2010 IEEE International Conference on Web Services

Monitoring, Prediction and Prevention of SLA Violations in Composite Services

Philipp Leitner[1], Anton Michlmayr[1], Florian Rosenberg[2], Schahram Dustdar[1]

[1]Distributed Systems Group
Vienna University of Technology
Argentinierstrasse 8/184-1, 1040 Vienna, Austria
lastname@infosys.tuwien.ac.at

[2]CSIRO ICT Centre
GPO Box 664, Canberra ACT 2601, Australia
florian.rosenberg@csiro.au

Abstract

We propose the PREvent framework, which is a system that integrates event-based monitoring, prediction of SLA violations using machine learning techniques, and automated runtime prevention of those violations by triggering adaptation actions in service compositions. PREvent improves on related work in that it can be used to prevent violations ex ante, before they have negatively impacted the provider's SLAs. We explain PREvent in detail and show the impact on SLA violations based on a case study.

I. Introduction

The paradigm of service-oriented computing [18] has changed the way companies do business. Nowadays, many companies are shifting towards the Software-as-a-Service (SaaS) model, providing coarse-grained valued-added services as a composition of existing Web Services [10]. In this context the concept of Service Level Agreements (SLAs), which are contracts between providers of composite services and their customers [7], is very important. SLAs govern the quality that customers can expect from the service. Quality requirements are formulated as a collection of Service Level Objectives (SLOs), which are numerical target values, and penalties for not fulfilling these objectives. For the service provider it is therefore vital to minimize cases of SLA violation (i.e., cases where one or more objective could not be fulfilled).

Currently, most research focuses on finding the reason for SLA violations *ex post* (after the violation has happened, e.g., [3], [23]). While this is certainly useful for later analysis and improving the composition it does not directly help preventing violations (i.e., the damage has already been done). In this work we present an *ex ante* approach, which allows for runtime prevention of violations, before they have happened. Runtime prevention is complementary to the *ex post* approaches, in that it helps minimizing SLA violations, even if it cannot substitute offline analysis and optimization of service compositions entirely. We dubbed our approach PREVENT (prediction and prevention based on event monitoring).

The main scientific contribution of this paper is a novel end-to-end approach to automated SLA violation prevention. We use predictions of SLA violations, which are generated using regression from monitored runtime data, to trigger adaptations in the service composition. These adaptations should improve the performance for this instance in such a way that violations are prevented. We argue that our research improves on existing works which consider prediction and prevention only independently.

The remainder of this paper is structured as follows. In Section II we present a case study which we use in the rest of the paper. In Section III we discuss our main contribution. In Section IV we present some experimental results. Section V contains a discussion of related approaches, and Section VI finally concludes the paper.

II. Scenario

To illustrate the core ideas of this paper we use the case of a sports equipment reseller. This company conducts business by ordering cheaper equipment parts (e.g., table tennis blades, rubbers and sponges) in large amounts directly from manufacturers, assembling them in-house, and selling them to a limited number of business partners (such as shops). Business is based entirely on a Web service model, i.e., shops order products via a Web service interface, and the business process of the reseller is implemented using composition, integrating both internal (e.g., assembling service, warehouse service) and external services (e.g., shipping service, banking services). We have illustrated this case in Figure 1. Note that the figure is simplified for clarity.

978-0-7695-4128-0/10 $26.00 © 2010 IEEE
DOI 10.1109/ICWS.2010.21

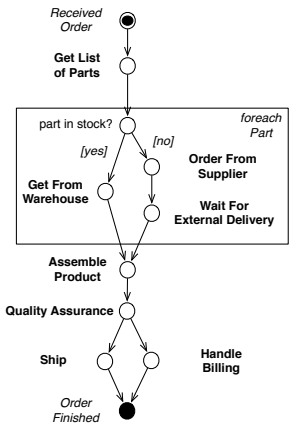

Figure 1. Illustrative Example

The interaction between the reseller and its customers is governed by individually negotiated SLAs. These SLAs contain a varying number of SLOs, with varying target values. For simplicity we focus on a simple SLA with only one SLO here: *Processing of a single order can take no more than 5 working days for custom-made products. For every additional day the client is entitled to a 3% discount, up to a maximum of 25%.*

III. Automated Prevention of SLA Violations

We have carried out our work on PREVENT as part of our ongoing VRESCO project [16]. VRESCO is a .NET based prototype which provides a registry for Web services metadata and QoS information (including QoS monitoring), as well as base services for dynamic service compositions. The VRESCO event engine (based on NEsper[1]), which emits status events whenever services are published or invoked, forms the technological basis of much work that we discuss here. See [16] for an in-depth description of the VRESCO system.

The overall architecture of PREVENT is sketched in Figure 2. We have identified three different phases of adaptation: (1) monitoring of runtime data, for which the *Composition Monitor* component is responsible, (2) prediction of violations, which is handled by the *SLO Predictor* component, and finally (3) the identification of possible preventative adaptation actions and application of these actions, carried out by the *Composition Adaptor* component. Composition Monitor and SLO Predictor are connected via the *Metrics Database*, which stores all collected runtime data. The Composition Monitor is configured using the *Metrics Definitions Database*. The SLO Predictor needs access to the *Checkpoint Database*, which specifies at which points in the execution of the composition the prediction should take place. The Composition Adaptor needs the information contained in the *Adaptation Actions Database* (a list of all available adaptation actions) and *SLA Database* to decide if process instances are likely to violate SLAs, and which actions can be triggered in these cases. The SLA, Adaptation Actions and Checkpoint Databases are considered to be inputs of our approach, i.e., SLA negotiation [8] or the process of identifying possible adaptation actions and checkpoints is out of scope. The same is true for modelling the impact models of adaptation actions (see below). We assume that a human domain expert is able to derive these models using e.g., Business Intelligence techniques. We will now present details to each of those components.

A. Event-Based Monitoring

The Composition Monitor is responsible for collecting all runtime information necessary for prediction of violations. Generally two different types of metrics are monitorable: *composition metrics* are defined based on one or a series of lifecycle events of the service composition, as produced by the VRESCO event engine (e.g., shipping activity has started, shipping activity has ended); *external metrics* contain data which stems from outside of the composition (e.g., the order date). The latter allow to seamlessly integrate any kind of external information which may influence SLA conformance.

We have depicted the PREVENT monitoring approach in Figure 3. The core of the Composition Monitor is the *Metric Processor*. On startup, the processor gets a parsed list of all metrics to monitor and their definitions from the *Definition Parser* (who has retrived this list from the Metrics Definitions Database). For every composition metric the processor needs to register exactly one event listener. It defines the composition lifecycle events forming the data basis for calculating this metric. For instance, for the metric "Payment Preference of Customer", a listener is generated which delivers the event issued after the activity "Get Payment Preferences" is finished. This event contains the necessary data in its body. The Metric Processor retrieves the raw metric value from a given event property path (e.g., "GetPaymentPrefsResult/Preference"), and (optionally) applies some postprocessing operation to this raw data. Postprocessing operations are given as scripts in the CSScript[2] language.

If the registered event listener delivers more than one event, aggregation functions (average, sum, min, max, count) may be used as part of the metric definition. External metrics are provided by an external data provider, e.g., a Web service or an external database. For these metrics the Composition Monitor does little more than retrieve the value from the external provider. Whenever the metric processor measures a metric at runtime (both composition and external metrics) it saves the value to

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

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

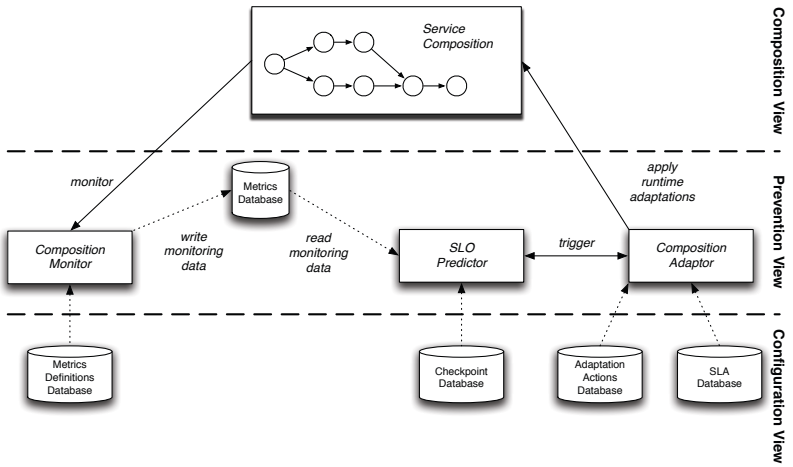

Figure 2. Approach Overview

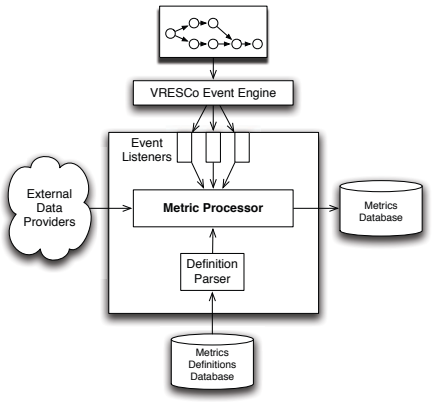

Figure 3. Composition Monitoring

the Metrics Database (identified by the ID of the current composition instance), where the value can be looked up by the SLO Predictor.

```
<fact name="average_exec_time_order_externally"
  type="integer"
  eql=
    "select _before.WorkflowId as id,
       (_after.WFTimestamp - _before.WFTimestamp)
       as data
     from BeforeWFInvokeEvent _before,
       AfterWFInvokeEvent _after
     where _before.WorkflowId = _after.WorkflowId
       and _before.ActivityName = 'order_externally'
       and _after.ActivityName = 'order_externally'
       and _after.BeforeReferer=_before.WFId"
  aggregation="avg"/>
```

Listing 1. Aggregated Metric Definition

Metric definitions are provided in an XML-based format. Listing 1 gives a sample metric definition, where the metric "average_exec_time_order_externally" is defined as the average time between the begin and end of the activity "order_externally". Since this activity is executed in a loop (cp. Figure 1) the event stream defined by the query (in "eql") may contain many events, which are aggregated using the aggregation function "avg". Event streams are defined using the Esper Query Language (EQL). The final result of the calculation is of type integer.

```
<fact name="total_nr_of_items"
  type="integer"
  eql="..."
  messagePath="GetPartListResult/Parts"
  script=
    "return (input as string).Split(';').Length;"
/>
```

Listing 2. Metric With Postprocessing

An example of a metric with postprocessing is given in Listing 2. This time we have omitted the event query for brevity. We assume that it returns a single event per composition instance. The list of items is accessible in this event by retrieving the event property "GetPartListResult" and, recursively, the property "Parts". There, the part list is stored as a simple string (a semicolon-separated list), so we apply a simple CSScript (basically plain C# code) to retrieve the number of items in the list.

B. Prediction of Violations

The metrics monitored by the Composition Monitor component are used by the SLO Predictor to identify problematic instances at runtime. We build upon our earlier results on prediction of SLA violations [15]. Our general approach is to predict SLO values at defined *checkpoints* in the composition execution via regression from measured and estimated runtime data.

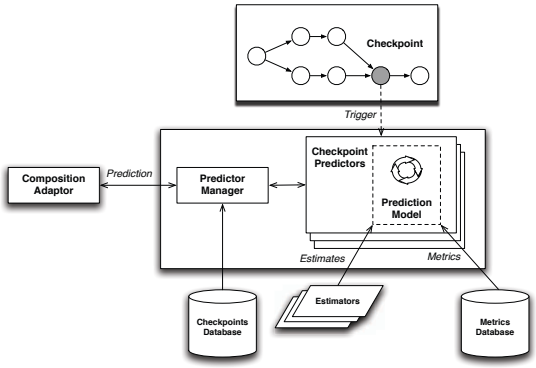

Figure 4. Prediction of Violations

In Figure 4 we have sketched our prediction approach. The *Prediction Manager* loads all checkpoint definitions from the Checkpoint Database, and instantiates one *Checkpoint Predictor* per definition. Definitions are again specified in an XML-based format. The predictor loads historical process data from the Metrics Database, and uses it to train a *Prediction Model*. The Prediction Model is usually (for numerical SLOs like the one sketched in Section II) a regression classifier, which is a black-box function taking a list of numeric and nominal values as input and producing a numeric value as output. The concrete implementaton of this classifier can be specified as part of the checkpoint definition. Internally, our system uses the WEKA machine learning toolkit[3], i.e., every regression classifier that WEKA supports can also be used for our approach. In our experiments we have generally used multilayer perceptrons, a variant of artificial neural networks [11]. At runtime, predictions are triggered if the checkpoint in the composition has been crossed. This is again implemented using lifecycle events emitted by the VRESCo event engine.

Most vital for the quality of the prediction is, of course, the available input data. Generally, there is a tradeoff to consider between timeliness and quality of predictions. Early predictions are less accurate, but allow for more adaptations to be carried out (since adaptations can only be applied to parts of the composition which have not yet been executed at the checkpoint). In this paper we do not discuss how to identify optimal checkpoints, however, some ideas on this topic can be found in [15], [23]. To allow for more timely predictions our approach leverages the concept of *Estimates*. They represent data which is not yet available in a checkpoint, but can in some way be estimated (e.g., the response time of a service which is to be invoked later in the composition). Estimates are produced by external components (*Estimators*), and are linked to measurable metrics. As part of our prototype system, we provide a limited number of default estimators, e.g., to produce estimates as the arithmetic mean of the last n measured values or based on QoS information available in VRESCo. However, oftentimes domain-specific estimators can be implemented which provide more accurate estimations, which in turn improve prediction quality (e.g., in the illustrative example the shipping time can be estimated rather accurately based on the address of the customer).

In order to be able to trigger adaptations based on predictions of SLO violations, it is essential to know the accuracy of the used prediction model. In [15], we defined two quality metrics: the *Mean Prediction Error* $\bar{e}$ is the artihmetic average of the differences between predicted (p_i) and monitored (m_i) values for a given number of instances (n) (Equation 1).

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

$\bar{e}$ tells us how far "off' our prediction in average is (lower $\bar{e}$ is, therefore, better). In addition, we use the *Prediction Error Standard Deviation* (denoted here simply as σ) to describe the variability of the prediction error (Equation 2).

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

High σ essentially means that the actual error for an instance can be much lower or higher than $\bar{e}$.

The Checkpoint Predictor triggers the Composition Adaptor if, according to the predicted SLO value, a violation is likely to occur. For this, the predictor compares the predicted SLO value with a given adaptation threshold (denoted as t_a). t_a is defined based on the actual SLO target value, often taking into account the mean prediction error and/or the prediction error standard deviation of the prediction model. For example, for the case described in Section II, the threshold for triggering adaptations may be the target SLO value minus 0.75 times the mean prediction error (Equation 3).

$$t_a = SLO - 0.75\bar{e} \tag{3}$$

The concrete definition of the threshold is, of course, depending on the preferences of the user and specific to every service composition.

C. Preventive Adaptation

Predictions of SLO violations finally trigger adaptations of the service composition. This is handled by the Composition Adaptor. This component has two main tasks. Firstly, it knows about all possible adaptation actions. Checkpoints are associated with exactly one SLO, and every action is assigned to one checkpoint, however, per SLO more than one adaptation action may be specified. The Composition Adaptor decides at runtime which subset of those actions are best suited to prevent a predicted

[3]http://www.cs.waikato.ac.nz/ml/weka/

violation. Secondly, after deciding which actions should be applied, the Composition Adaptor executes the identified adaptation actions.

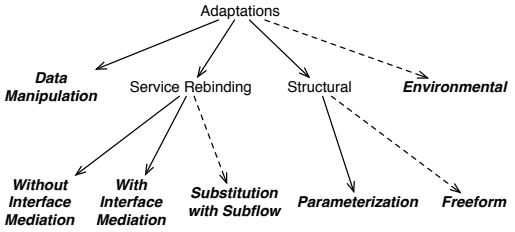

Figure 5. Taxonomy of Adaptation Actions

Just like monitoring data and prediction checkpoints, adaptation actions are specified using an XML dialect. In Figure 5 we have given a simple taxonomy of possible adaptation actions. Filled arrows represent actions that are currently supported by PREVENT, actions indicated by dashed arrows remain part of our future work.

Data Manipulations are the most simple type of adaptation action. In this case the service composition is in fact not changed. Instead the data flow of the composition is intercepted and adapted. In Listing 3 a simple data manipulation action is defined. In this example, the input message of the activity "ship" is intercepted, and the flag which indicates if express (e.g., overnight) shipping should be used is set to the constant value "true". More complex data manipulations can also copy or calculate values from other messages in the workflow.

```
<repairaction>
  <dataManipulationAction
    targetActivity="ship"
    message="input"
    messagePath="request/UseExpressShipping">
    <constant type="boolean">true</constant>
  </dataManipulationAction>
</repairaction>
```

Listing 3. Data Manipulation Action

More complex than data manipulation is *Service Rebinding*. We currently support all kinds of 1:1 rebinding, i.e., all cases where one service invocation is mapped to exactly one different invocation. We leverage the VRESCO mediation features to enable rebinding between services with different interfaces (*Service Rebinding With Interface Mediation*), however, cases where a single invocation needs to be mapped to a composition of invocations (*Substitution with Subflow*) are currently not supported. Definition of rebinding actions is relatively simple (the user only needs to specify which service instance, identified using a VRESCO ID, she wants to use for which activity). Finally, PREVENT allows to apply a limited number of changes to the structure of the service composition, namely, to remove or add activities at runtime. Conceptually, this is similar to the idea of parameterized workflows [13]. Therefore, we refer to these changes as *Parameterization*. We currently do not support *Freeform* adaptation of the composition structure, i.e., adaptations where any part of the workflow can be substituted by any other legal workflow fragment. Nor do we currently support *Environmental Adaptations*, i.e., adaptation of the execution environment, such as upgrading the hardware of the machine running the service composition.

The Composition Adaptor can use two different strategies to optimize the subset of adaptations which should be applied. Using the "safe" strategy the adaptor finds the subset which is most likely to prevent the violation (in the end, this means applying the adaptations with the biggest positive impact on the SLO value). Using the "minimal" strategy the adaptor finds a subset of adaptations which is still likely to prevent the violation, but where the actual SLO value is as close to the adaptation threshold t_a as possible. The advantage of the latter strategy is that in general only a minimal number of adaptation actions are applied, which may be desirable for the service provider if adaptations are costly. However, the disadvantage is that sometimes (if the impact prediction is too far off) violations that could have been prevented slip through.

```
<action name="USE_EXPRESS_SHIPPING">
  <repairaction>... </repairaction>
  <constraints>
    <conflictswith>USE_STD_SHIPPING</conflictswith>
  </constraints>
  <improvement>
    <improvedMetric>SHIPPING_TIME</improvedMetric>
    <improvementEstimate name="estimatedShippingTime"
      type="integer">
      <estimatorClass
        class="Estimator.ConstantEstimator"/>
        <argument value="1"/>
      <estimatedField name="SHIPPING_TIME"/>
    </improvementEstimate>
  </improvement>
</action>
```

Listing 4. Improvements and Constraints

For both strategies the adaptor needs to estimate the impact of a given set of adaptation actions in advance. This can be done by reusing the same principles and techniques that we have already used in the SLO Predictor. For this we associate all adaptation actions with one or more *improvement estimates* (see Listing 4). Improvement estimates are alike to the estimates used during prediction, in that they provide an estimation for a not-yet-known metric. They estimate the value that a given metric will likely have after the adaptation action has been applied. In the example the estimation is that when using express shipping the "shipping time" metric is always exactly "1 day". More complex estimations are of course also possible. Similar to the estimates used during prediction these are delivered by external estimator components, which are oftentimes domain-specific. Using these improvement estimates the

prediction is re-evaluated in the SLO Predictor, generating a prediction of the SLO value after adaptation. Just like the estimates used during prediction of violations, these improvement estimates are defined as part of the adaptation action. Note that we can use the same technique and model to predict both violations and impact of adaptations if we assume that every adaptation action has been applied a reasonable number of times before.

Adaptation actions are not necessarily independent. In general, any pair of actions can either be mutually exclusive (see the constraint defined in Listing 4), independent or co-dependent (either both or none of the actions need to be applied). The Composition Adaptor takes these dependencies into account when evaluating possible combinations of actions. Currently, the adaptor simply enumerates all possible combinations of adaptations to find the combination that is optimal according to the given strategy. Obviously, this is only feasible for a very small number of possible adaptation actions. More sophisticated and scalable optimization schemes are part of our future work and not considered here. Additionally, we currently assume that the order of adaptation actions is not important. We argue that this assumption is reasonable for non-conflicting rebinding and data manipulation actions. In the future, more complex adaptation actions will need us to take the order of actions into account.

IV. Experimentation

To evaluate the PREVENT approach we have implemented the case study described in Section II using Windows Workflow Foundation [21] technology. All experiments have been carried out on an Intel Xeon Dual CPU X5450 with 3.0 Ghz and 32 GByte RAM. On this machine we have hosted all services used in the composition, the composition engine, VRESCO and a number of test clients. The services emulate realistic QoS behavior (e.g., the response times of services follow a Monte Carlo simulation with service-specific parameterization).

We assume that clients expect the service to follow the SLO mentioned in Section II (however, for our experimentation we map the SLO target value of 5 days to 31500 time units). The average SLO value in the case study simulation was 26439 time units, with a variance of 4396. The number of violations per 100 instances (without adaptations) is generally between 15 and 30. We have defined a checkpoint before the activity "Assemble Product", and (for the second and third test) 5 possible adaptation actions – 3 data manipulation actions which are mutually exclusive (do shipping with priority 2 to 4), a service rebinding action (using a faster banking service for all payment-related activities), and a parameterization action (skip the quality assurance activity). Our general experimentation approach was as follows. Before every test we have cleaned the Metrics Database to guarantee independent tests. We then started 20 parallel clients and let them execute 15 orders each, resulting in an intial data set of 300 orders. This data formed our initial database of historical process instances (training phase). Afterwards, we commenced the actual experiments by running 5 additional orders per client, where we measured the predicted SLO value, the actual measured SLO value, and the adaptations that have been carried out. In all following plots the x-axis depicts the 100 instances monitored after the training phase. The y-axis depicts the SLO values. We also show the target SLO value as pink (dashed) line at 31500.

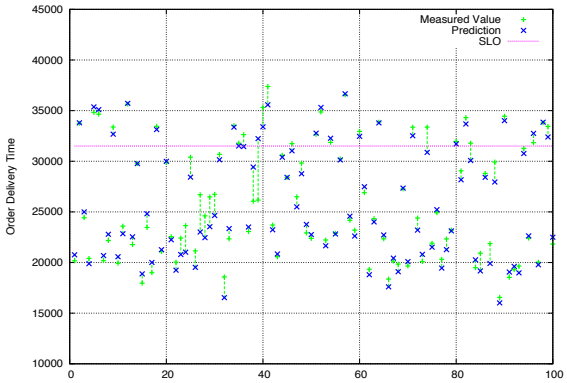

Figure 6. Predicted and Measured Values

Figure 6 shows the first test run, where we disabled adaptation altogether (i.e., we removed all 5 adaptation actions mentioned above). As we can see predictions (blue x) and measured values (green $+$) are generally close, so the prediction quality is good ($\bar{e}$ is around 700 in this experiment). However, there are some instances where prediction is off by a larger margin, e.g., the instances 38 and 39. In this test run we predicted 24 violations, and actually measured 27 (3 violations have not been forseen by PREVENT).

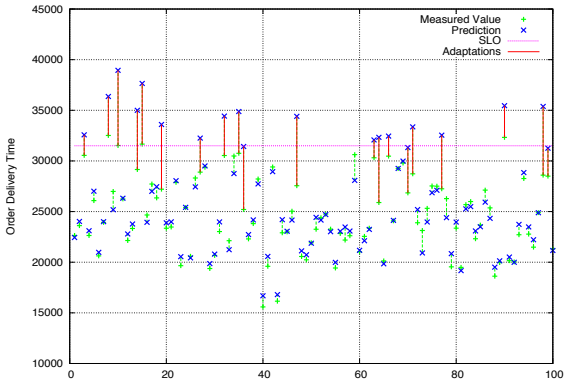

Figure 7. Safe Adaptation Strategy

In Figure 7 we show the impact of enabling adaptation using the "safe" strategy. In the instances depicted with a

filled red line adaptations have been triggered. In general, the length of this line correpsonds to the amount of adaptation applied (plus or minus the prediction eror), i.e., if the line is longer more actions, or actions with bigger impact, have been applied. Since we used the "safe" strategy the lines are generally of similar length (this strategy always applies the same adapatation actions, the ones that have the biggest positive impact on the performance). We can see that many predicted violations have successfully been prevented using adaptation (e.g., 3, 10, 14). However, in some cases even with adaptation the violation could not be prevented (e.g., 8). Note that in 2, 71 and 99 adaptations have been triggered even though the actual prediction was slightly below the target value. This is because the trigger threshold t_a is slightly below the actual target value (cp. Equation 3). In this experiment we have predicted 18 violations, and only measured 4 violations after adaptation, i.e., about 78% of all predicted violations have been successfully prevented.

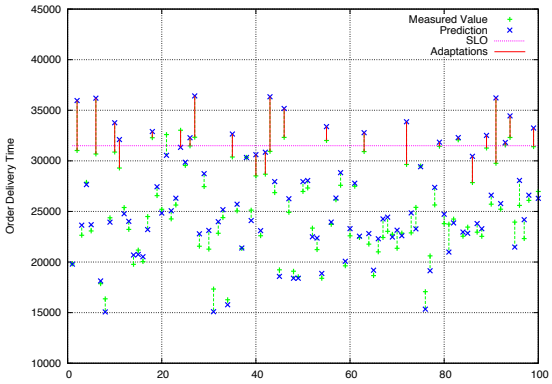

Figure 8. Minimal Adaptation Strategy

In Figure 8 we changed the adaptation strategy to "minimal". We can see that overall the impact of adaptation is reduced (the filled red lines are shorter in average), since PREVENT tried to apply only as much adaptation as needed to prevent a violation. However, this has the effect that in some cases (e.g., 18 or 44) violations are not successfully prevented because "too little" adaptation is done. In this experiment we predicted 20 violations, and actually had 8 violations after adaptation. This means that only 60% of all violations could be prevented. Note that the "minimal" strategy might still be superior for service providers if adaptation is expensive, i.e., if they have a strong interest in applying only adaptation actions that are strictly necessary.

V. Related Work

The work discussed in this paper belongs to the wider area of SLA management. Many languages (both academic and industrial) have been published for formulating SLAs as well as their monitoring and enforcement (e.g., WSLA [7] or SLAng [22]). Castellanos et al. have introduced the differentiation of approaches which *explain* and approaches which *predict* violations [5]. Most current approaches in SLA management clearly fall into the former class, such as recent work by Bodenstaff et al. [3], [4] or Wetzstein et al. [23]. Contrary, PREVENT belongs more to the latter class, for which less research is available. To the best of our knowledge the only papers which deal with runtime prediction of SLA violations in Web service compositions at the time of this writing are our own earlier work [15], as well as work by Sahai et al. [20] and Zeng et al. [24]. All of these approaches use some means of learning from recorded historical process data to estimate the performance of ongoing instances. However, none of them provide means to automatically prevent violations after they have been detected, which is the main contribution of this paper. Similar work on the prediction of performance problems (in a different domain) has been published by Duan and Babu [9]. Their *Fa* tool uses Bayesian networks to proactively identify performance problems in database systems. All of these approaches demand for a comprehensive set of existing process data to learn from. This kind of data can be collected using runtime monitoring of compositions, as presented e.g., by Baresi et al. [2] or Moser et al. [17]. The former work presents an integration of two separate research approaches, which result in a powerful monitoring facility for WS-BPEL based compositions. The main focus seems to be on the definition, monitoring and correlation of process instance data. By contrast, the latter work limits monitoring to a defined number of QoS attributes, such as response time or availability.

There is also a significant body of work on the adaptation of service compositions. Among these, we can distringuish two main groups: (1) approaches which adapt compositions exclusively by exchanging service bindings (cp. the service rebinding actions in Figure 5), and (2) approaches which are able to change the composition structure (structural actions in Figure 5). In a very primitive form service rebinding adaptations are already possible using WS-BPEL directly (by using the *Dynamic Partner Link* construct), however, in practice problems such as Web service selection at runtime, QoS-based optimization or interface differences ask for more advanced service rebinding facilities. These problems are covered by frameworks such as PAWS [1] or WS-Binder [19]. One straight-forward approach to structural adaptation has been presented by Karastoyanova et al. [13]. In this paper the authors discuss the idea of parameterized WS-BPEL processes, where activities can be enabled or disabled dynamically. This approach has its merits because of its clean design, however, it clearly cannot provide "real" structural dynamicity, because all possible adaptations already have to be contained in the original process. More complex adaptations are made possible by applying the

aspect-oriented programming (AOP) paradigm to service compositions [6], [12], [14]. The adaptation approach implemented in PREVENT generally also belongs to this category, even if adaptations are currently more limited than AOP-based approaches. However, we plan to extend our work with more complex adaptation actions as part of our future research.

VI. Conclusion and Future Work

Preventing violations of SLAs is a main concern for providers of service compositions aiming at satisfying their customers. While most current research in the area considers the explanation of violations after they have happened we propose the PREVENT system, a framework for runtime prediction and subsequent prevention of violations. PREVENT is based on the idea of monitoring and analyzing runtime data to trigger adaptation actions in endangered composition instances. Our system is based on the VRESCO runtime environment, which provides facilities used for monitoring and adaptation. We have shown how PREVENT can be successfully used to significantly reduce the number of violations in an illustrative case study.

While the current incarnation of PREVENT is promising, there are still some open issues. Most importantly our current approach does not scale to a larger number of adaptation actions per checkpoint, since optimization is based on full enumeration. Secondly, the adaptation actions supported so far are somewhat limited in that complex structural adaptations of the composition are not supported. Finally, we currently do not take the costs of adaptations into account (e.g., there are costs associated with using express shipping instead of regular shipping, which may in some cases be higher than the gain of not violation the SLA). We plan to work on all of these issues as part of our future work in this project.

Acknowledgement

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

References

[1] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani, "PAWS: A Framework for Executing Adaptive Web-Service Processes," *IEEE Software*, vol. 24, no. 6, pp. 39–46, 2007.

[2] L. Baresi, S. Guinea, R. Kazhamiakin, and M. Pistore, "An Integrated Approach for the Run-Time Monitoring of BPEL Orchestrations," in *Proceedings of the 1st European Conference Towards a Service-Based Internet (Servicewave08)*, 2008.

[3] L. Bodenstaff, A. Wombacher, M. Reichert, and M. Jaeger, "Monitoring Dependencies for SLAs: The MoDe4SLA Approach," in *Proceedings of the 2008 IEEE International Conference on Services Computing (SCC'08)*, 2008.

[4] L. Bodenstaff, A. Wombacher, M. Reichert, and M. C. Jaeger, "Analyzing Impact Factors on Composite Services," in *Proceedings of the 2009 IEEE International Conference on Services Computing (SCC '09)*, 2009.

[5] M. Castellanos, F. Casati, U. Dayal, and M. Shan, "Intelligent Management of SLAs for Composite Web Services," in *Databases in Networked Information Systems*, 2003.

[6] A. Charfi and M. Mezini, "Ao4bpel: An aspect-oriented extension to bpel," *World Wide Web*, vol. 10, no. 3, pp. 309–344, 2007.

[7] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef, "Web Services on Demand: WSLA-Driven Automated Management," *IBM Systems Journal*, vol. 43, no. 1, pp. 136–158, 2004.

[8] E. Di Nitto, M. Penta, A. Gambi, G. Ripa, and M. L. Villani, "Negotiation of Service Level Agreements: An Architecture and a Search-Based Approach," in *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07)*, 2007.

[9] S. Duan and S. Babu, "Proactive Identification of Performance Problems," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, 2006.

[10] S. Dustdar and W. Schreiner, "A Survey on Web Services Composition," *International Journal of Web and Grid Services*, vol. 1, no. 1, pp. 1–30, 2005.

[11] S. Haykin, *Neural Networks and Learning Machines: A Comprehensive Foundation*, 3rd ed. Prentice Hall, 2008.

[12] D. Karastoyanova and F. Leymann, "BPEL'n'Aspects: Adapting Service Orchestration Logic," in *Proceedings of 7th International Conference on Web Services (ICWS 2009)*, 2009.

[13] D. Karastoyanova, F. Leymann, J. Nitzsche, B. Wetzstein, and D. Wutke, "Parameterized BPEL Processes: Concepts and Implementation," in *Proceedings of the International Conference Business Process Management (BPM'06)*, 2006.

[14] W. Kongdenfha, H. R. Motahari-Nezhad, B. Benatallah, F. Casati, and R. Saint-Paul, "Mismatch Patterns and Adaptation Aspects: A Foundation for Rapid Development of Web Service Adapters," *IEEE Transactions on Services Computing*, vol. 2, no. 2, 2009.

[15] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann, "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)*, 2009.

[16] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "End-to-End Support for QoS-Aware Service Selection, Binding and Mediation in VRESCo," *IEEE Transactions on Services Computing (TSC)*, 2010 (forthcoming).

[17] O. Moser, F. Rosenberg, and S. Dustdar, "Non-Intrusive Monitoring and Service Adaptation for WS-BPEL," in *Proceedings of the 17th International Conference on World Wide Web (WWW'08)*, 2008.

[18] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer*, vol. 40, no. 11, 2007.

[19] M. D. Penta, R. Esposito, M. L. Villani, R. Codato, M. Colombo, and E. D. Nitto, "WS Binder: a Framework to Enable Dynamic Binding of Composite Web Services," in *Proceedings of the International Workshop on Service-Oriented Software Engineering (SOSE'06)*, 2006.

[20] A. Sahai, V. Machiraju, M. Sayal, A. P. A. v. Moorsel, and F. Casati, "Automated SLA Monitoring for Web Services," in *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM)*, 2002.

[21] D. Shukla and B. Schmidt, *Essential Windows Workflow Foundation*. Microsoft .Net Development Series, 2006.

[22] J. Skene, D. D. Lamanna, and W. Emmerich, "Precise service level agreements," in *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, 2004.

[23] B. Wetzstein, P. Leitner, F. Rosenberg, I. Brandic, F. Leymann, and S. Dustdar, "Monitoring and Analyzing Influential Factors of Business Process Performance," in *Proceedings of the 13th IEEE EDOC Conference (EDOC'09)*, 2009.

[24] L. Zeng, C. Lingenfelder, H. Lei, and H. Chang, "Event-Driven Quality of Service Prediction," in *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC'08)*, 2008.