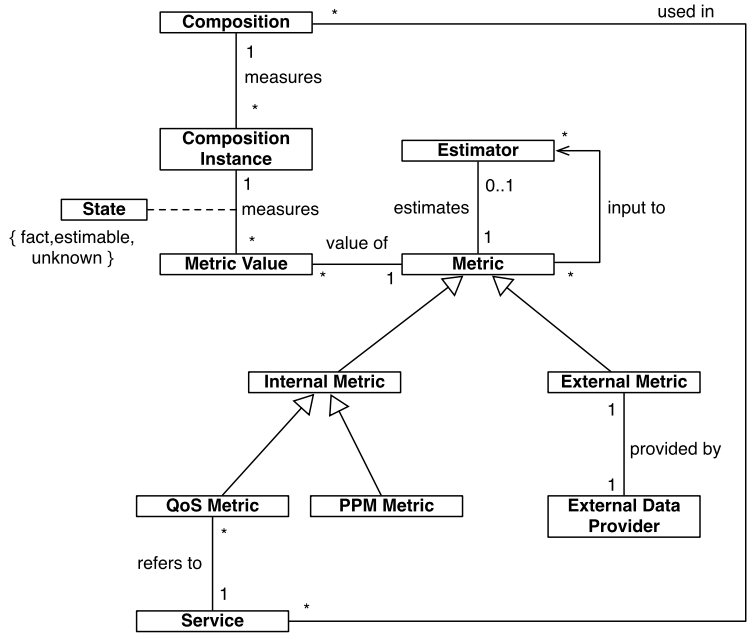

Fig. 4 Metrics data model

these different types of runtime data are then fed into a prediction model, essentially a pre-trained statistical model used for generating one or more predictions of concrete SLO values from data. Finally, in the **management phase**, the predictions of SLO values are matched to existing customer SLAs, and visualized in a dashboard for human operators, along with collected statistical information about the past performance of predictors.

In the following, we will focus mostly on the prediction and management phases. A deeper discussion of the monitoring is out of scope of this paper. The interested reader may refer to [27, 51] to find more information on our ideas with regard to CEP-based monitoring.

4.2 Input data and checkpoint definition

A vital ingredient of our approach are metrics. Basically, metrics is the umbrella term we use for all different types of data used as input of SLO predictions. We present an overview of the relevant data model in Fig. 4. Generally, we distinguish between internal and external metrics. Internal metrics can be monitored directly from the service composition, while external metrics originate in external data source, e.g., an external customer relations management service providing customer information. We do not go into more detail about external data providers. Internal metrics are further separated into QoS metrics, i.e., typical Web service QoS information [31], and PPMs [52]. PPMs are domain-specific metrics, which are often defined recursively, and which make business sense in the domain of the provider of the process. They

are the basis for defining the SLOs of a process. Examples of PPMs include simple metrics, such as the customer that has ordered a product, or the number of parts that are not in stock for a given order. A more complex PPM is the duration of the part ordering subprocess in the ACMEBOT case. QoS metrics are always associated to exactly one service used in the service composition, while PPMs are typically not bound to any concrete service.

Metrics can be monitored for each composition instance. At any point in the execution of a process, every metric can be in one of three states. (1) Input data can be available as **facts**. Facts represent data which is already known at this time. Generally speaking, this includes all data which can already be monitored at this point in the execution. Typical examples of facts are the QoS of services that have already been invoked. (2) Input data can be **unknown**. Unknowns are the logical opposites of facts, in that they represent data which is entirely unknown at this point in the composition instance. For instance, before starting the assembling process, the assembling time will be unknown. (3) Finally, as a kind of middle ground between facts and unknowns, input data can also be **estimable**. Such estimates are produced by dedicated estimator components, which are metric-specific. Generally, an estimator is a function that takes any number of other metrics (typically metrics, which are already available as facts) as input and produces an estimation for a yet unknown metric, to serve as input to prediction. Evidently, not all metrics are estimable. QoS metrics are often relatively easy to estimate, as techniques such as QoS monitoring [34] can be used to create an approximation of e.g., the response time of a service before it is actually invoked. As PPMs are generally domain-specific, it is often harder to estimate PPMs in advance. However, in many cases, this is possible as well. For instance, given the address of the customer and the selected shipping option, it is possible for domain experts to estimate the shipping time before even starting the shipping process.

It is obvious that one important decision for our approach is at which point in the service composition we want to predict violations. We refer to this decision point as the prediction checkpoint. Earlier checkpoints have access to less data, but the remaining time for reacting to prospective faults is higher. Conversely, generating predictions later increases the quality of the predictions, but reduces the time available for reaction. This tradeoff is visualized in Fig. 5. However, note that it is also possible to define multiple checkpoints for a given process, generating both early (but inherently unsure) warnings and more reliable late predictions.

4.3 Identification of factors of influence

Finding good prediction checkpoints at which the prediction is reasonably accurate and still timely enough to react to problems demands for some domain knowledge about influential factors of composition performance. Factors of influence are rarely obvious, even to domain experts. Hence, a process has been devised based on the principle of dependency analysis [50, 51], which can be used by business analysts to identify factors of influence. This process is summarized here.

Our approach for identifying factors of influence is a semi-automated process. It relies on the domain knowledge of a human business analyst, but supports her with automation and knowledge discovery tools to ease repetitive tasks. The high-level

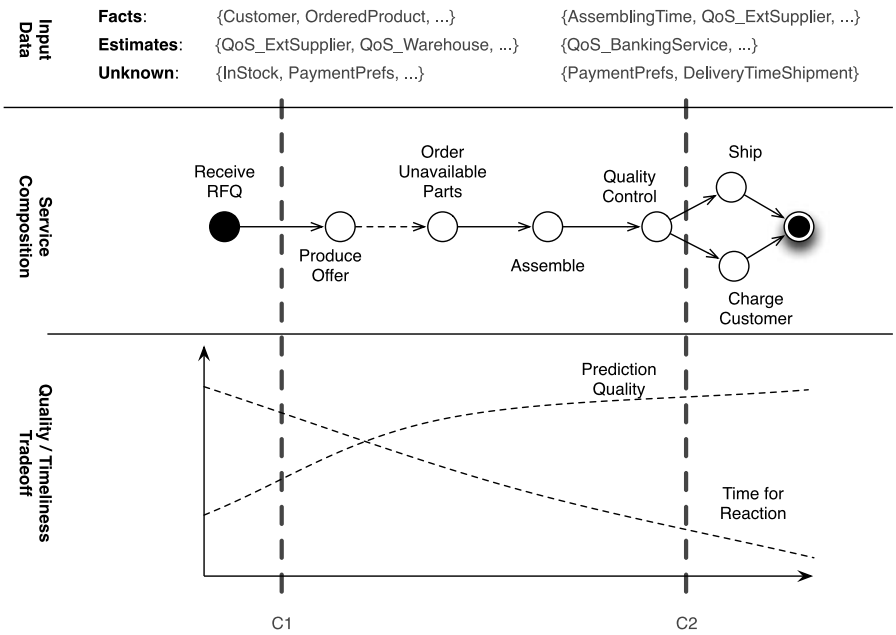

Fig. 5 Example checkpoints in the ACMEBOT process

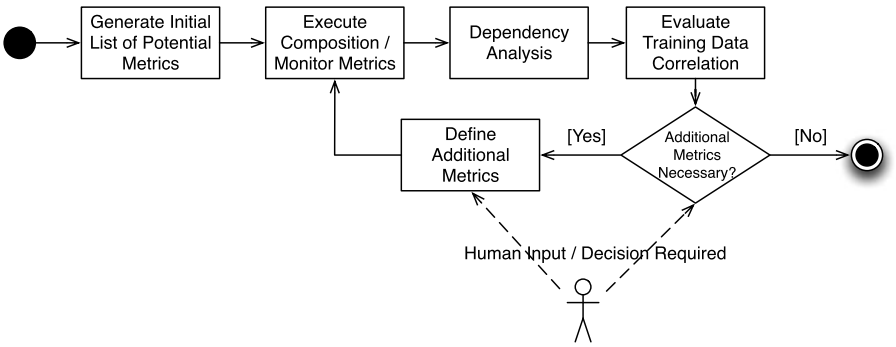

Fig. 6 Factors of influence analysis process

process is sketched in Fig. 6. As a first step, an (initial) list of potential factors of influence is defined. Typical QoS metrics can be generated automatically (e.g., for every used service, response time and availability metrics can be generated). Additionally, the business analyst may include PPMs and external data, which both need to be defined manually. For every potential factor of influence, a monitor is defined or generated, which specifies how this metric can be measured from a running instance. For instance, for QoS metrics, we define CEP expressions, which indicate how this metric can be calculated from service composition monitoring events. To give one trivial example, the execution time of an activity in the composition is the times-

tamp of the respective "activity finished" event minus the timestamp of the according "activity started" event. Secondly, a data set containing these metrics needs to be generated, either by simulating the composition in a Web service test environment, such as Genesis2 [24], or by monitoring real executions with monitoring of all potential factors of influence enabled. Afterwards, a so-called dependency tree is generated from the data set. The dependency tree is essentially a decision tree, containing the factors that best explain SLO violations in the composition. Generating the dependency tree boils down to training a decision tree using, e.g., the C4.5 [40] algorithm from the available data. The third step is then to use these factors, as contained in the dependency tree, to try and train a prediction model. If this prediction model has a sufficiently high training data correlation (see Sect. 4.4) against the measured data set, these factors can be used as input to the prediction model. If the correlation is not sufficient, the business analyst needs to identify the reason for the lacking performance. Generally, she will then go on to define additional metrics and their monitors, and repeat from the second step.

4.4 Prediction quality management

In the management phase, one essential activity is prediction quality management, i.e., the process of evaluating the performance of prediction models by comparing predictions with actual outcomes, as soon as they become available. As different statistical methods are used to predict different types of SLOs, we also need different quality indicators to evaluate their performance. Typically, if the prediction performance as measured by these quality indicators, is not sufficient, the prediction model needs to be re-trained, by taking into account new monitoring data, or additional factors of influence.

4.4.1 Quality indicators for metric SLOs

For metric SLOs (both, on instance-level and aggregated), we mainly use the training data correlation *corr*, the mean prediction error $\bar{e}$, and the prediction error standard deviation $\sigma_{\bar{e}}$ for quality management. *corr* is a standard machine learning approach to evaluate regression models, which captures the statistical correlation between the actual outcomes in the training data set, and the values that the predictor would generate if applied to the historical values. In our approach, *corr* is used mainly to evaluate freshly generated prediction models, when no actual runtime predictions have yet been carried out (e.g., as part of the dependency analysis process outlined in Sect. 4.3). This indicator is inherently overconfident, as during training all estimates are replaced for the facts that they estimate. However, a low training data correlation is an indication that important metrics are still unknown in the checkpoint, i.e., that the checkpoint may be too early in the composition to do much good. More important at runtime than *corr* is $\bar{e}$, as defined in Eq. (1).

$$\bar{e} = \frac{\sum_{i=0}^{n} |m_i - p_i|}{n} \tag{1}$$

$\bar{e}$ represents the average (Manhatten) difference between predicted and monitored values, i.e., how far "off" the prediction is on average. n is the total number of predictions, p_i is a predicted value, and m_i is the measured value to prediction p_i (that is,

every prediction is compared to the value that has been measured after this instance was finished). Finally, $\sigma_{\bar{e}}$ is used to describe the variability of $\bar{e}$, as defined in Eq. (2). e_i is the actual prediction error for an instance ($m_i - p_i$). Essentially, high $\sigma_{\bar{e}}$ means that the actual error for an instance can be much lower or higher than $\bar{e}$, which in turn makes the prediction less reliable for the end user.

$$\sigma_{\bar{e}} = \sqrt{\frac{\sum_{i=0}^{n}(e_i - \bar{e})^2}{n}} \tag{2}$$

4.4.2 Quality indicators for nominal SLOs

For nominal SLOs, we cannot just calculate a mean distance of predictions and actual SLO values. Instead, we need to look at each prediction individually. To this end, confusion matrices are often used. They are essentially tables with n rows and columns, where n is the number of different values that the SLO can have (in the example of the "Quality Control Positive" SLO from Sect. 3 n is therefore 2). Every cell xy in the confusion matrix contains the number of cases, in which the actual SLO value turned out to be x, and the prediction was y. Evidently, only the values on the main diagonal of the confusion matrix contain correct predictions, while all other cases represent some sort of prediction error.

$$F = 2 \cdot \frac{prec \cdot recall}{prec + recall} \tag{3}$$

While the confusion matrix visualization is helpful to get a quick impression of the prediction performance, it is hard for a human operator to objectively compare the performance of two predictors solely based on it. For these cases, it is better to aggregate the information contained in the confusion matrix, for instance using the F-measure, as defined in Eq. (3). In essence, the F-measure is the harmonic mean of precision and recall [43], which can be derived from the confusion matrix.

5 Statistical models

In the following, we discuss our concrete instantiations of different statistical models for prediction SLA violations. Depending on the type of SLO, we use one of three different approaches: (1) for nominal, instance-level SLOs, we use decision tree classifiers [39], (2) for metric, instance-level SLOs, we use Artificial Neural Networks (ANNs) [23], and (3) for metric, aggregated SLOs, we use Autoregressive Integrated Moving Average (ARIMA) models [6].

5.1 Prediction of nominal, instance-level service level objectives

Nominal SLOs are interesting, as many prediction approaches used in related work (e.g., resource analysis as discussed in [21]) are not well-suited to handle them. Essentially, predicting a nominal SLO is a classification problem. We use J54, the WEKA [17] implementation of the well-known C4.5 algorithm [22, 40]. Note that

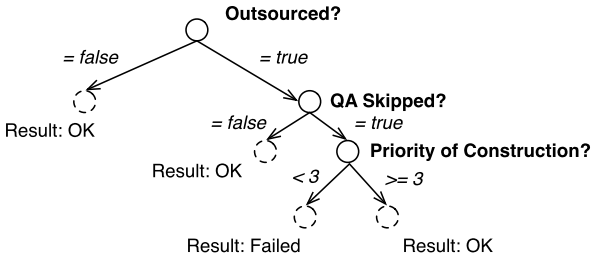

Fig. 7 Example decision tree for SLO quality control positive

it is also feasible to plug in other classification algorithms as implementations of the prediction models for nominal SLOs. We have also experimented with Bayes networks [16], but do not report on these results in this paper for reasons of brevity.

Decision trees [39] are an approach primarily used for classification of data. The decision tree is a directed, acyclic graph with one root node and any number of leaf nodes (nodes with an out-degree of 0). Every leaf node represents a classification to one of the classes, every other node represents a decision. When data has to be classified using the decision tree, one reads the tree from the root node downwards, always continuing with the edge indicated by the outcome of every decision evaluated against the data, until a leaf is reached. This leaf is the result of the classification. A primitive example of a decision tree in the SLO prediction domain is depicted in Fig. 7. This tree exemplifies, what the tree-based prediction model for the SLO Quality Control Positive (as introduced in Table 1) could look like.

5.2 Prediction of metric, instance-level service level objectives

There are inherently two quite different approaches to predict metric SLOs: (1) predict the concrete value of the metric SLO, and compare this predicted value with the target value defined in the SLA, or (2) predict violation or non-violation directly. The former is a regression problem, while the latter is a binary classification problem. If one chooses to follow the latter approach, decision trees can again be used to carry out predictions. However, we have decided to follow the former approach, as predicting the concrete SLO value provides much more fine-grained information to the end user. For instance, using our approach, users also get an idea "how much" the SLA will be violated, which may also be relevant to quantify the actual damage of the violation. Hence, a suitable regression algorithm for generating predictions of metric SLO values is required.

In our work, we use Artificial Neural Networks (ANNs) [23]. ANNs are a machine learning technique inspired by the inner workings of the human brain. Basically, ANNs consist of one or more layers of nodes (neurons) and directed connections between neurons. Oftentimes, ANNs consist of an input layer, output layer and one or many intermediary layers.

An abstracted example of a multilayer perceptron with one hidden (intermediary) layer for predicting the metric SLO Order Fulfillment Time is given in Fig. 8. Input values are mapped to nodes in the input layer. From these inputs, a complex system of weights and activation functions produces a single output value (the SLO). However, keep in mind that the number of intermediary layers is in general not fixed.

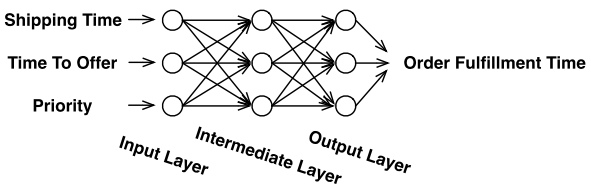

Fig. 8 Example network for predicting the order fulfillment time

5.3 Prediction of aggregated service level objectives

In order to predict aggregated SLOs, one needs to be able to predict how many future instances there will be in the remainder of the aggregation interval, as well as predict how many of those instances will violate the target value. For both prediction problems we can use the technique of time series analysis, i.e., the technique of predicting future values based on the observed values of a series of historical values. In this paper, we consider two different types of time series, namely, those with trends and those with seasonality. Time series with seasonality are stationary, i.e., the mean does not change over time. However, the concrete values change significantly with the "season". Time series with trends are non-stationary, i.e., their mean changes with time. If we model the evolution of an SLO's values as a stationary process, it means that the SLO value oscillates around a constant mean value and shows no significant trend in any direction away from the mean. We argue that stationarity is a valid assumption for most SLOs, at least in the short or medium run. However, over longer periods of time, it may occur that an SLO converges against a new mean value and shows a different oscillation scheme. Hence, an appropriate time interval for the stationary process has to be considered. ARIMA models [6, 45] have emerged as a popular approach to predict future values in time series, where the data show evidence of non-stationarity.

Similar to training of ANNs for instance-level SLOs, ARIMA models need to be fitted to training data before they can be used for prediction. We have devised a fitting process that uses historical instances of the service composition, however, details to this parameter optimization process are out of scope here. Please refer to [15] for more details.

6 Implementation

To evaluate the ideas discussed in this paper, we have implemented an end-to-end prototype dubbed E-dict (short for Event-driven prediction).

6.1 Implementation overview

E-dict is based on Microsoft .NET 3.0[1] tooling and integrates various other well-known third-party software packages. In this section, we detail the realization of our prototype.

[1] http://www.microsoft.com/download/en/details.aspx?id=31.

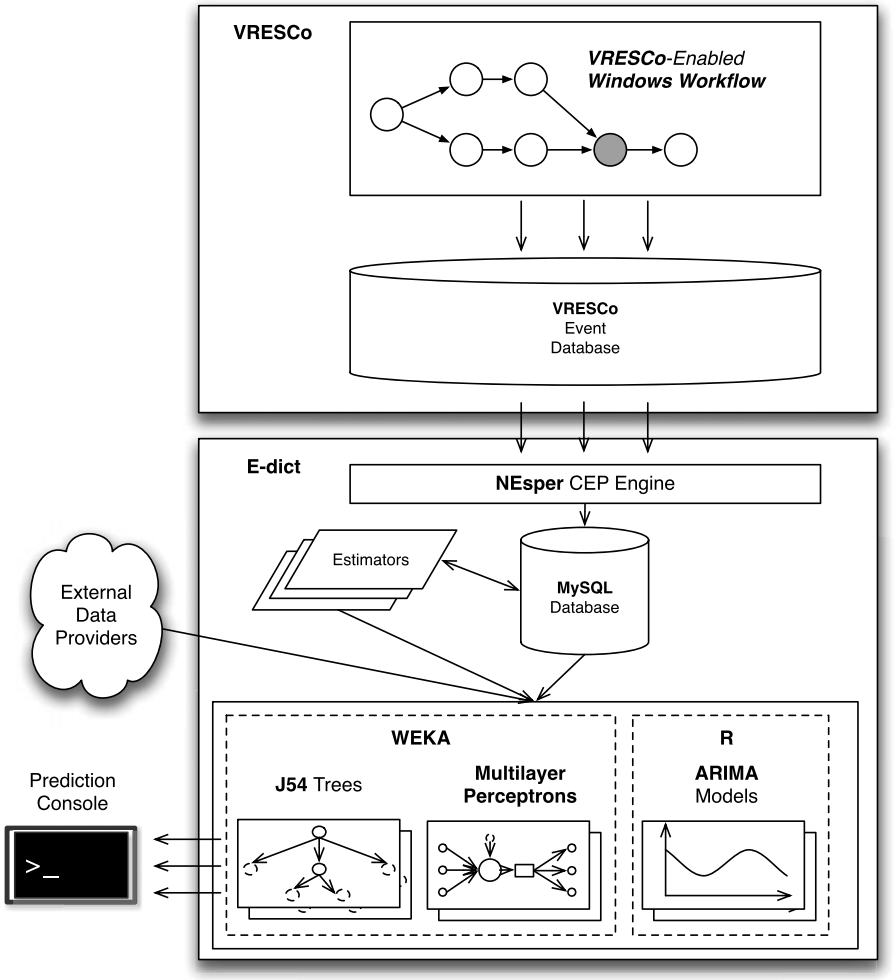

Fig. 9 Implementation overview

Figure 9 shows how we have practically implemented the conceptual framework discussed in Sect. 4. As far as possible, we aimed to prevent reproducing existing work. Hence, we used, adapted and integrated a plethora of state-of-the-art tools, algorithms and research prototypes. In the figure, we printed such existing tools and technology in **bold**.

As can be seen, E-dict is heavily based on an earlier research prototype, VRESCo (Vienna Runtime Environment for Service-Oriented Computing) [35]. VRESCo is a novel environment for building, running, and monitoring highly dynamic composite services based on Microsoft Windows Workflow Foundation (WWF) technology.[2] VRESCo allows for event-driven monitoring via the VRESCo event engine

[2] http://msdn.microsoft.com/en-us/netframework/aa663328.

(described in detail in [33]). The event data collected by this event engine is our main interface towards the E-dict framework, which uses the NEsper complex event processing engine[3] to generate interpretable metrics from the low-level monitoring data collected by VRESCo. NEsper is the .NET version of the state-of-the-art open source CEP engine Esper,[4] and functionally mostly equivalent. The generated metrics are then stored in a local database, for which we use the open source database management system MySQL 5.0 community edition.[5]

To implement the statistical algorithms described in Sect. 5, we utilize the well-known machine learning toolkit WEKA [17], which implements both, J54 decision trees (an open source implementation of C.45) and multilayer perceptrons. As WEKA is implemented in the Java programming language (while our E-dict tooling is .NET based), we wrapped WEKA using a RESTful Web service [42] to integrate the tool more easily with our prototype. To implement ARIMA models, we use the R statistical library [41] as foundation. As with WEKA, a service wrapper was necessary to seamlessly integrate R with the remainder of the prototype system. Finally, prediction results are reported to the user. At the moment, E-dict is commandline-based, hence, predictions are reported via a prediction console.

6.2 E-dict configuration

Configuration of E-dict happens mostly via XML configuration files. While a full discussion of the basis configuration is out of scope for this article, we will illustrate the basic configuration instruments.

Listing 1 exemplifies the XML-based definition of a metric. Metrics are defined as EPL (Esper Processing Language) statements on event streams, as received from the VRESCo event engine. The complex event(s) produced by this EPL statement can then be postprocessed in a number of ways. In the example, we retrieve some payload data from the complex event via the message path `GetPartListResult/Parts`. Afterwards, we apply a CS-Script[6] to the retrieved property. Using such scripts, we are able to implement complex transformations on the processed events. The result of the statement is of type integer. As part of E-dict, we provide some tooling to generate monitoring definitions of many often-used metrics (e.g., for the response time of each service used in the composition, or for each input and output message of each service invocation). However, generally, domain experts are required to write definitions for more domain-specific metrics (as the one depicted in Listing 1) manually.

Another relevant aspect of E-dict configuration is fine tuning of the statistical models used in the SLO predictor component. For WEKA, we allow machine learning savvy users to configure the machine learning model via the same parameters, which are also used by the WEKA commandline interface.

[3] http://esper.codehaus.org/about/nesper/nesper.html.

[4] http://esper.codehaus.org/.

[5] http://www.mysql.com/products/community/.

[6] http://www.csscript.net/.

```
<metric name="total_nr_of_items"
  type="integer"
  epl=
    "select
       _event as msg, _event.WorkflowId as id
     from
       AfterWFInvokeEvent _event
     where
       _event.ActivityName = 'get_parts_list'"
  messagePath="GetPartListResult/Parts"
  script="return (input as string).Split(';').Length;"
/>
```

Listing 1 Example metric definition in XML

Table 2 Base statistics for case study SLOs

(a) Order fulfillment time				(b) Quality control	
Min	Max	Mean	StdDev	True	False
28588 ms	49939 ms	37693 ms	4626	2687	2266

7 Evaluation

We base our evaluation of the E-dict approach on an implementation of the actual order handling part of the ACMEBOT case, i.e., the subflow starting with the activity "Receive Order" until the end of the business process. The case study has been implemented using .NET Windows Communication Foundation[7] (WCF) technology. A MySQL 5 database is used as data backend for VRESCo, and all necessary components are deployed on the same Windows server machine, in order to reduce the impact of external influences, such as network latency. The service composition itself has been implemented as a dynamic service composition using Microsoft WWF. The technical implementation of this case encompasses more than 40 activities. These activities are monitored using VRESCo, and 40 different metrics are generated for each process instance using the CEP engine. These metrics include both QoS information, such as the response time of every used Web service, and PPMs, such as customer identifiers or which products have been ordered. These metrics form the foundation, from which we will generate concrete predictions for SLOs.

For reasons of brevity, we focus on instance-level SLOs in our evaluation. Some numerical results with regards to prediction performance for aggregated SLOs can be found in [15]. To evaluate the prediction mechanisms for instance-level SLOs, we have used two representative SLOs, the Order Fulfillment Time (i.e., the end-to-end execution duration of the implemented process) as metric SLO, and Quality Control (`true` if the quality was deemed acceptable, `false` otherwise) as nominal SLO.

In order to bootstrap our data-based prediction, we have initialized the system with up to 5000 historical executions of the ACMEBOT process. We have summarized the basic statistics of this bootstrap data in Table 2a for the SLO Order Fulfillment Time and Table 2b for the SLO Quality Control.

[7] http://msdn.microsoft.com/en-us/library/ms735967(VS.90).aspx.

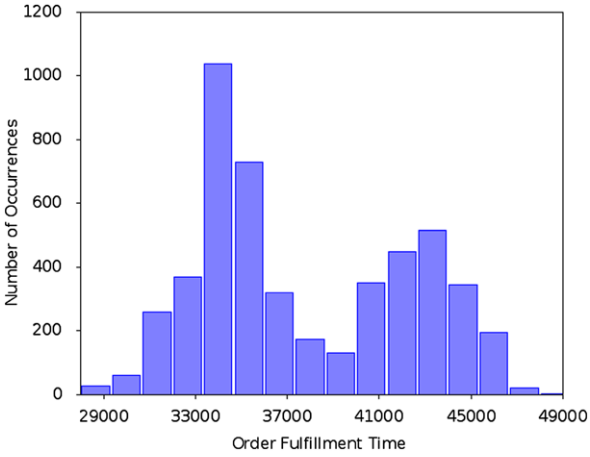

Fig. 10 Histogram for SLO order fulfillment time

Table 3 Overhead for training and prediction using ANNs

(a) Training overhead

Instances	Training [ms]
250	24436
500	49168
1000	96878
2500	240734
5000	481101

(b) Prediction overhead

Instances	Prediction [ms]
250	59
500	70
1000	82
2500	126
5000	209

Furthermore, Fig. 10 visualizes the distribution of SLO values for Order Fulfillment Time as histogram. For illustrative purposes, we have discretized continuous values into discrete ranges.

7.1 Performance overhead

Two different factors are relevant for quantifying the performance overhead of prediction. Firstly, we need to evaluate the time necessary for building the used prediction models (training time). Secondly, we need to measure how long the actual runtime usage (prediction) of these models takes. In Table 3, these measures are depicted for ANNs (in milliseconds). Evidently, the training time depends linearly on the number of historical instances that are available. Furthermore, it can be seen that ANN training takes a significant amount of time, even for relatively modest training data sizes. However, considering that model rebuilding can be done sporadically and offline, this large training times seem acceptable. Additionally, after the initial construction of the first prediction model, there is no time when no model is available at all. Instead, whenever retraining becomes necessary, the new model is trained offline, and exchanged for the old model as soon as training is finished.

As a second interesting performance aspect, the prediction time, i.e., the time necessary to produce predictions at runtime, has been measured. Table 3b again sketches

Table 4 Overhead for training and prediction using decision trees

(a) Training overhead		(b) Prediction overhead	
Instances	Training [ms]	Instances	Prediction [ms]
250	183	250	14
500	219	500	14
1000	258	1000	14
2500	460	2500	14
5000	623	5000	13

Table 5 Prediction quality of SLO order fulfillment time

corr	$\bar{e}$	$\sigma_{\bar{e}}$
0.98	767	581

these measurements for ANNs. Note that this overhead is more significant for the runtime performance than the training time, as it refers to the online part of prediction. Fortunately, the results depicted show that the absolute time necessary for prediction (roughly between 60 and 200 ms) is rather small.

Table 4 provides the same data for J54 decision trees, as used to predict nominal SLOs. Firstly, Table 4a shows that the time required for training decision trees is much smaller than for ANNs. Indeed, even though the training time is again increasing linearly with the number of available historical instances, the absolute time required for constructing the tree stays well below 700 ms even for 5000 instances. Using these trees for runtime prediction is equally efficient (Table 4b). We measured a constant prediction time of approximately 14 ms, independent of the amount of data used to train the tree in the first place.

7.2 Quality of prediction

Even more important than the time necessary to generate prediction models or runtime predictions is the quality of these predictions. To visualize the prediction quality in the ACMEBOT case, Fig. 11 plots for 100 distinct instances the predicted value for the Order Fulfillment Time (x) and the respective measured value (+). As can be seen, predictions and measured values are generally close, i.e., the prediction quality is good. However, for some isolated instances, the prediction is off by a larger margin (e.g., instances 82 or 91).

This more intuitive feeling for prediction quality is further substantiated in Table 5, which contains the three basic quality indicators for prediction as defined in Sect. 4 (training data correlation *corr*, prediction error $\bar{e}$, and prediction error variance $\sigma_{\bar{e}}$). With 0.98, *corr* is very high, indicating that the model might even be slightly overfitted to the training data. The actual prediction error $\bar{e}$ is 767 (or about 2 % of the mean process duration), with a $\sigma_{\bar{e}}$ of 581, which indicates a generally good prediction quality, but a relatively high variation of the error.

Furthermore, we have evaluated the quality of predictions of nominal SLOs based on the SLO Quality Control. We provide the confusion matrix for this SLO in Table 6a (again based on predictions and actually measured values for 100 instances

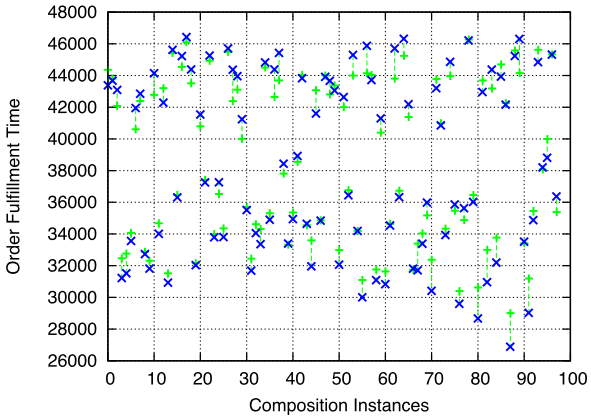

Fig. 11 Predicted and actual values for SLO order fulfillment time

Table 6 Prediction quality of SLO quality control

(a) Confusion matrix

		Predicted	
		true	false
Actual	true	53	2
	false	4	41

(b) Aggregated metrics

prec	*recall*	*F*
0.9399	0.9404	0.9402

of the service composition). As we can see, only 6 of 100 instances have been predicted inaccurately, 4 of which have been false positives (i.e., they were predicted as violating the SLA, but have not actually done so).

Furthermore, we provide aggregated quality indicators for this SLO in Table 6b. Precision, recall and F are in the area of 0.94, indicating a very strong overall prediction accuracy.

8 Related work

The problem discussed in this paper has been subject to a number of research papers in the past. In this section, we briefly introduce this earlier research, and compare it with our own contributions.

In general, even the earliest work on (Web) service level agreements, for instance SLAng [46], WSLA [10] or WSOL [47, 48], has acknowledged the fact that runtime management has to be a central element of every SLA infrastructure. For the service provider, important runtime SLA management tasks include (1) monitoring of SLOs (to decide if SLAs have been violated in the past), (2) analysis of past violations (to improve the business process, so that those violations can be prevented for the future), and, finally, (3) prediction of future violations before they have happened.

SLA monitoring is strongly related to QoS monitoring, as SLOs can often be broken down into lower-level QoS metrics. A plethora of different patterns can in principle be applied to QoS monitoring [36], but, in the context of SLA management, event-driven monitoring [53] can be considered state of the art. In our work, we also use an

event-driven monitoring approach to gather the metrics required for prediction. Our monitoring approach is based on the VRESCo SOA infrastructure [35], more concretely, on the eventing features of VRESCo [33]. In this paper, a deeper description of metric monitoring was out of scope, but details can be reviewed in [27].

Post-mortem analysis of SLA violations is still an active research area. For instance, Mode4SLA has recently been proposed as a framework to explain [4] and, consequently, prevent [5] SLA violations. Mode4SLA follows a top-down approach, and identifies dependencies of SLAs on the performance of the underlying base services used in a service composition. A similar approach is also followed by the dependency analysis approach discussed in [50, 51]. This work is particularly interesting in the context of this paper, as we have reused dependency analysis as a foundational piece of our factors of influence identification process (see Sect. 4.3). Finally, another important area of research in post-mortem SLA analysis is the mapping of lower-level QoS metrics to SLAs. One approach researching this kind of dependencies is the FoSSI project [13, 14], which deals mostly with SLAs in a cloud computing context.

A priori identification of SLA violations is still mostly a research topic (as opposed to SLA monitoring, which has by now started to diffuse into industrial practice), even though initial work in the area dates back to 2002 [9, 44]. This work uses a data-driven approach similar to the one discussed in this paper, but mostly stays on an abstract level, without discussing concrete application areas, algorithms or metrics. Others have consequently worked on prediction for concrete domains, e.g., finish time prediction for (human-centric) business processes [11] or the performance of service-based applications deployed on top of an enterprise service bus (ESB) [29]. More recently, other researchers have started to pick up on the idea of SLA prediction for service compositions, including [28], which the current paper is based on. Hence, some of our basic notions, as well as our general approach towards predicting metric, instance-level SLOs, have originated in [28]. A similar approach has also been discussed by [54]. These papers use a statistical approach similar to the current paper, even though the research discussed in the latter paper focuses more on event-based monitoring as an input to prediction than the actual prediction process. Contrary, others have used significantly different data-driven prediction approaches, based, for instance, on online testing [18, 32] or static analysis [21].

There is also some related research with regard to the application of time series models to performance prediction, even if mostly for other domains. For instance, [3] utilizes ARMA models to predict service quality in call centers. Nominal, aggregated SLOs were not discussed in this paper, mostly because of their limited practical relevance. However, existing literature knows of approaches to apply time series analysis to nominal values (e.g., [38]), should nominal, aggregated SLOs become practically relevant. Only very recently, some other authors have started to explore the field with applications of ARIMA and GARCH (generalized autoregressive conditional heteroskedasticity) models to predict aggregated SLAs [1].

Finally, our work is also related to testing of service-based systems. A good overview over this important field is given in [8]. In the past, we have also introduced an approach for upfront testing of service compositions to identify integration problems, and, hence, avoid SLA violations [19]. A similar approach was also dis-

cussed in [20], where root causes of functional service incompatibilities are detected using decision tree technology.

9 Conclusions

Identifying potential cases of SLA violations will increasingly become a major priority of service providers around the world. Detecting violations before they have happened allows providers to proactively take countermeasures, reducing the loss associated with violations. In this paper, we have presented a general framework for predicting SLA violations in a business process implemented as a service composition. Our approach is grounded in the usage of statistical models for data-driven prediction. Different types of SLOs ask for different prediction models: metric SLOs on instance-level are predicted using ANN-based regression, aggregated metric SLOs are covered using ARIMA models, an implementation of time series analysis, and, finally, nominal SLOs are tackled via decision trees. We demonstrated the applicability of these techniques to the problem of SLA violation prediction via numerical experiments.

While this paper presents self-contained research with little open ends, some practical problems remain. Firstly, we do not go into much detail about the concept of estimators. For some factors of influence, generating estimations is trivial (e.g., for service response times). As part of our future work, we plan to research methods and techniques for generating estimations in non-trivial cases. We conjure that it should be possible to recursively use the same approach that we use for generating predictions for estimation of factors of influence, as the problem is clearly strongly related. Secondly, one limitation of the current approach is that no notion of "doubt" about a prediction exists. That is, while our models will always generate a prediction, it is not clear how trustworthy this prediction really is. The prediction error standard deviation $\sigma_{\bar{e}}$ helps, but obviously the concrete uncertainty can be much lower or higher for single predictions. We need to investigate means to generate not only the predictions, but also associated confidence intervals.

Acknowledgements The research leading to these results has received funding from the European Community's Seventh Framework Program [FP7/2007-2013] under grant agreement 257483 (Indenica), as well as from the Austrian Science Fund (FWF) under project references P23313-N23 (Audit 4 SOAs).

References

1. Amin, A., Colman, A., Grunske, L.: An approach to forecasting QoS attributes of web services based on ARIMA and GARCH models. In: Proceedings of the 2012 IEEE International Conference on Web Services, pp. 74–81. IEEE Computer Society, Washington, DC (2012). doi:10.1109/ICWS.2012.37
2. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web Services Agreement Specification (WS-Agreement). Tech. rep., Open Grid Forum (OGF) (2006). http://www.gridforum.org/documents/GFD.107.pdf, Last Visited: 2011-07-19
3. Balaguer, E., Palomares, A., Soria, E., Martín-Guerrero, J.D.: Predicting service request in support centers based on nonlinear dynamics, ARMA modeling and neural networks. Expert Syst. Appl. **34**(1), 665–672 (2008). doi:10.1016/j.eswa.2006.10.003

4. Bodenstaff, L., Wombacher, A., Reichert, M., Jaeger, M.: Monitoring dependencies for SLAs: the MoDe4SLA approach. In: Proceedings of the 2008 IEEE International Conference on Services Computing (SCC'08), pp. 21–29. IEEE Computer Society, Washington, DC (2008). http://portal.acm.org/citation.cfm?id=1447562.1447847. doi:10.1109/SCC.2008.120
5. Bodenstaff, L., Wombacher, A., Reichert, M., Jaeger, M.C.: Analyzing impact factors on composite services. In: Proceedings of the 2009 IEEE International Conference on Services Computing (SCC'09), pp. 218–226. IEEE Computer Society, Los Alamitos (2009)
6. Box, G.E.P., Jenkins, G.M.: Time Series Analysis—Forecasting and Control. Holden-Day (1976)
7. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. **25**(6), 599–616 (2009). doi:10.1016/j.future.2008.12.001
8. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: QoS-aware replanning of composite web services. In: Proceedings of the IEEE International Conference on Web Services (ICWS'05), pp. 121–129. IEEE Computer Society, Washington, DC (2005). doi:10.1109/ICWS.2005.96
9. Castellanos, M., Casati, F., Dayal, U., Shan, M.C.: Intelligent management of SLAs for composite web services. In: Databases in Networked Information Systems (2003)
10. Dan, A., Davis, D., Kearney, R., Keller, A., King, R.P., Kuebler, D., Ludwig, H., Polan, M., Spreitzer, M., Youssef, A.: Web services on demand: WSLA-driven automated management. IBM Systems Journal **43**, 136–158 (2004). doi:10.1147/sj.431.0136
11. Dongen, B.F., Crooy, R.A., Aalst, W.M.: Cycle time prediction: when will this case finally be finished? In: Proceedings of the 2008 OTM Confederated International Conferences, pp. 319–336. Springer, Berlin (2008)
12. Dustdar, S., Schreiner, W.: A survey on web services composition. International Journal of Web and Grid Services **1**(1), 1–30 (2005)
13. Emeakaroha, V.C., Brandic, I., Maurer, M., Dustdar, S.: Low level metrics to high level slas - lom2his framework: bridging the gap between monitored metrics and sla parameters in cloud environments. In: Proc. Int High Performance Computing and Simulation (HPCS) Conf, pp. 48–54 (2010). doi:10.1109/HPCS.2010.5547150
14. Emeakaroha, V.C., Netto, M.A.S., Calheiros, R.N., Brandic, I., Buyya, R., De Rose, C.A.F.: Towards autonomic detection of SLA violations in cloud infrastructures. Future Gener. Comput. Syst. (2011). doi:10.1016/j.future.2011.08.018
15. Ferner, J.: Using Time Series Analysis for Predicting Service Level Agreement Violations in Service Compositions. Master's thesis, Vienna University of Technology (2012)
16. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Machine Learning **29**, 131–163 (1997). http://portal.acm.org/citation.cfm?id=274158.274161. doi:10.1023/A:1007465528199
17. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. SIGKDD Explorations **11**(1), 10–18 (2009). http://portal.acm.org/citation.cfm?id=1656274.1656278. doi:10.1145/1656274.1656278
18. Hielscher, J., Kazhamiakin, R., Metzger, A., Pistore, M.: A framework for proactive self-adaptation of service-based applications based on online testing. In: Proceedings of the 1st European Conference on Towards a Service-Based Internet (ServiceWave'08), pp. 122–133. Springer, Berlin (2008)
19. Hummer, W., Raz, O., Shehory, O., Leitner, P., Dustdar, S.: Testing of Data-Centric and Event-Based Dynamic Service Compositions. Softw. Test. Verif. Reliab. (2013, to appear)
20. Inzinger, C., Hummer, W., Satzger, B., Leitner, P., Dustdar, S.: Identifying incompatible service implementations using pooled decision trees. In: 28th ACM Symposium on Applied Computing (SAC'13), DADS Track (2013)
21. Ivanovic, D., Carro, M., Hermenegildo, M.: An initial proposal for data-aware resource analysis of orchestrations with applications to predictive monitoring. In: Proceedings of the 2009 International Conference on Service-Oriented Computing (ICSOC'09), pp. 414–424. Springer, Berlin (2009). http://portal.acm.org/citation.cfm?id=1926618.1926662
22. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan-Kaufmann, San Mateo (1993)
23. Jain, A.K., Mao, J., Mohiuddin, K.M.: Artificial neural networks: a tutorial. IEEE Computer **29**, 31–44 (1996). doi:10.1109/2.485891
24. Juszczyk, L., Dustdar, S.: Script-based generation of dynamic testbeds for SOA. In: Proceedings of the 2010 IEEE International Conference on Web Services (ICWS'10), pp. 195–202. IEEE Computer Society, Washington, DC (2010). doi:10.1109/ICWS.2010.75
25. Keller, A., Ludwig, H.: The WSLA framework: specifying and monitoring service level agreements for web services. Journal on Network and Systems Management **11**, 57–81 (2003). http://portal.acm.org/citation.cfm?id=635430.635442. doi:10.1023/A:1022445108617

26. Leitner, P., Hummer, W., Dustdar, S.: Cost-based optimization of service compositions. IEEE Trans. Serv. Comput. **99** (2011). http://doi.ieeecomputersociety.org/10.1109/TSC.2011.53
27. Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S.: Monitoring, prediction and prevention of SLA violations in composite services. In: Proceedings of the IEEE International Conference on Web Services (ICWS'10), pp. 369–376. IEEE Computer Society, Los Alamitos (2010)
28. Leitner, P., Wetzstein, B., Rosenberg, F., Michlmayr, A., Dustdar, S., Leymann, F.: Runtime prediction of service level agreement violations for composite services. In: Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC'09), pp. 176–186. Springer, Berlin (2009). http://portal.acm.org/citation.cfm?id=1926618.1926639
29. Liu, Y., Gorton, I., Zhu, L.: Performance prediction of service-oriented applications based on an enterprise service bus. In: Proceedings of the 31st Annual International Computer Software and Applications Conference, COMPSAC'07, vol. 01, pp. 327–334. IEEE Computer Society, Washington, DC (2007). doi:10.1109/COMPSAC.2007.166
30. Luckham, D.: The Power of Events: an Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Reading (2002)
31. Menascé, D.A.: QoS issues in web services. IEEE Internet Computing **6**(6), 72–75 (2002). doi:10.1109/MIC.2002.1067740
32. Metzger, A., Sammodi, O., Pohl, K., Rzepka, M.: Towards pro-active adaptation with confidence: augmenting service monitoring with online testing. In: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'10), pp. 20–28. ACM, New York (2010). doi:10.1145/1808984.1808987
33. Michlmayr, A., Rosenberg, F., Leitner, P., Dustdar, S.: Advanced event processing and notifications in service runtime environments. In: Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08), pp. 115–125. ACM, New York (2008). doi:10.1145/1385989.1386004
34. Michlmayr, A., Rosenberg, F., Leitner, P., Dustdar, S.: Comprehensive QoS monitoring of web services and event-based SLA violation detection. In: Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing (MWSOC'09), pp. 1–6. ACM, New York (2009)
35. Michlmayr, A., Rosenberg, F., Leitner, P., Dustdar, S.: End-to-end support for QoS-aware service selection, binding, and mediation in VRESCo. IEEE Transactions on Services Computing **3**, 193–205 (2010)
36. Oberortner, E., Zdun, U., Dustdar, S.: Patterns for measuring performance-related QoS properties in distributed systems. In: Proceedings of the 17th Conference on Pattern Languages of Programs (PLoP) (2010)
37. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: state of the art and research challenges. IEEE Computer **40**(11), 38–45 (2007)
38. Pruscha, H., G"ottlein, A.: Forecasting of categorical time series using a regression model. Economic Quality Control **18**(2), 223–240 (2003). http://www.heldermann-verlag.de/eqc/eqc18/eqc18014.pdf
39. Quinlan, J.R.: Induction of decision trees. Machine Learning **1**, 81–106 (1986)
40. Quinlan, J.R.: Improved use of continuous attributes in C4.5. Journal of Artificial Intelligence Research **4**, 77–90 (1996)
41. R Development Core Team: R: a Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna (2008). http://www.R-project.org. ISBN 3-900051-07-0
42. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly (2007)
43. Rijsbergen, C.J.V.: In: Information Retrieval. Butterworths, Stoneham (1979)
44. Sahai, A., Machiraju, V., Sayal, M., Moorsel, A.P.A.V., Casati, F.: Automated SLA monitoring for web services. In: Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM) (2002)
45. Shumway, R.H., Stoffer, D.S.: Time Series Analysis and Its Applications. Springer, Berlin (2010)
46. Skene, J., Lamanna, D.D., Emmerich, W.: Precise service level agreements. In: Proceedings of the 26th International Conference on Software Engineering (ICSE'04), pp. 179–188. IEEE Computer Society, Washington, DC (2004). http://portal.acm.org/citation.cfm?id=998675.999422
47. Tosic, V., Ma, W., Pagurek, B., Esfandiari, B.: Web service offerings infrastructure (WSOI)—a management infrastructure for XML web services. In: Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS'04), pp. 817–830 (2004)
48. Tosic, V., Pagurek, B., Patel, K., Esfandiari, B., Ma, W.: Management applications of the web service offerings language (WSOL). Information Systems **30**(7), 564–586 (2005). doi:10.1016/j.is.2004.11.005

49. Van Der Aalst, W.M.P., Hofstede, A.H.M.T., Weske, M.: Business process management: a survey. In: Proceedings of the 2003 International Conference on Business Process Management, BPM'03, pp. 1–12. Springer, Berlin (2003). http://dl.acm.org/citation.cfm?id=1761141.1761143
50. Wetzstein, B., Leitner, P., Rosenberg, F., Brandic, I., Dustdar, S., Leymann, F.: Monitoring and analyzing influential factors of business process performance. In: Proceedings of the 13th IEEE International Conference on Enterprise Distributed Object Computing (EDOC'09), pp. 118–127. IEEE Press, Piscataway (2009). http://portal.acm.org/citation.cfm?id=1719357.1719370
51. Wetzstein, B., Leitner, P., Rosenberg, F., Dustdar, S., Leymann, F.: Identifying influential factors of business process performance using dependency analysis. Enterprise Information Systems **4**(3), 1–8 (2010)
52. Wetzstein, B., Strauch, S., Leymann, F.: Measuring performance metrics of WS-BPEL service compositions. In: Proceedings of the Fifth International Conference on Networking and Services (ICNS'09). IEEE Computer Society, Los Alamitos (2009)
53. Zeng, L., Lei, H., Chang, H.: Monitoring the QoS for web services. In: Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07), pp. 132–144. Springer, Berlin (2007)
54. Zeng, L., Lingenfelder, C., Lei, H., Chang, H.: Event-driven quality of service prediction. In: Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC'08), pp. 147–161. Springer, Berlin (2008)