FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Using Time Series Analysis for Predicting Service Level Agreement Violations in Service Compositions

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Johannes Ferner
Matrikelnummer 0625379

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.Dr. Schahram Dustdar
Mitwirkung: Univ.Ass. Mag.Dr. Philipp Leitner

Wien, 08.03.2012

(Unterschrift VerfasserIn) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Using Time Series Analysis for Predicting Service Level Agreement Violations in Service Compositions

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Johannes Ferner
Author 0625379

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Mag.Dr. Schahram Dustdar
Assistance: Univ.Ass. Mag.Dr. Philipp Leitner

Vienna, 08.03.2012

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Johannes Ferner
Kuefsteingasse 42, 1140 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift VerfasserIn)

Acknowledgements

I would like to thank my parents, Mag. Roswitha and Dr. Helmuth Ferner, for enabling my studies and encouraging and assisting me in all concerns. The Faculty of Informatics, the Distributed Sytems Group and o.Univ.-Prof. Dr. Schahram Dustar for giving me the permission to write this thesis. My gratitude goes to Univ.Ass Dr. Mag. Philipp Leitner for his generous support and many enlightening discussions.
Especially, I would like to thank Muna Awad-Masalmeh for supporting me in difficult times and being lovely patient when the evenings program consisted of programming.

Abstract

Service Level Agreements (SLAs) play a fundamental role in service compositions. As they represent binding contracts between service consumers and service providers, their violation is usually linked with profit loss. In order to maximize profit a multitude of SLA violation detection mechanisms exist that can roughly be classified into two categories: the ex-post and ex-ante detection, whereas the latter is fundamental when preventing SLA violations and subsequently is of great interest.
Most of the current approaches either focus on the ex-post detection or insufficiently consider the behavior of SLAs over time. Moreover, the accurate prediction of target values, being of categorical nature, is still open.
The following thesis elaborates a new ex-ante SLA violation detection mechanism, which makes use of the general statistical class of Autoregressive Integrated Moving Average (ARIMA) models. Thereby the past realization values of specific target values - called Service Level Objectives (SLOs) - are analyzed and fitted into a time series model. As the application of SLAs involves the validation of as many services as possible at the same time, the reduction of human intervention is desirable. Therefore, the creation and training of the statistical models is done automatically.
All theoretical models are implemented and tested within the research prototype Vienna Runtime for Service Compositions (VRESCo) and the results are validated in the course of a practically relevant example use case, proving the great potential of ARIMA models in the domain of forecasting SLA violations: Differently behaving SLAs (e.g. the delivery time of a reseller) result in a prediction accuracy ranging from 99.31% up to 99.77%. In the case of categorical SLAs the resulting prediction accuracy of 7.18% indicates the need of further research work.

Kurzfassung

Service Level Agreements (SLAs) spielen eine fundamentale Rolle in Service Kompositionen. Sie stellen verbindliche Verträge zwischen Service Verbrauchern und Service Dienstleistern dar und schreiben somit ein zu leistendes Qualitäts-Mindestmaß eines Services vor. Da die Verletzung von SLAs meist mit Kosten verbunden ist, existiert eine Vielzahl an Methoden zu deren Erkennung, welche grob in zwei Kategorien eingeteilt werden können: die ex-ante und ex-post Methode. Letztere beschreibt die Vorhersage von Verstößen bevor diese überhaupt auftreten und ist daher von großer Bedeutung.
Aktuell verbreitete Ansätze fokussieren meist auf die ex-post Erkennung oder messen dem Zeitreihenverhalten von SLAs zu wenig Bedeutung zu. Darüber hinaus ist die Vorhersage von Zielwerten kategorischer Art noch weitgehend unbehandelt.
Die folgende These erarbeitet eine neue ex-ante Methode zur Erkennung von SLA Verletzungen, welche auf der allgemeinen statistischen Klasse von Autoregressive Integrated Moving Average (ARIMA) Modellen basiert. Dabei werden die vergangenen Realisierungen von spezifischen Zielwerten - genannt Service Level Objectives (SLOs) - analysiert und in ein Zeitreihenmodell eingepasst.
Da der Anwendungsbereich von SLAs die Validierung von möglicherweise vielen Services zur gleichen Zeit beinhaltet, ist es wünschenswert menschliche Interventionen weitestgehend zu vermeiden. Daher erfolgt die Erzeugung und Adjustierung der statistischen Modelle automatisch. Alle theoretischen Modelle sind im Rahmen des Forschungs-Prototyp Vienna Runtime for Service Compositions (VRESCo) umgesetzt und die Ergebnisse anhand eines praxisnahen Anwendungsfalls validiert. Diese zeigen das große Potenzial der ARIMA Modelle im Bereich der Prognose von SLA Verletzungen: SLAs mit unterschiedlichem Verhalten über die Zeit (z.B. die Lieferzeit von einem Reseller) erreichen eine Vorhersagegenauigkeit von $99,31\%$ bis $99,77\%$. Die Testergebnisse für kategorische SLAs ergeben eine Vorhersagegenauigkeit von $7,18\%$ und weisen somit auf die Notwendigkeit weiterer Forschungen hin.

Contents

1 Introduction **1**
1.1 Motivation . . . 2
1.2 Problem Statement . . . 2
1.3 Technical Contribution . . . 3
1.4 Thesis Organization . . . 3

2 State of the Art **5**
2.1 Service Oriented Architecture . . . 5
Windows Communication Foundation . . . 6
Service Level Agreements . . . 9
2.2 Time Series . . . 10
Regression . . . 11
Differencing Time Series . . . 11
Autoregressive Integrated Moving Average . . . 11
Building Autoregressive Integrated Moving Average Models . . . 13

3 Related Work **15**
3.1 SLA Violation Detection . . . 16
3.2 Time Series Analysis . . . 20

4 Use Case **25**
Challenges . . . 26

5 Contribution **29**
5.1 Overview . . . 29
5.2 Model . . . 33
5.3 Implementation . . . 37

6 Evaluation **43**
6.1 Performance . . . 44
6.2 Simple Cases . . . 46
6.3 Time Series with Trend . . . 51
6.4 Saisonality . . . 52
6.5 Categorical Cases . . . 53

6.6 Results Review 55

7 Conclusion **57**

7.1 Summary 57

7.2 Future Work 57

A Diagrams **59**

A.1 Complete Workflow Diagram of the Use Case 59

B Listings **65**

B.1 Listings of Implemented R Code 65

Bibliography **73**

CHAPTER 1

Introduction

The way of designing software has changed significantly over the past decades. Requirements for today's applications focus more and more on aspects of distributed environments, leading to certain key success factors such as adaptability, reusability, scalability, openness etc.
As a result, development processes are getting more complex and the need for concepts, which support software developers in managing all involved difficulties, arises. That's where Service Oriented Architecture (SOA) [9, 28, 31] comes into play. SOA is a paradigm that aims to support the development of platform-independent and reusable software, by structuring pieces of coherent functional modules into autonomous services that can be accessed by any user - independently from platforms or transmission protocols. Moreover, customers that want to use certain services need to search and find them. This is done by so-called service registries whose purpose is the provision of search mechanisms and standardized description of the offered services. Such a description normally not only includes the pure functionality but also information about *how* this functionality is offered. If, for example, a customer needs to solve a time critical task, performance attributes of the service may be as important as the functionality itself. In order to be able to classify provisioned services by means of their non-functional aspects one or more SLAs [2, 16, 8] can be established between service consumers and providers. Such agreements not only describe quality aspects but also measurements that have to be taken, if the agreed issues were not fulfilled. As it is not desirable to violate beforehand specified SLAs ongoing research work is exploring the automatic detection and statistical analysis of such events to minimize them and their impact. A multitude of methods exist [48, 59, 35, 14] that range from the uncomplicated descriptive case, where the violation of an SLA is detected and simply reported to a specific entity, to sophisticated mechanisms that use statistical analysis to predict the occurrence of undesirably events and automatically seek for adequate replacements [30, 27, 37, 11, 55, 57]. Especially the second case is getting more and more important if one considers the enforcement of whole business process through so-called Service Compositions.
Service Compositions cascade a multitude of services - that are discovered and called through service registries - so that specific work flows can be executed automatically. It is obvious that the automatic prediction of undesirable service conditions (e.g. service not available or service

has an unusual high response time) imposes a great potential to enhance the performance of work flows. In the right environment service compositions can be changed dynamically during execution. Together with precise statistical tools, low-performing parts can be replaced by others leading to a better overall performance of work flow executions.
This thesis aims to explore the prediction of SLA violations before they actually occur. It is achieved through the statistical analysis of data that is collected during execution phases. Especially (but not only) historical data of concerned services acts as information base for the deduction of future behaviors.

1.1 Motivation

SLAs play an important role in service compositions. They represent contracts between two or more parties that constitute certain quality requirements upon service usage. As quality is hardly measurable, SLAs consist of SLOs: functions that map numerically measurable metrics onto a target value. When the target value is met, the service is considered to provide the expected quality. In the other case, the quality is not met and a penalty, dependent on the divergence, is triggered.
As service compositions are an integral part of nowadays business processes, the settlement and enforcement of viable SLAs is vital. It lies in the very nature of SLAs that their violation is almost certainly linked with profit loss - directly or indirectly. If, for example, a reseller is not able to deliver his products in time - according to beforehand specified SLAs -, a payoff between customer dissatisfaction and a discount (profit loss) has to be considered.
As it is desirable to avoid such violations, the need for automated prediction techniques arises. Thereby, the creation of statistical models is helpful, as they provide a possibility to predict future service behaviors in respect to their quality attributes. Once an undesirably event is detected, the impact can then be minimized by replacing the problematic service with another one. The creation of such models should be done automatically since it speeds up work flow executions and lowers costs.
Current approaches are available, but do not consider the behavior of SLO values over time [48, 59, 35, 14]. In reality, many applications are dependent on their own former states and exhibit some kind of auto regressive behavior [5, 50, 60]. This thesis aims to predict SLA violations that occur in service compositions, by analyzing historical SLO values and creating a mathematical model that describes dependencies on past values as well as on external factors. Therefore, an application is implemented within the research prototype VRESCo [54] which is used to evaluate the work within a sample use case scenario.

1.2 Problem Statement

As mentioned before, techniques to predict SLA violations as well as a multitude of statistical models, to describe the mathematical behavior of time series, already exist. However, the application of SLAs is very wide and allows the definition of many quality aspects of different kinds. This variety makes it hard to create measuring and predicting techniques to cover all possible forms of SLAs. Most of the current SLA violation detection techniques focus on the cause rather than trying to avoid them. Few approaches for the ex-ante detection exist but are instance based. This means, that the history of the SLO process is not considered. The involvement of historical data makes it possible to detect SLA violations in earlier points of service composition executions and therefore helps to minimize the impact thereof.

1.3 Technical Contribution

The main contribution of this thesis deals with the automated forecast of target values of specific SLAs in service compositions. Therefore the *C#* project *TSForecasting* is implemented within the research prototype VRESCo. Its purpose is the creation of time series models to predict the future outcome of SLO values. Additionally, it makes use of so-called estimates and facts, which are treated as external factors that can influence the behavior of SLOs over time. This work addresses the issues that arise from the following research questions:

- How to fit and estimate a model, which mathematically describes a set of values as a time series that depends on its own past realization values, without human intervention?
- Is it possible to avoid SLA violations before they actually occur by describing a set of SLO values as mathematical regression model with exogenous inputs? What is the prediction accuracy of this method?
- How to create an Autoregressive (AR) model for categorical SLO values?

1.4 Thesis Organization

Chapter 2 provides an overview of time series in general, a more specific view on the class of ARIMA models and methods to parameterize and fit them. Moreover, a basic understanding of SOA, Windows Communication Foundation (WCF) [7] and SLAs is established. Recent research work, that is closely related to the automatic detection of SLA violations, is presented in Chapter 3. After a short description of a use case in chapter 4, which contains a sample scenario for testing and evaluating purposes, a detailed overview of the contributed work is given in chapter 5. While the evaluation is detailed in Chapter 6, the last chapter closes this thesis by concluding the results and providing an outlook of possible future work.

CHAPTER 2

State of the Art

2.1 Service Oriented Architecture

In general, Service Oriented Architecture (SOA) describes an abstract concept of software architecture, that deals with the growing complexity of nowadays software development. Although its concepts are known for a long time, it gained popularity with the rise of web services. The basic idea of SOA involves the separation of complex systems into several independent services, that interact in a standardized way.
Since SOA describes an abstract concept, it is hard to provide an exact definition. Among various definitions [9, 31, 28, 15, 63, 33, 32, 53, 42, 52, 39], this Thesis uses a basic specification according to [31, 28]. Hence, at least the following requirements must be satisfied to support a SOA:
First of all, it should be possible to describe and publish a service. As SOA is strongly related to web services, it is common to use its standards and technology. Specifically, the Web Service Description Language (WSDL) [65] - which is based on the XML standard [19] - is a good choice, as it has an open and well known specification. Furthermore a public registry such as UDDI [38] is used to publish the service along with its standardized description. Such registries generally fulfill the second requirement. That is the discovery of specific services. A client is able to search for services, according to their descriptions and keywords. Finally, a service has to be accessible. A common implementation of this requirement is the SOAP [66] protocol in combination with the HTTP standard [18].
Since this Thesis only provides a basic overview of the SOA definition, the interested reader can find a comprehensive overview in [9, 31, 28]. Figure 2.1 depicts a graphical overview of the basic SOA concept.

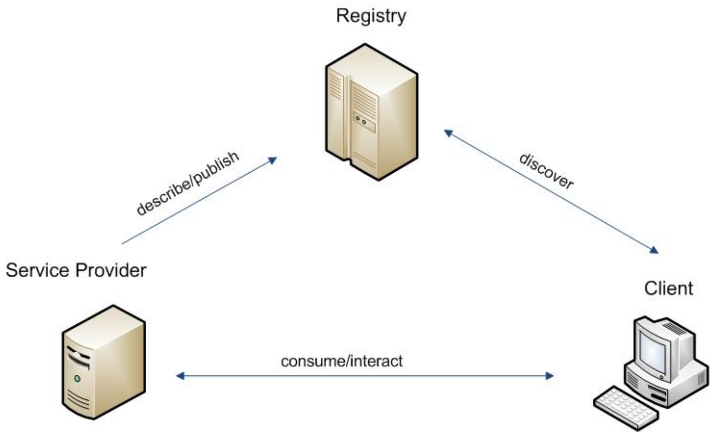

Figure 2.1: Graphical Overview of SOA

Windows Communication Foundation

Windows Communication Foundation (WCF) was introduced in 2006 as part of the Microsoft .NET Framework 3.0. Its main goal is the creation of a platform, that unifies existing communication technologies, as well as ensuring the interoperability with other technologies by providing a service oriented approach. In order to ensure this the WCF uses SOAP [66] in the messaging layer, as well as the WSDL [65] specification. As the WCF is a very extensive feature, only a basic overview is provided. For specific details one can have a look at [7, 36, 62]

Windows Communication Foundation Services

A WCF Service consists of a service class, a hosting process and one or more endpoints. The definition of a service class can be made in any Common Language Runtime (CLR)-based [46] language. It typically involves a public interface, that is known to the caller, a specific implementation and a set of attributes, that define data types and service behaviors. The standard way of implementing a WCF service is to create an interface, that is annotated with the *ServiceContract* attribute. Every operation that should be accessed remotely is marked as *OperationContract*. A sample Interface is shown in Listing 2.1

```
/*
 * The attribute [ServiceContract] defines
 * this interface as public service
 */
[ServiceContract]
public interface IService
{
  /*
   * [OperationContract] enables the public access of this method
   */
  [OperationContract]
  Boolean dataValid(ServiceData data);
}
```

Listing 2.1: A Sample Interface for a WCF Service

If the service only deals with primitive data types, it is enough to implement this interface and link it to a service host - such as the Internet Information Services (IIS) or the Windows Activation Service (WAS) [44]. Listing 2.2 shows a sample implementation of interface 2.1.

```
//this implementation is invoked
public class TheService : IService
{
  public Boolean dataValid(ServiceData data)
  {
    return data.isValid();
  }
}
```

Listing 2.2: Interace implementation of 2.1

If, however, the service should be hosted in an arbitrary process, WCF provides the *System.ServiceModel.ServiceHost* class, that takes as argument the class type, creates a new hosting process and publishes the service.
The specification of complex data types is done by using the attributes *DataContract* - which defines an arbitrary class or other type as a usable data structure - and *DataMember* marking any primitive type as part of the contract. A complex type has to be serializable, and is therefore defined in the *System.Runtime.Serialization* name space. See Listing 2.3 for a sample data type declaration.

```
/*
 * class has to be serializable if
 * datatype is used within a service
 */
[Serializable]
public class ServiceData
{
  /*
   * every attribute marked as [DataMember]
   * can be used in a service invokation
   */
  [DataMember]
  boolean isValid;
  /* more data types */

  /*
   * some method
   */
  public Boolean isValid()
  {
    return this.isValid();
  }
}
```

Listing 2.3: A Sample Data Type declaration for WCF Service Usage

As described before, a WCF service is made public through one or more endpoints. The most convenient way to define an endpoint is to provide an XML configuration file, which contains the description of the endpoint address, a specific binding and the class type of the service contract. Details can be found in [7]
Listing 2.4 gives an example for publishing a WCF Service within the *System.ServiceModel.ServiceHost* class.

```
//the wcf namespace
using System.ServiceModel;
using System.Runtime.Serialization

namespace Services
{
  //class contains code to publish a WCF Service via service host
  public class WCFService
  {
    static void main(String[] args)
    {
      //create the host
      ServiceHost host = new ServiceHost(typeof(IService));
      host.Open();
      //wait for invokations...
      host.Close();
    }
  }
}
```

Listing 2.4: Publishing a WCF Service in an arbitrary Process

The following listing depicts a simple example for a WCF service definition.

Windows Communication Foundation Clients

The creation of a WCF client is relatively simple. All the client has to know, is the definition of the service contract and all included data contracts. With the given information a proxy can be created and linked with an according endpoint address. The client can then invoke all public service operations through the proxy. [7]
A simple example for this is illustrated in the following listing:

```
using System.ServiceModel;
//the namespace of the public interface and the data contracts
using Services;

namespace Client
{
  //a simple example to invoke a service through a proxy
  class WCFClient
  {
    static void Main(String[] args)
```

```
      {
        //specify a binding
        WSHttpBinding binding = new WSHttpBinding();
        //configuration of the binding ...

        //specify binding, and enpointaddress
        ChannelFactory<IService> cf = new ChannelFactor<IService>(binding, new
            EndpointAddress("http://myservice.host.at/service"));
        //retreive the proxy
        IService service = cf.CreateChannel();

        //will return true
        service.dataValid(New ServiceData(true));
      }
    }
}
```

Listing 2.5: A simple example for a WCF service client

Following illustration presents a graphical overview of a WCF service invocation through a proxy

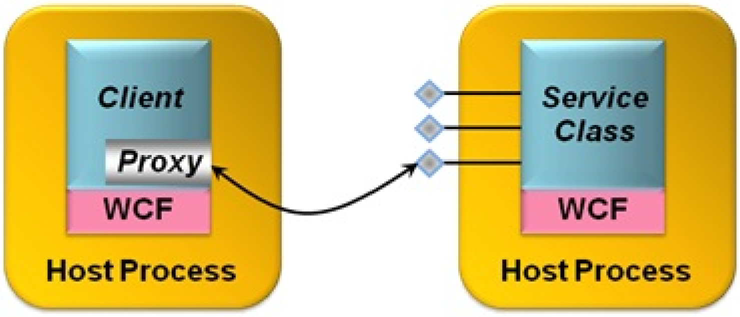

Figure 2.2: Graphical overview of a WCF service invocation (taken from [7])

Service Level Agreements

Service Level Agreements (SLAs) are an important concept in the context of SOA. Although a significant effort was put into the development of standards, that describe the functional behavior of services, the question of quality aspects is still open. Basically, an SLA is a contract between a service provider and a service consumer, that defines certain non-functional requirements. These requirements are typically expressed through so-called Quality of Service (QoS) parameters, which describe the mutual understandings and expectations of a service between several stakeholders [2].
Formally, an SLA consists of Service Level Objectives (SLOs), that define the *guaranteed condition of a service in a given process* [16]. An SLO is therefore a function that yields a measurable output, indicating whether a certain quality requirement of a service is satisfied or not. If, for

example, an SLA states that the availability of a service has to be at least 99%, the according SLO is characterised by the following equation:

$$Availability = \frac{ServiceInvocations_{Successful}}{ServiceInvocations_{Total}} \tag{2.1}$$

Moreover an SLA describes actions, that must be taken in case of violations and SLA parameters (called metrics), depicting the methods of measurements. A metric can either define how to measure a service item (resource metric), or how to aggregate other metrics (composite metric). In the above example the metric $ServiceInvokations_{Total}$ depicts a composite metric that is measured by summing up the resource metric $ServiceInvocation$ over a month.

An SLA is typically specified within an XML schema [17] definition, that constitutes basic rules of the SLA definition [8, 2, 34, 61, 16, 21, 4, 64, 69, 41]. When definitions of concrete SLAs are available, they are measured and - in case of a violation - previously constituted actions have to be taken. Such actions could be the notification of the SLA signers. However in most cases these actions are defined by penalties that the service provider has to pay. In the case of Availability an appropriate penalty would oblige the service provider to offer a 5% discount if the availability, falls below 95% and a 10% discount if it is lower than 90%.

2.2 Time Series

The following description of Time Series is mostly based on [5, 60, 50, 12, 49, 47]. The aim of time series analysis is to construct a mathematical model for a given set of time-ordered data, in order to obtain assumptions about its probabilistic behavior.

A time series or stochastic process is a collection of random variables $\{x_1, x_2, x_3, \dots\}$,where $\{xt\}$ denotes a random variable at a specific time point t. In general, the index t consists of integer values $\{^+_-1, ^+_-2, \dots\}$. A good description of a stochastic process x_t can be obtained by the probability distribution $p(xt)$ and its first 2 moments. Namely, the mean

$$\mu = E(x_t) = \int_{-\infty}^{\infty} xp(x)\mathrm{d}x \tag{2.2}$$

and the auto covariance function

$$\gamma(s,t) = cov(x_s, x_t) = E[(x_s - \mu_s) * (x_t - \mu_t)] \tag{2.3}$$

which can be normalized to the interval [-1, +1] with the Autocorrelation Function (ACF) function given by

$$\rho(s,t) = \frac{\gamma(s,t)}{\sqrt{\gamma(s,s) * \gamma(t,t)}} \tag{2.4}$$

Since stochastic processes often require weak stationarity a short definition is provided here. Details can be found in [60]. A stochastic process is said to be stationary if

1. the mean μ_t is a constant and does not depend on the time t
2. the auto covariance function $\gamma(s, t)$ only depends on the time lags $|t - s|$ and not on the specific time point itself

Regression

The classical regression model consists of a univariate or multivariate dependent variable $\{X_t\}$, which produces an output over time, that needs to be predicted, a collection of independent variables $\{z_{t1}, \ldots, z_{tm}\}$ - also called covariates - and a number of fixed coefficients, that need to be estimated. The classical regression model has the following form

$$x_t = \beta_1 z_{t1} + \cdots + \beta_m z_{tm} + \epsilon_t \tag{2.5}$$

where x_t denotes the dependent variable $z_{t1}, \ldots, z_{tm}$ the matrix of independent variables, $\beta_1, \ldots, \beta_m$ the parameters, that need to be estimated and ϵ_t the classical error term with a zero mean $E(\epsilon_t) = 0$ and a constant variance $V(\epsilon_t) = \sigma^2$

The unknown parameter vector β can be estimated with common methods like the Ordinary Least Squares (OLS) estimator, or the Maximum Likelihood (ML) estimator [60, 50].

Differencing Time Series

The main goal of differencing time series arises from the need to remove non-linear trends from a given process, such that the resulting series can be represented as a linear model. The order can be determined by examining the polynomial order of the original function. Consequently, if the original series possesses a quadratic trend, the resulting stationary series can be obtained by taking the second order difference.

In general, time series can have two different kinds of trends. The first one is resulting from the polynomial order of the process. Additionally, the process can hold a seasonal component. While the polynomial trend can be removed by taking the d^{th} difference of the original time series, the seasonal trend can be removed by taking the d^{th} difference at a specific lag. In case of a yearly trend, a time series with monthly measured data has a seasonal trend with a lag of 12 month. The d^{th} difference at lag 12 will therefore remove the yearly trend.[60, 5]

Autoregressive Integrated Moving Average

Autoregressive Integrated Moving Average (ARIMA) models are a class of models, that deal with time correlated stochastic processes, that arise from a so-called white-noise process.
A white noise process is defined as a set of uncorrelated normally distributed random variables $w_t$with a mean $E[w_t] = 0$ and a constant variance $V(w_t) = \sigma^2$. It can be denoted by

$$w_t \; wn(0, \sigma^2) \tag{2.6}$$

The general purpose of ARIMA models is to map the resulting output of a given time series to a white noise process through different transformations, such as differencing and applying linear filters to past values. [5, 50, 26, 67]

Moving Average Model

The Moving Average (MA) model is a linear filter that produces an averaged value of past realizations of a white noise process. The number of past terms can be specified by a lag q and describes the dependence on former outcomes. The $MA(q)$ process is therefore defined as

$$X_t = \epsilon_t + \sum_{i=1}^{q} \phi_i * \epsilon_{t-i} \tag{2.7}$$

where ϵ_t is a white noise process and the parameters $\phi_1, \dots, \phi_q$ are the coefficients that should be estimated. The MA process is stationary.[5, 60, 50]

Autoregressive Model

The Autoregressive (AR) model of order p is a stochastic process, that is linearly dependent on its own past p Values. More specifically, it consists of a weighted sum of its former p realizations and a white noise process ϵ_t.

The mathematical formulation of an $AR(p)$ process is given by

$$X_t = \alpha + \sum_{i=1}^{p} \beta_i * X_{t-i} + \epsilon_t \tag{2.8}$$

Where α is a constant parameter called intercept and $\beta_1, \dots, \beta_p$ are parameters that need to be estimated.

The model is called autoregressive, because the estimation of the parameters is normally done by using a regression method like OLS or ML estimation.
Since an $AR(p)$ process is not always stationary, it must be ensured that the roots of the characteristic equitation of the process lie outside the unit circle, so that the process converges. Details can be found in [5, 60, 50]

Autoregressive Moving Average Model

The $ARMA(p, q)$ model is a mix of the AR and the MA model with the parameter p specifying the autoregressive and q the moving average orders. If the parameter q is set to zero the Autoregressive Moving Average (ARMA) Model falls back to an AR Process of order p and it is a MA Process respectively if the parameter p is set to zero.
The ARMA Model of orders p and q is given by

$$X_t = \alpha + \sum_{i=1}^{p} \beta_i * X_{t-i} + \sum_{i=1}^{q} \phi_i * \epsilon_{i-1} + \epsilon_t \tag{2.9}$$

Where ϵ_t represents the values of a white noise process, α is a constant term called intercept, $\beta_1, \ldots, \beta_p$ are coefficients of the autoregressive term and $\phi_1, \ldots, \phi_q$ the coefficients of the moving average term respectively. Both ϕ_i and β_i need to be estimated.[5, 60, 50, 26, 67]

Autoregressive Integrated Moving Average Model

The ARIMA model extends the linear ARMA approach, by allowing the dependent process to be non-stationary. In general the concerned time series has a polynomial trend of a certain degree d. If the process possesses such a trend, it can be transformed into a stationary process by taking the d^{th} difference. If the resulting stationary process satisfies an $ARMA(p, q)$ model, the original time series is said to be an $ARIMA(p, d, q)$ process.

The mathematical formulation is given by

$$\Delta^d X_t = \alpha + \sum_{i=1}^{p} \beta_i * X_{t-i} + \sum_{i=1}^{q} \phi_i * \epsilon_{i-1} + \epsilon_t \tag{2.10}$$

Where d denotes the order of differencing of the original time series X_t [5, 60, 50]

Building Autoregressive Integrated Moving Average Models

In order to be able to build an ARIMA model - given a non-seasonal time series - several transformation have to be performed. First of all, it is desirably to stabilize the variance of the process. A common method for correcting an invariability in growing data is to perform a Box-Cox transformation (details can be found in [58]).
When the process is stabilized, the orders p (autoregressive), d (difference) and q (moving average) of the models have to be estimated.
In a first step the order of difference d is determined. A common and very widely used method is the manual detection, by looking at plots and applying a kind of trial and error method. As [5] suggest, the autocorrelation function can be a good indicator for the stationarity of a given time series X_t. That is, if the autocorrelation function comes close to zero very quickly, the process might be stationary. In the other case, a difference $\Delta^d X_t$ might be stationary. As most of the times the difference order does not exceed three, it may be sufficient to only examine the first five orders with different lags between 1 and 20. Common lags are 4,8 and 12.[5]

If it is not possible to manually examine the stationarity of a process, because a model has to be created within an automatic procedure, two test statistics provide the necessary information: The Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test [10] indicates whether a given time series is stationary or not.
In the case of seasonal stationarity, the Canova–Hansen (CH) test [6] can be used.
As it was applied in the manual approach, the trial and error method can be used to automatically obtain a stationary process. This is done by iterating through common lags until both statistics show a satisfying result.

For the estimation of the autoregressive and moving average parameters the Partial Autocorrelation Function (PACF) is of great interest.
The PACF ϕ_{kk} describes the correlation of $(x_t, x_s), |t-s| = k$ where the linear dependence of $\{x_{t+1}, \ldots, x_{s-1}\}$ is removed. Details for obtaining the PACF can be found in [5] or [60]
The procedure of finding the autoregressive order p and the moving average order q is as follows:

- if the process has more than 0 MA terms and no AR parts the ACF comes close to zero after lag q and the PACF decays exponentially.
- if the process has more than 0 AR terms and no MA parts the PACF comes close to zero after lag p and the ACF decays exponentially.
- if the process has both - more than 0 AR and MA terms - the ACF decays exponentially at lag q and the PACF decays exponentially at lag p.

An extensive description about this procedure is provided by [5, 60].

Quality of Fitted Models

As mentioned before the ARIMA model building is often done by using trial and error. Although there are supporting methods like the ACF or PACF, it is often necessary to build several versions with different parameters and compare the fit of each model based on some performance criteria. The first criterion that can be used is the Akaike's Information Criterion, Bias Corrected (AICc), which compares the error of the fitted model with the number of used parameters. The AICc is given by

$$AICc = log(\hat{\sigma}_k^2) + \frac{n+k}{n-k-2} \tag{2.11}$$

where $\hat{\sigma}_k^2$ denotes the ML estimator for the variance, k the number of model parameters and n the sample size
Another common criterion is the Bayesian Information Criterion (BIC) - also called Schwarz Information Criterion (SIC) - which tends to be superior for a large sample size n

$$BIC = log(\hat{\sigma}_k^2) + \frac{klog(n)}{n} \tag{2.12}$$

where $\hat{\sigma}_k^2$ denotes the ML estimator for the variance, k the number of model parameters and n the sample size
Given one of the 2 criteria it is possible to select the best model. [60]

CHAPTER 3

Related Work

As the practical part of this work is based on the research prototype Vienna Runtime for Service Compositions (VRESCo) [54, 51], a short overview is presented here.
As its name suggests VRESCo mostly deals with challenges of SOA. All core features of VRESCo are exposed as Web Services and can be accessed either directly through the SOAP protocol or by using the included client library. VRESCo consists of the following core services:

- The *Query Engine* offers the possibility to search for any entity within the system (e.g. a service) by using the VRESCo Query Language (VQL).
- Clients can subscribe to any event at the *Notification Engine*.
- The *Publishing/Metadata Service* provides possibilities to add metadata and publish it along with a specific service.
- The *Management Service* handles all access control related data.
- Services can be composed according to requirements on QoS attributes using the *Composition Engine*

A comprehensive description of VRESCo can be found in [2] or [51].
A graphical overview of VRESCo is provided in Figure 3.1.

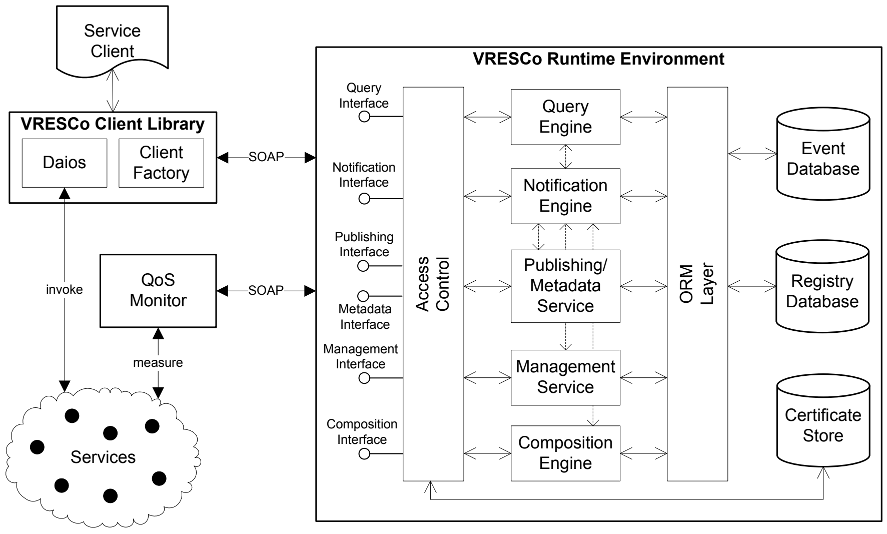

Figure 3.1: Graphical Overview of VRESCo (taken from [54, 51])

3.1 SLA Violation Detection

In general, there are two different possibilities to detect the violation of a specific SLA. The first approach deals with the detection of the case when it already occurred. This can be interesting when it is necessary to have an exact measure of impact that the violation imposes. When the violation of an SLA is known, the source of the problem can be identified and future problems can be avoided. This post hoc method is therefore crucial for the creation of reliable service compositions.
On the other hand, it is desirable to avoid SLA violations as much as possible. Therefore, the second approach tries to detect them before they actually occurred. This is done by using former collected data to deduce the possible future behavior of services. As this kind of detection occurs in an early stage of an execution of a service composition, it provides the possibility to dynamically find and invoke alternative services in case of a predicted violation. It is therefore an important technique to avoid impacts of possible SLA violations.

[48] suggests ADULA - a framework for automated dynamic update of Business Process Execution Language (BPEL) processes. Its basic functionality is based on the automatic transformation of existing BPEL workflows. Once they are modified, the processes are registered within the ADULA framework, which then assigns them a unique name and creates dynamic bindings. Upon an execution of a service composition the transformed version requests a new binding for concerned services and notifies ADULA when it has finished or an error occurred. As opposed to the original service composition, the transformed one is invoked by proxies, which measure performance-related QoS attributes, such as the response time of a specific service. When ADULA receives a notification of success/failure a *Violation Detector* component is

triggered, which then requests QoS parameters from the proxy and performs a statistical hypothesis test to determine whether an SLA is violated or not. Especially the average response time is of great interest as it indicates possible performance degradations. In case of an SLA violation the process is modified according to a specific *Repair Action*. Specifically, if a service is the possible origin of the recognized problem, the *Service Manager* component puts the concerned services into a quarantine and renews dependent bindings. Additionally, the whole process instance is restarted, if the problem originator was a stateful service. Suspended process parts are reused after a certain amount of time or - in case of a failure - when they are repaired. The proposed method of [48] is a good way to prevent multiple violations of the same SLA. However, the ADULA framework lacks of efficient methods to predict possible performance degradations as the statistical models are rather rudimentary. Autoregressive relations or a non-linear trend cannot be described by hypothesis tests, that are based on simple QoS parameters such as the average response time of a service. Moreover, [48] focuses mainly on performance-related attributes, so that it is not possible to predict categorical values of interest.

[59] focus more on the detection of SLAs concerning network QoS and performance parameters. Specifically, the following network metrics:

- One Way Delay (OWD)
- Inter Packet Delay Variation (IPDV)
- Packet Loss Ratio (PLR)

are used to provide an exact description of the current profile/condition of a monitored network. Since the computation of these metrics is linked with bandwidth loss and computational effort, the profiles are created in advance (or updated when necessary) and stored along with a metric whose gathering is less problematic - namely the Inter Packet Arrival Time (IPAT). The advantage of the IPAT is, that it is less resource consuming and can be linked with the before mentioned metrics through a predefined function - called Valid Valid IPAT Distribution (VID). When sufficient VIDs are collected, the network is monitored by sending the IPATs to a specific machine, which then creates an empirical distribution within a given interval and compares it with all known VIDs. This is done by computing the Kullback Leibler Divergence (KLD) (given in the following equation) between the current and every known VID.

$$K(P,Q) = \sum_{i} P(x) log \frac{P(x)}{Q(x)} \tag{3.1}$$

where Q is the distribution to be tested, P the reference distribution and $K(P,Q)$ the degree of similitude between them.

Since the KLD expresses the relative entropy between two distributions, the VID with the minimum outcome is assumed to be the most similar distribution. When the VID of interest is found, an SLA violation is triggered if the profile does not satisfy the specified SLA. The KLD only compares existing distributions which makes it necessary to compute many theoretical distributions before being able to detect possible SLA violations. Moreover, the question of exogenous inputs is left open, as every additional input has to be manually related to the current IPAT distribution. The approach of [59] is therefore not optimal for the automatic detection of SLAs in service compositions.

An instance based approach - which is implemented in the research prototype VRESCo - is suggested by [35]. As mentioned before, one of the core components of VRESCo is the composition engine. Among the dynamic composition of services, it provides the basic functionality for detecting SLA violations. This is done by defining checkpoints, which are triggered at predefined points within the execution of a specific service composition. Each checkpoint contains a predictor component that makes use of historical data and additional information in order to predict a target value of interest. The additional information consists of facts and estimates. While the first represents information that is known before a service invocation takes place, estimates are user defined functions, that deliver an approximately value of measures that are not yet known. A checkpoint is defined within an XML file and contains the following SLA related information:

- An SLO together with its target value and a penalization function if the target value is not met.
- A list of facts and estimates to be included within the prediction of the SLO value.
- A specific prediction model, that is used to derive possible future SLO values.

The approach of [35] is very comprehensive and allows the detection of SLAs of many possible kinds (including the categorical case). The drawback of this method is, that the current implemented prediction models are instance based and therefore not sufficiently considering the behavior of time series. The current prediction models are Multilayer Perceptron Networks and Decision Trees which are trained and evaluated with the help of the WEKA machine learning tool [68]. The contribution of this work extends this approach by modifying the predictor component as well as introducing new prediction models, that are able to model historical data as time series.

[14] suggest the Detecting SLA Violation infrastructure (DeSVi), which helpts to monitor and detect SLA violations in the Foundations of Self-governing Infrastructures (FoSII). FoSII is an infrastructure that proposes methods for the automatic SLA management in cloud environments. To do so, DeSVi introduces a *Runtime Monitor*, which allocates the necessary resources for a service in a Virtual Machine (VM) and deploys it. The service itself has to implement the following three interfaces:

- A *Negotiation Interface*, that is used to manage and establish SLAs.
- The *Application Management Interface*, being responsible for application startup, shutdown and data uploads.
- The *Self-Management Interface*, which describes actions in case of SLA violations.

In order to be able to deal with SLAs, the proposed method makes use of the Low Level Metrics to High Level SLA (LoM2His) framework [13] that allows to transform resource metrics into high level SLA parameters. The transformation rules are implemented by the service provider in a Domain Specific Language (DSL). When a service is deployed, *Host Monitors*, that communicate with the help of the Java Messaging Service (JMS) [40], are used to continuously measure relevant resource metrics. The prediction of possible SLA violations is carried out by the *Runtime Monitor* which uses knowledge databases [3] and Case-based Reasoning (CBR) to deduce future outcomes of SLA parameters. The CBR procedure gathers a formatted instance of a problem - called case - and then looks for similar cases in order to determine possible solutions for the given problem. A knowledge database is used to store the known cases and - if a solution is satisfying - adding or modifying relevant cases. The following steps are executed when a new problem instance arrives:

- retrieve the most similar cases.
- reuse relevant information of the retrieved cases to solve the problem
- revise the deduced solution
- store relevant information inside the knowledge database

The main drawback of the approach of [14] is, that the time series behavior of the SLAs cannot be modeled because CBR does not consider the evolution of SLOs over the time.

3.2 Time Series Analysis

In general, time series can be modeled in different kinds. Since the two mostly considered ways are ARIMA models and state space models, a brief overview of the former is given in the following. State space models consist of a non-observable process X_t, which is not known and produces an output Y_t. It is assumed, that Y_t is linearly dependent on X_t with an additional white noise process ϵ_t. A state space model is defined by two equations. Namely the state space equation:

$$X_{t+1} = A_t X_t + B_t + \epsilon_{t+1} \tag{3.2}$$

which describes the behavior of state X_t over time, and the observation equation:

$$Y_t = C_t X_t \eta_t \tag{3.3}$$

A_t, B_t and C_t are known sequences, η_t and ϵ_t are white noise processes. The estimation of the non-observable state is possible by applying the Kalman-Filter [50]. It is noteworthy, that every ARIMA model can be converted into a state space model representation. See [50] for details.

[30] implemented two different approaches for the automatic forecasting of time series. The first approach is exponential smoothing of underlying state space models. There are 15 different types of exponential smoothing classes, which result from varying the trend- and seasonal components. The following table from [30] depicts the summary of all possible model classes:

		Seasonal Component		
	Trend Component	N (None)	A (Additive)	M (Multiplicative)
N	(None)	N, N	N, A	N, M
A	(Additive)	A, N	A, A	A, M
A_d	(Additive damped)	A_d, N	A_d, A	A_d, M
M	(Multiplicative)	M, N	M, A	M, M
M_d	(Multiplicative damped)	M_d, N	M_d, A	M_d, M

Table 3.1: Possible Exponential Smoothing Classes (taken from [30])

Each of these methods can be expressed by two according state space representations (see [27] for details). When an appropriate model is selected, the conditional mean of a future observation:

$$\mu_{t+h}|t = E(y_{t+h}|x_t) \tag{3.4}$$

can be estimated. x_t contains unobserved components, such as trend and seasonality. In order to derive the conditional mean, the state space representation of the according model is taken, and the parameters of the model are estimated with the maximum likelihood method. The approach of automated forecasting is as follows:

- Apply every appropriate model for a given time series
- Select the best model according to the AICc

- Produce the forecast

The second method involves the automatic estimation of an appropriate ARIMA model. As ARIMA models are described in section Section 2.2 the automated process is briefly summarized. In a first step, the time series has to be tested for stationarity. Therefore, two tests are used: The KPSS test, to determine whether the underlying series is stationary or not, and the CH test, which is applied when the data is seasonal. After the selection of the order of difference, the orders of AR and MA terms has to be estimated. This is done as follows:

1. Try the following four models to start with: ARIMA(2,d,2), ARIMA(0,d,0), ARIMA(1,d,0), ARIMA(0,d,1). The model with the smallest AICc is now selected as the *current model*.

2. Try to vary the model parameters according to the following rules: at a first step, the parameter p or q is varied by ∓ 1. Then both parameters are varied by ∓ 1. At last the constant parameter is excluded or included. Every time a model has a lower AICc than the *current model* it becomes the new *current model*. When no better model is found, the algorithm has finished.

As [30] describes, the number of model classes for ARIMA is quite larger than that of the exponential smoothing. Although exponential smoothing often provides better results, the ARIMA modeling approach is more preferable to use when dealing with the automatic detection of SLA violations, where the target SLO values can have an arbitrary underlying model. The ARIMA method of [30], which was implemented in R - A Statistical Language [56], was used in the contribution of this thesis, as it is considered to be optimal for automatic forecasting.

Another semi-automated ARIMA modeling approach is suggested by [37]. This method was implemented in MATLAB [43] and involves the following steps:

1. Achieve a seasonal time series: This iterative procedure uses the natural logarithm transformation for variance non-stationarity, seasonal differencing for seasonal non-stationarity and normal differencing for mean non-stationarity. Thereby, the transformation methods are applied iteratively and then tested by manual judgment and an automatic hypothesis test.

2. When a stationary time series is obtained the model parameters need to be estimated: This step is again iterative, trying out different model settings and then applying a test. According to prior research work the model orders do not exceed 2 for nearly every possible real-world case. Since the resulting number of models is relatively low, all the combinations are fitted and then, the best one - according to the Root Mean Squared Error (RMSE) - is selected.

The approach of [37] is semi-automated, because a manual judgment for the stationarity of a time series is needed. Therefore, an SPS simulation template was created by using the Simphony modeling environment [25]. Although this method is very similar to the one, that is used in the practical part of this thesis, it is not suitable for this purpose since it requires human interaction.

[11] suggests a mixed state Hidden Markov Model (HMM) in order to represent a time series as the output of a Markov Process. The mixed states are characterized by $\Psi(t) = (s(t), x(t))$ where $s(t) \in S = s_1, ..., s_n$ represents the number of possible states. The state space view of the underlying time series can be written as:

$$\Psi(t+1) = F(\Psi(t)) + \epsilon_\Psi(t) \tag{3.5}$$

$$y(t) = G(\Psi(t)) + \epsilon_y(t) \tag{3.6}$$

where F is a function, that describes the state space dynamics, y an observation, calculated from the state with the help of G and ϵ_Ψ and ϵ_y white noise processes. To find a numerical solution for the mixed state HMM, the model has to be rewritten in the following form:

$$s(t+1) = F_s(\Psi(t)) + \epsilon_\Psi(t) \tag{3.7}$$

$$x_i(t+1) = F_i(\Psi(t)) + \epsilon_{x_i}(t) \tag{3.8}$$

$$y(t) = G(\Psi(t)) + \epsilon_y(t) \tag{3.9}$$

With the constraints that the distribution $P(y(t), x(t)), s(t) = s_i, s(t+1) = s_j$ is multivariate normal with a mean and covariance matrix according to transitions $s_i \rightarrow s_j$, [11] refers to this model as mixed state HMM. When a specific model is chosen, it is possible to estimate its parameters with the Maximum Likelihood method. This approach lacks of an efficient way for the parameter estimation.

[57] suggests an Support Vector Machine (SVM) based approach, where an SVM regression is carried out. Thereby, the basic function $f(x) = sign(wx + b)$ is minimized according to a prediction error from a set of samples $(x_1, y_1) \ldots (x_n, y_n)$. The hyperplane vector w is given by the training set (x_i, y_i) and the coefficients (α_i), which are estimated during the optimization process: $w = \sum \alpha_i y_i x_i$. The optimization problem is given by:

$$\Psi(w, \xi, \xi^*) = \frac{1}{2}(w^T w) + C \sum_{i=0}^{n} \xi_i, i = 1, \ldots, n \tag{3.10}$$

subject to:

$$y_i(x^T, x_i + b) \leq 1 - \xi_i, i = 1, \ldots, n \tag{3.11}$$

$$\xi_i \geq 0, i = 1, \ldots, n \tag{3.12}$$

Kernel functions are used to extend the given linear approach to non-linear problems. In the case of an $AR(p)$ model, the linear kernel function $k(x, y) = xy$ can be used to predict the outcome of an $AR(p)$ process. As [57] describes, the SVM regression tends to have a lower RMSE than the ARMA approach. However, a problem arises from choosing the appropriate kernel function. As there are many possible kernel functions, which need to be chosen manually, it is hardly possible to develop this approach as fully-automated method. Furthermore the training of SVM models tends to be much slower, as the optimization problem can be very time consuming.

A regression model for categorical time series is proposed by [55]. In this case, the response variable Y_t, representing the outcome of possible $q + 1$ categories, consists of q dichotomous variables. That is, each variable can only take the values 0 or 1. Formally defined:

$$Y_{t,j} = \begin{matrix} 1, & Y_t = category_j \\ 0, & else \end{matrix} \tag{3.13}$$

With the help of Y_t, a process $(Z_t, X_t), t = 1, 2, \ldots$ is considered, where X_t represents a q-dimensional response vector $X_t = \{Y_{t,1}, Y_{t,2}, \ldots, Y_{t,q}\}$ and Z_t are covariates. The conditional expectation vector $\mu_t = E(X_t|Z_1, X_1, \ldots, Z_{t-1}, X_{t-1}, Z_t)$ can then be modeled by $\mu_t = h(\eta_t)$, where $h : \mathbb{R}^q \to \mathbb{R}^q$ is a specific response function and $\eta_t = (\eta_{t,1}, \ldots, \eta_{t,q})^T$ is the linear regression term. As [55] describes, the regression term can be written as the following:

$$\eta_t = \alpha + \gamma\mu_{t-1} + \Lambda^T(X_{t-1})\lambda + 1\beta^T Z_t \tag{3.14}$$

where $1 = (1, \ldots, 1)^T \in \mathbb{R}^q$ and $\Lambda(x)$ a q' x q - matrix which expresses the models dependence on the last response X_{t-1}. $(\alpha, \beta, \gamma, \lambda)$ are coefficients that need to be estimated. Since η_t is dependent on former outcomes X_{t-1}, this equation represents a multivariate AR process. [55] further suggest the following response function:

$$h_j(\eta) = \frac{exp(\eta_j)}{1 + \sum_{k=1}^{q} exp(\eta_k)}, j = 1, \ldots, q \tag{3.15}$$

which describes a multivariate logistic regression model. The parameters can be obtained by using the Maximum Likelihood estimator. The basic concept of this method is used in the practical part of this work, since it describes a good way to deal with categorical time series. However, this will be extended, as it only covers the special AR(1) case.

CHAPTER 4

Use Case

In this section, the scenario that is used to test and evaluate the contributed work is presented. As the practical part is integrated into the research prototype VRESCo, the results will be related as much as possible to former research work. Especially the outcome of [54] serves as interesting comparison factor.
In order to carry out a suitable evaluation, the same use case as it was employed in [54] will be taken here. Details thereof are emitted; however, a short overview is presented in this section.
The scenario consists of a customer, that orders a specific product from a reseller. Based on the specification details of the product - called RFQ (request for quotation) - , the reseller carries out a cost and time estimation of the order process. When the estimation is satisfying, and the customer confirms, the reseller has to check whether all required components are in-house. If this is not the case, the missing parts have to be requested from a set of external suppliers. Once the external delivery has finished, the composition of the product is carried out and an optional quality assurance takes place. The shipping of the complete product is now initialized, while an invoice is created. Finally - when the customer has delivered the payment according to a chosen payment option and the desired product is delivered - the order process has finished. In order to simulate the scenario as realistic as possible, a Windows Workflow (WWF) with the following WCF services is used:

- Assembly Planning Service: calculation of assembly costs and time
- Warehouse Service: availability check and reservation of required parts
- Construction Service & External Construction Service: planning of the (external) product assembly
- Customer Service: settlement of the payment options & details
- Supplier Service: availability check of external parts, estimation of the shipping time, purchase order of external parts

- Banking Service: payment processing
- Billing Service: calculation of additional costs, invoice generation
- execution of the shipping process

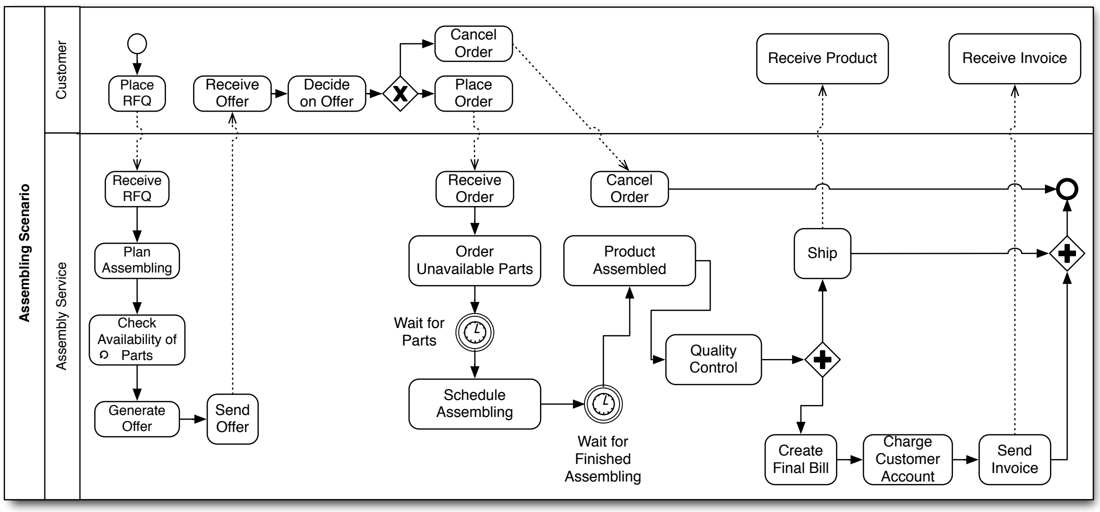

Figure 4.1: Graphical Overview of the Assembly Process (taken from [54])

A complete workflow diagram can be found in Appendix A.1

Challenges

To have a clear look on the challenges provided by this example, five SLAs are established. These SLAs are signed between the reseller and its customers. The five according SLOs are as shown in Table 4

Each of these SLOs is mapped to SLAs by specifying target values that are penalized in case of a violation, as shown in Table 4

It is clear that the reseller tries to avoid the violation of these SLAs, since it is linked with profit loss. Furthermore, a key client could choose another reseller, if some or all SLAs are violated frequently. In order to achieve such a goal, the service composition can be reorganized by interchanging (or omitting) specific services, so that the balance between costs, quality and shipping time can be preserved. If - for example - the price of the product falls below the estimated price, but the delivery time is expected to be higher, the reseller could use a faster shipping service, which in turn is more expensive. As VRESCo enables the dynamic adaptation of service compositions [54], the only further requirement is a reliable method to predict the outcome of future SLO values, such that the service composition can be reorganized during execution. The forecast of the specified SLOs poses some problems that need to be considered. These problems are consequently addressed by the contribution of this work:

SLO #	Name	Description	Measure
1	Delivery Time	The time between receiving an RFQ (request for quotation) and the final product delivery	Working Days
2	Payment Processing Time	The time between generating an invoice and the settled payment	Working Days
3	Shipping Processing Time	The time between the reseller sends the product and the customer receives it	Working Days
4	Quality	The product has the specified quality	Category
5	Additional Costs	The cost overflow of estimation and real costs	Overflow Percentage

Table 4.1: Possible SLOs

SLO #	Target Value	Penalty
1	<= 6	The reseller has to offer a discount of 5% for each additional day. Maximum: 20, No addition to SLO #2 or #3
2	<= 3	The reseller offers a 3 % disount per additional day. Maximum: 15%. No addition to SLO #1 or #3
3	<= 3	The reseller has to offer a discount of 5 % per additional day. Maximum: 20%. No addition to SLO #1 and #2
4	= requested quality	If the quality is not satisfying, the reseller has to reship an appropriate product within 7 days.Additionally a discount of 5% is offered
5	<= 5	Reseller cannot charge more than 5% cost overflow

Table 4.2: SLO Target Values and their Penalties

- A recent market research of the reseller has shown, that the overall workload of a year for all delivering partners is constant. However, the monthly workload varies very strong. Specifically, the shortest *Shipping Processing Time* is expected to be in the winter season, following a constant increase until the summer season, where the workload has its maximum in July or August. The *Shipping Processing Time* then decays, again constantly, up to the winter season having its minimum in January or February. A graphical representation of the workload research can be found in Figure 4.2

- The *Payment Processing Time* depends on available banking services. Since considered banks process thousands of transactions per day, the *Payment Processing Time* follows a random distribution. If a client initializes several transactions in a row, the payment process takes longer, because of a temporary increase of workload. However, in the case of key clients, the bank assigns a higher priority to the concerned payments and the *Payment Processing Time* is shorter. Consequently, banking transactions are not only randomly distributed, but also follow an AR process.

- *Additional Costs* mainly depend on two factors. Firstly, the priority - which is chosen by the client - influences the delivery time of the product. Consequently, if a customer orders a product with a high priority, faster and more expensive services have to be used and the costs are more likely to be higher than expected. Secondly, the overall workload of the reseller delays products with higher priorities. This requires - again - faster and more expensive services. It follows that *Additional Costs* are both: AR, because key customers order frequently at the same priority in recurrent time points, and MA, because the overall workload of the reseller changes in weekly cycles.

- The *Quality* of the Product is classified in categories. The forecasting approach has to be able to deal with categorical time series. Since the same customers order products of the same quality in recurrent time points, the *Quality* of a Product is an AR process.
- The *Delivery Time* of a product depends on all other SLOs - which are time series. Therefore, the *Delivery Time* is a composition of multiple time series and follows an ARIMA process of unknown orders.

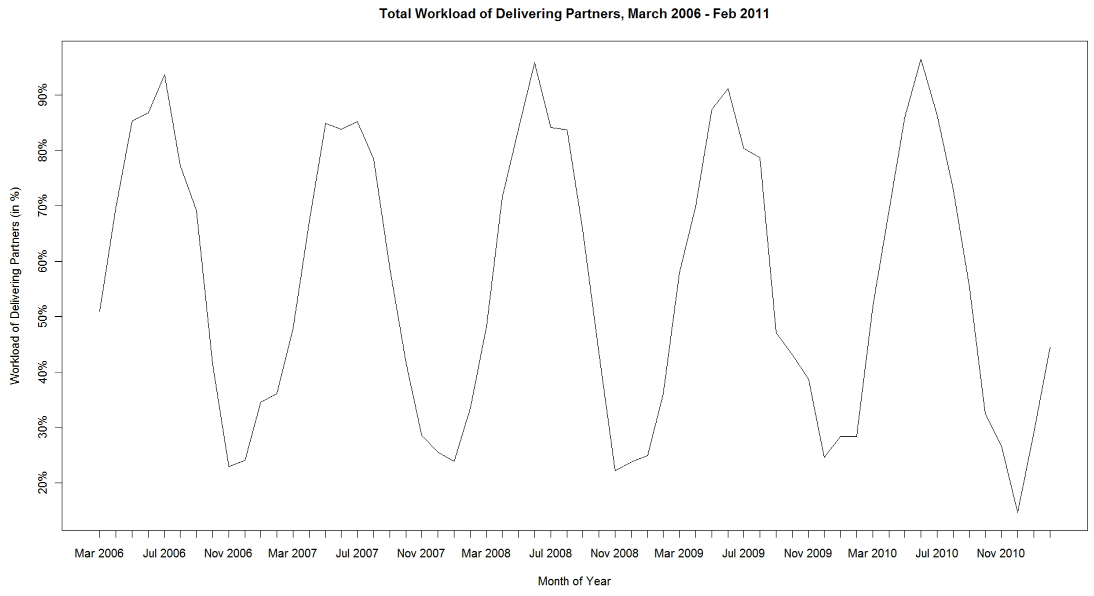

Figure 4.2: Workload of Delivering Partners

CHAPTER 5

Contribution

The main contribution of this thesis is the development of a methodology to automatically create and train a statistical model, given a specific time series. Thereby, the characteristic of each new instance is different, so that the creation of selection and testing mechanisms, that require no human intervention, is needed. The generalized class of ARIMA models build the base of the algorithms, which are integrated and tested within the $C\#$ research prototype VRESCo.

5.1 Overview

In this section the architectural essentials of the contribution are briefly depicted. The theoretical model is implemented, as already mentioned, in the research prototype VRESCo.
All relevant data is coming from the *Composition Engine* whose main purpose is the automatic composition and execution of Windows Workflows. In order to be able to define and measure SLAs an XML configuration file is defined and passed to the *Prediction Manager* component. It contains all necessary information such as the defined facts, metrics and at which point the prediction should take place. The *Prediction Manager* then passes the configurations as well as historical SLO values to the *Checkpoint Predictor*. As the *Checkpoint Predictor* is hooked into the execution flow it has the possibility to retrieve facts and estimates for a specific SLOs. When the work flow has started its execution, the *Checkpoint Predictor* component collects necessary facts and estimates and invokes the training method of the *Forecasting Service*, while resuming the execution of the workflow meanwhile. When it is called, the *Forecasting Service* first makes a lookup to the *Forecasting Database* to retrieve previously measured SLO instances and prediction models. Based on the collected information it then builds a new - if it was not created before - or trains an existing model respectively. The model building process includes the transformation of data, such as converting categorical variables into a set of dichotomous vectors. The actual training takes place in the *Forecasting Engine* which is connected to the Service through a special library (see section 5.3 for details). Once it is trained the model is persisted in the *Forecasting Database* and the *Forecasting Service* waits for further requests.

Every time the *Composition Engine* reaches a checkpoint the execution is suspended and the SLO prediction process, which is evaluating the next service that should be called, is launched. At this point the current facts and estimates of the concerned service are passed to the prediction method of the *Forecasting Service*. After retrieving the beforehand trained model from the *Prediction Database* the *Forecasting Service* makes a call to the *Forecasting Engine* and passes the model and - if available - current facts and estimates. In the case of missing facts or estimates a statistical model without any external values (whether or not they were collected before) is created. In the other case the additional information is included as well. After the *Forecasting Engine* has finished the prediction process necessary information such as the model dimensions, the statistical error and especially the prediction value itself is returned to the *Forecasting Service* which in turn passes the results to the *Checkpoint Predictor*.
If the resulting prediction indicates that an SLA will be violated by the next service that is scheduled within the execution, the composition engine can - depending on the configuration - invoke an alternative service to continue the work flow or resume without any modifications in the other case. Additionally, the retrieved data can be passed to a client where the data can be further analyzed.
Figure 5.1 clarifies the described overall architecture of the contributed work.

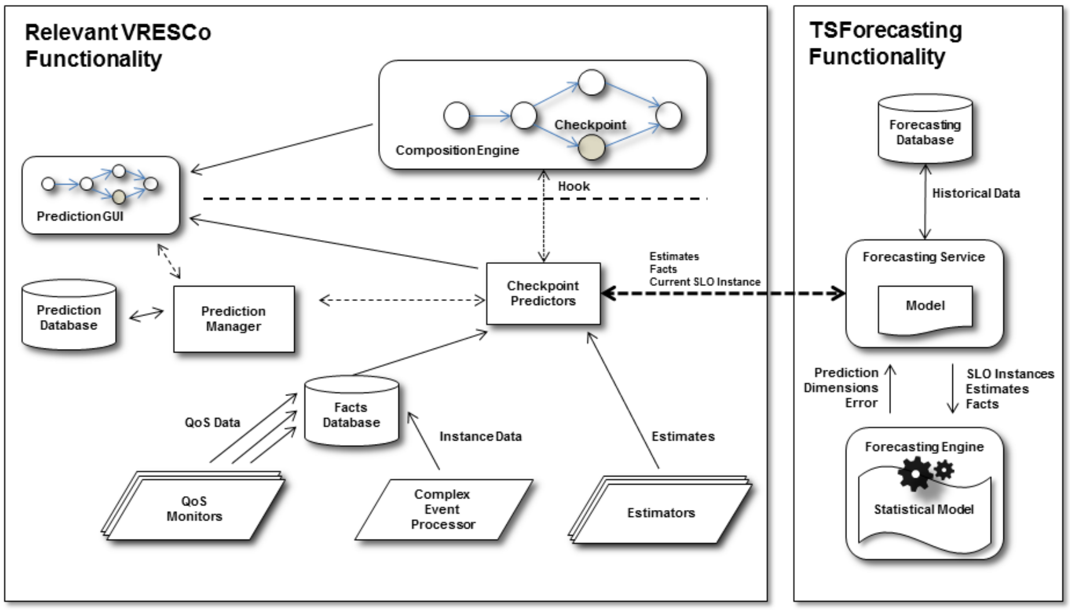

Figure 5.1: Overview of the Model Architecture (adapted from [35])

For the better understanding an example execution - according to the use case of section 4 - is depicted in the following. The graphical representation is depicted in figure 5.2, which contains an extended flow chart, not only capturing the steps of an order, but also showing the facts and estimates that are relevant for the prediction process. The work flow is fully executed in course of an incoming order, and the prediction is carried out for the *Delivery Time*. The target value *Delivery Time* is chosen, because the reseller has signed an SLA that commits him to offer a discount of 5% for each day, that the order is delayed. If a violation of the SLA is predicted, the reseller wants to change the normal shipping modus to a faster one. The boundary for the *Delivery Time* was negotiated beforehand with the customer and amounts to 10 days (80.000 time units). It is important to mention that one day corresponds to 8.000 time units of the reseller (for internal calculation purposes).

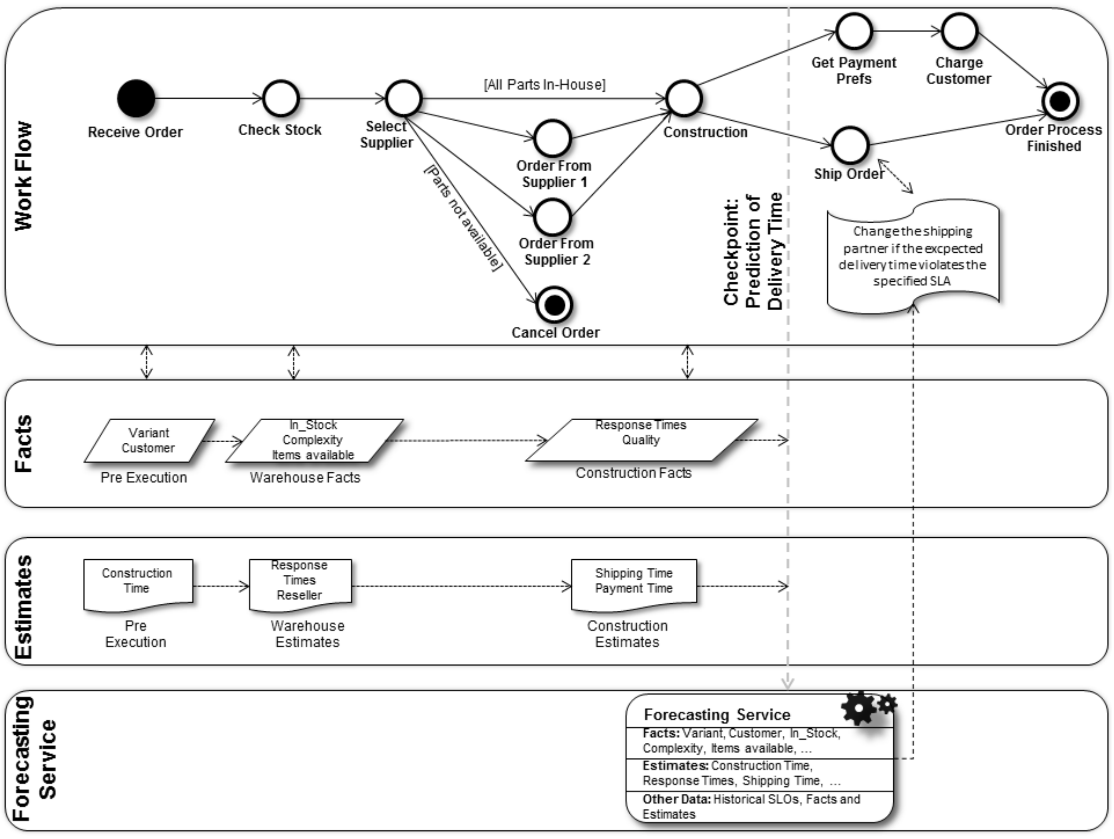

Figure 5.2: Example Execution of the Work Flow with Delivery Time as Target SLO (adapted from [35])

The customer *111-11234* is ordering a *van* with the product variant *deluxe*. As the order is received by the reseller, the first step is to check the warehouse and find out whether or not all products are available. In this phase of the work flow execution, the provided information is relatively sparse: some facts like the total number of items that the reseller needs for the assembling process or the complexity of the product. Moreover, an estimate of the average delivery time of the two external suppliers is known at this time. As not all required parts are in-house the total response time of the external suppliers can be measured in the construction step. At this point the application provides the estimates for the *Shipping Time* and the *Payment Processing Time* which strongly influence the overall *Delivery Time* and therefore are important factors for the following prediction process. Before executing the lasts steps in the work flow it is suspended and the prediction takes place. Therefore all collected data as listed in the following is sent to the *Forecasting Service*:

- The Customer: 111-11231
- The Product: van
- Complexity of the product: 5 (ranging from 0 to 15)
- Total number of items the van consists of: 9
- The estimated average delivery time of the external suppliers: 174
- The measured total delivery time of both external suppliers: 365
- The overall construction time for the product (including external deliveries): 6399
- The estimated payment processing time: 2540
- The estimated shipping time: 1200
- A list of 100 previously measured total *Delivery Times* (SLOs)

The most important information is extracted from the historical SLO values. With 100 historical values, the prediction is relatively reliable and therefore serves as a good decision help.

As the prediction of 10.78 days (86.229 time units) is violating the specified boundary of 10 days (80.000 time units), the reseller is now able to change the transportation type to *Express Shipping* in order to deliver the product in time. As the transportation type *Standard Shipping* is much cheaper this option is only chosen if the delivery time is expected to be above the specified boundary. Under the premise that the SLA violation for the *Delivery Time* will occur, the reseller is able to avoid this event and consequently lowers the overall costs: The beforehand agreed price discount of 5% is more expensive than the faster delivering option. However, the express delivery is unnecessary in most of the times, which is the reason for that the reseller tries to use the transportation type *Standard Shipping* whenever it is possible.
As in most of the times the prediction is accurate and the total *Delivery Time* of 10.54 days (84.280 time units) is in fact too high when using the standard delivery option. If the reseller chooses to change the transportation type to *Express Delivery*, the SLA will not be violated leading to lower costs.
Given scenarios like this one, the *Composition Engine* of the VRESCo framework has the possibility to automatically seek for better alternative options. In this case the delivering service would have been automatically changed to *Express Shipping* resulting in a cheaper overall ordering process.

5.2 Model

The training and evaluation of the statistical model mainly takes place in the *Forecasting Engine* component of section 5.1. As stated before, the general class of ARIMA models is used to capture the mathematical relations, that a specific time series expresses. This is mainly due to the fact that many real-world processes exhibit dependencies on their own past. The contributed model analyses SLO values of SLAs, that were specified along with a set of facts and estimates. As the proposed model is not constrained to numerical values only, the case of categorical time series is also considered. The mathematical formulation of the statistical model is repeated in the following equation:

$$\Delta^d X_t = \alpha + \sum_{i=1}^{p} \beta_i * X_{t-i} + \sum_{i=1}^{q} \phi_i * \epsilon_{i-1} + \epsilon_t \qquad (5.1)$$

where X_t represents the prediction value, $\sum_{i=1}^{p} \beta_i * X_{t-i}$ the historical SLO values, ϵ_i the error terms and α a constant term.

In a first step, all historical realizations of SLOs, along with the according facts and estimates, are collected. Facts and estimates are used as exogenous variables that can influence the behavior of the SLO development in time. All categorical variables are transformed into several dichotomous variables, according to the number of possible classifications. That is, for every categorical instance, a vector of same length as the number of categories is created. This vector consists of but zeros, except on the index of the corresponding category. If, for example, there is a categorical variable *Quality* with possible classifications A,B and C, then an instance of value B would be translated into its according vector $(0, 1, 0)$. If all required data is available, two different cases have to be considered:

- The dependent variable is categorical and therefore requires a multivariate model.
- The dependent variable is numerical.

The categorical case only considers multivariate AR models and is therefore relatively simple: A first *current model* is evaluated, by using least squares regression with one AR lag. The *current model* is now iteratively compared to results of other models, having a lag ranging from two to six. The *current model* changes, if the current AICc is lower than the new one, or if the former could not be estimated due to an error. As the outcome of this method is varying around -2 and 2, the resulting vector has to be transferred back to a valid vector of dichotomous variables, where only one entry is set to 1 while the others are zero. This is done by assuming the category with the highest value to be the most probable one. If, for example, the outcome of the categorical forecast is (1.2, 0.3, 0.02) according to categories (A,B,C), then the most probable category is A and the vector is transformed into (1,0,0). A graphical representation of the procedure is given in Figure 5.3.

The model estimation process of numerical time series is more complicated, as it also considers MA terms, trends and seasonality: In a first step, it has to be determined whether a series

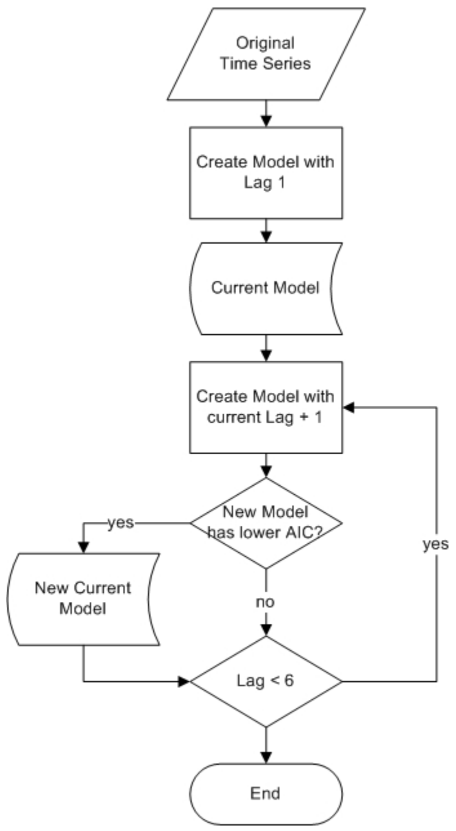

Figure 5.3: Procedure for Estimating the best VAR model

is stationary in its mean. This is done by applying the KPSS test. If the test is showing, that the underlying series is not stationary, then the first difference is taken, and the resulting output is tested again. When the mean is stabilized, the process is tested with the CH test to detect seasonal non-stationarity. If this is the case, a sort of trial and error procedure takes place, to find the problematic lag. The lags 4, 6, 8, 12 and multiples of them have proved to be the most frequent. When the problematic lag is identified the first difference is taken, and the CH test is applied again until the underlying series is fully stationary.Figure 5.4 depicts the process stabilizing procedure:

When a stationary time series is obtained, the model orders have to be estimated. This is done within the following steps:

- Choose the model with the lowest AICc among the following: ARIMA(2,d,2), ARIMA(0,d,0), ARIMA(1,d,0), ARIMA(0,d,1). (Figure 5.5)

- Try to find a model with a lower AICc by varying either the AR or the MA orders by ± 1. If a better model is found, it becomes the current model and the step is repeated. If no better model can be found, the process proceeds to the next step. (Figure 5.5)

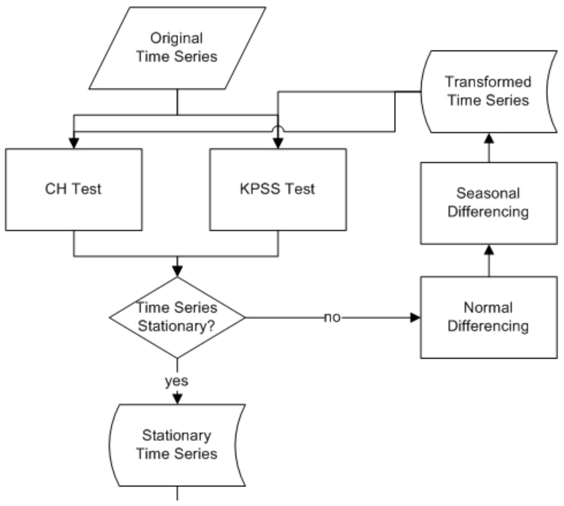

Figure 5.4: Partial Step of the Model Building Process to obtain a stationary Model

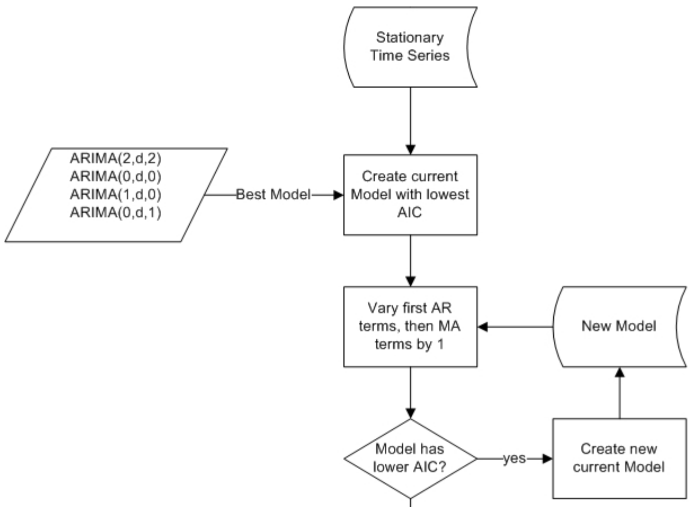

Figure 5.5: Partial Step of the Model Building Process to choose the Initial Current Model and find a better Model by varying AR and MA Terms

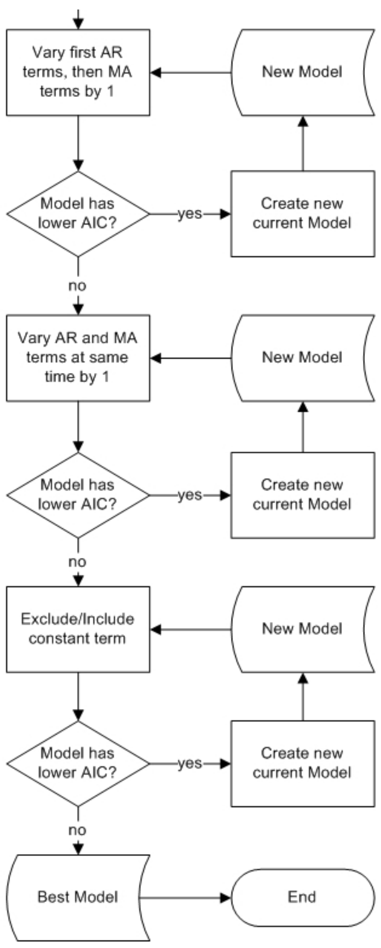

Figure 5.6: Last Partial Step of the Model Building Process to find a better Model by varying MA and AR Terms and including/removing an Intercept

- Try to find a model with a lower AICc by varying both: the AR and the MA orders by ± 1. A better result will again become the current model and the step is repeated. If no better model can be found, the next step is executed accordingly. (Figure 5.6)
- Try to find a model with a lower AICc by excluding or including the intercept and repeat this step if a better model is found. In the other case, the procedure supposes the current model to be the best one and the procedure finishes. (Figure 5.6)

As an example, the equation for an ARIMA(2,2,1) model with 99 historical realizations is given in the following:

$$\Delta^2 X_{100} = \alpha + \beta_{99} * X_{99} + \beta_{98} * X_{98} + \phi_{99} * \epsilon_{99} + \epsilon_{100} \tag{5.2}$$

$\Rightarrow$

$$X_{100} = \Delta^2 X_{100} - X_{99} = \Delta^2 X_{100} - \Delta^2(\alpha + \beta_{98} * X_{98} + \beta_{97} * X_{97} + \phi_{98} * \epsilon_{98} + \epsilon_{99}) \tag{5.3}$$

where X_t are the realization values of the SLOs, β_i the values of a white noise process and α, β_i and ϕ_i the coefficients to be estimated.

As the AICc serves as main performance indicator for the accuracy of a model, its equation (from chapter 2) is repeated in the following:

$$AICc = log(\hat{\sigma}_k^2) + \frac{n+k}{n-k-2} \tag{5.4}$$

where $\hat{\sigma}_k^2$ denotes the ML estimator for the variance, k the number of model parameters and n the sample size

To preserve the performance and avoid endless loops, the maximum overall orders of the model does not exceed 5.

For both of the mentioned methods, two models are fitted: namely one with and one without exogenous inputs. This is done to prevent prediction errors. The measured data comes from monitors of specific service compositions that can partially fail. If this is the case, the involved monitors cannot measure any data. As the forecasting mechanism needs current inputs of exogenous inputs, it would raise an error if any of the needed data is not provided. In order to overcome that issue, the contributed model first tries to make a prediction, involving the influencing external factors. If this is not possible, the basic model which considers the value of interest only, takes over and returns a forecast.

5.3 Implementation

The contribution of this work is implemented within the research prototype VRESCo, which is based on the programming language $C\#$ .NET 3.5 [45] and uses WCF as communication model. Thereby, the project *TSForecasting* is exposed to the VRESCo framework as a WCF service. It is directly invoked by the *Service Composition* component. The project makes use of the statistical language R [56], mainly because it is a powerful, open source application that includes a rich set of statistical operations. Furthermore, R possesses a dynamic packaging system, which offers the possibility to download and use many open source libraries. It is published under the GNU General Public License GPL 2 [23]. As R is written in C, its Dynamic Link Library (DLL) cannot be directly used within a $C\#$ application. To overcome that issue, the open source project R.NET [1] is used to call the unmanaged code of the R DLL . R.NET is published under the GNU General Public License GPL2 as well. The project is a modified version in order to be compatible with $C\#$ .NET 3.5. Unnecessary parts, which rely on .NET 4.0, were removed and the whole library was recompiled and integrated within the *TSForecasting* project.

As possibly many service compositions are handled at the same time, the according data structures for SLO forecasting are established as follows:
The fundamental part of the *TSForecasting* project is a *TSProcess*, which can be uniquely identified by its Globally Unique Identifier (GUID). A *TSProcess* stores all relevant information for

predicting the future values of an SLO. It contains an SLO ID, the underlying model, a current prediction value, which is set when a new prediction is retrieved, and a set of time series instances. As a *TSProcess* is responsible for creating and storing new models and training them, it also contains the following public methods:

- *updateInstance:* This method is called every time before a model is trained. It takes as argument a specific instance that has the same description (*TSField*) as one of the contained *TSInstances*. The new Instance is then merged with the corresponding existing one, by adding new realization values, if not already existent.
- *trainProcess:* As the data consolidation is already done by *updateInstance*, the *trainProcess* method does not have any arguments. In fact, it only delegates the call to the underlying model and then returns without any result.
- *doPrediction:* The prediction follows the same scheme as the model training. The *TSProcess* delegates any prediction call to the underlying model, then fetches the prediction result and stores it in his member variable *predictionValue*.

Instances are encapsulated within a *TSInstance* class. Among a numerical ID, they contain a detailed field description (*TSField*), a list of realization values, and the boolean variable *isDependentInstance* to indicate whether it is the SLO value to be predicted or not. Moreover, the *TSInstance* class offers the following important methods:

- *addValue:* This method takes a *TSValue* as argument and appends the value at the end of the existing list of *realizationvalues*.
- *mergeInstance:* Whenever a big amount of data has to be unified, this method should be called with another *TSInstance* as argument. It merges all the data of the new *TSInstance* with its own. Duplicates are removed by comparing the *instanceIdentifier* of the new *TSValue* with existing ones.
- *getDoubleSeries:* As the further data processing requires numerical values, this method converts the internal structure of the stored time series into an array of double values. The numerical array can then be passed to the *R* environment.
- *getDoubleSeriesList:* When the underlying instance is of nominal nature, the transformation into a list of dichotomous vectors is done by this method.

As mentioned, the *TSField* is responsible for storing all relevant information about time series. Such information includes the type of the value of interest, which is specified with the help of the contained enumeration *TSFieldType*. The currently supported types are: NUMERICAL, DECIMAL and NOMINAL. If the *TSFieldType* is specified as NOMINAL, a list of possible classifications has to be included. Moreover, the *TSField* contains methods to transform given values, so that they can be used for the prediction process. In the case of a NOMINAL field, the transformation method takes a string value, which has to be available in the list of possible classifications. It then returns a vector of dichotomous variables which only contains zeros, except at the index, where the corresponding classification is located. The *TSField* offers the following important methods:

- *getValue:* As the realization values of time series are internally stored within string variables, this method offers the possibility to transform the string representation of numbers into real numbers. Additionally, the method rounds a non-decimal value if the *TSFieldType* is specified as DECIMAL.
- *getValueForNominal:* This method should be called, if the *TSFieldType* is set to NOMINAL. It converts categories into a vector of dichotomous variables. It will throw an Exception if it is called within a non NOMINAL *TSField* instance.

As one can see in figure 5.7 the *TSProcess*, *TSInstance*, *TSField*, *TSValue* and the service classes *IForecastingService* and *ForecastingService* together implement the *Forecasting Service* component that is depicted in section 5.1.

The model of a *TSProcess* is described by a subtype of the *IForecastModel*. This interface defines the two most important methods:

- *train:* This method consolidates the available data of all *TSInstances* and transforms them into numerical lists in order to build up a suitable model. Its implementation (*ARIMAModel*) does so by forwarding all relevant information to the *RCommunicator* and storing the byte representation of the model that was built by the *R* environment.
- *doPrediction:* this method takes the future values of all required estimates and facts, and then calculates an actual 1-step forecast for the SLO of interest. The current implementation restores the byte representation of the beforehand created model and forwards the values into the *R* environment. When a prediction is done it gathers the result from the *R* environment and stores it in its member variable.

Before calling one of those methods, at least the series to be predicted has to be set. Additionally, the external inputs - if present - can be specified as well. As mentioned above the current only implementation is the *ARIMAModel*, which fits an appropriate ARIMA model and estimates its parameters. To do so, the *ARIMAModel* makes use of the helper class *RCommunicator*, which handles all communication between the *TSForecasting* project and *R*. In detail, the *RCommunicator* has two responsibilities:

- Converting $C\#$ data types to transmittable information, so that it can be sent to the *R* environment, and converting information that is received from the *R* environment back to $C\#$ data types.
- Handling all the communication between the *R* environment and the project.

The communication between the *R* environment and the *RCommunicator* is implemented within two public methods:

- *trainModel:* this method takes as argument an instance of the *ARIMAModel*, collects all relevant information from it, and then sends it to the *R* environment. The created model is then serialized within the *R* environment, sent back in form of a byte representation and then further returned.

- *doPrediction:* As in the training method, an instance of the *ARIMAModel* is passed as an argument. Additionally, the numerical representation of current facts and estimates is forwarded (if present). The model then gathers the serialized *R* model and sends it to the environment. Once the model is restored, the prediction takes place. The results are directly stored within the instance of the *ARIMAModel*, which is then returned.

Together with the two model classes *IForecastModel* and *ARIMAModel* the *RCommunicator* implements the functionality for the *Forecasing Engine* component of the architectural overview (see section 5.1 for details), whereas the model classes reflect the functionality of the *Statistical Model*.

Additionally, the project implements the WCF service interface *IForecasingService*, which specifies the basic contract for any other WCF service, that wants to use its forecasting features. It is the main class of the *Forecasting Service* component of section 5.1 and exposes the following methods:

- *initProcess:* Creates and stores a new *TSProcess*. Thereby the desired *modelClassifier* - which equals to the fully qualified class name of the model implementation class (currently only *ARIMAModel*) - and the SLO ID - for which the prediction should be done - are passed as an argument. If *isAdaption* is set to true, it searches for an existing *TSProcess* with the ID *processIDToUse*. Once the *TSProcess* is created or successfully found, its ID is returned, so that the originator of the call can use this process to train and predict a model.

- *trainProcess:* Builds up and trains the model for a given *TSProcess*. Thereby the new time series instances are passed as parameters and then merged with the existing data. The method then calls the *trainProcess* method of the underlying *TSProcess*.

- *makePrediction:* Like the *trainProcess* method, also this one delegates the call to the underlying *TSProcess*. Additionally it forwards the current estimates and facts, that are needed for the prediction process of models with external regressors (if given), and then returns the prediction result to the caller.

- *deleteProcess:* deletes an existing process, including all stored historical data.

The implementing class - *ForecastingService* - uses a simple persistence mechanism based on NHibernate [20] which is implemented within the *ProcessDao* and represents the specific implementation for the *Forecasting Database* component of the architectural overview (see section 5.1). It is not only responsible for persisting a single *TSProcess*, but rather a whole object tree that is included within a *TSProcess*. This means that also all values, field descriptions and

even the serialized *R* model are included. Since NHibernate requires all entities to possess a serializable ID, the classes *TSInstance*, *TSField* and *TSValue* implement the *Persistable* Interface whose only purpose is to ensure a valid ID type. The abstract class *SimpleGenericDao* possesses generic persistence mechanisms so that the implementation of methods for new entity types - in case of a program extension - is as easy as possible.
Since it can be called by multiple clients concurrently, the *TSForecasting* project includes a locking mechanism for shared resources, such as the *R* environment. To prevent failures within the *R.NET* library, a short pause of 100 milliseconds is made to allow the garbage collection to free unused resources.

Figure 5.7 depicts the Unified Modelling Language (UML) class diagram [24] of the *TS-Forecasting* project.

As mentioned before, the contribution of this thesis makes use of special *R* packages, that were specifically built for the automatic creation of ARIMA models: Namely the *dse* package [22], the *forecast* package [29] and the package *tsforecasting* - which was created within the contribution of this thesis. The main purpose of the *tsforecasting* package, is to merge the capability of handling multivariate AR models (*dse*) with the creation of univariate ARIMA models *forecast*. Its source code can be found in Appendix B.1.

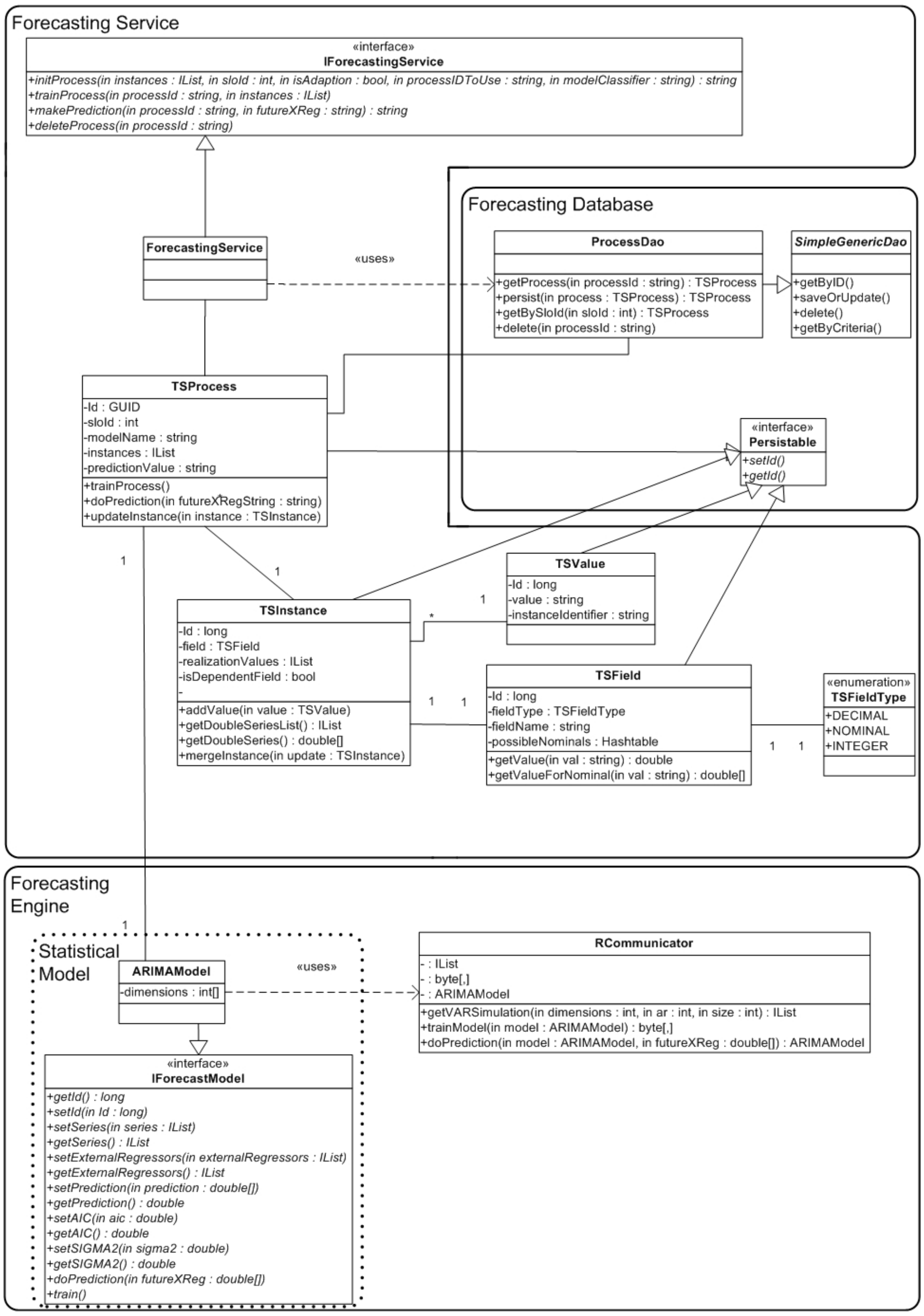
Forecasting Service
«interface»
IForecastingService
+initProcess(in instances : IList, in sloId : int, in isAdaption : bool, in processIDToUse : string, in modelClassifier : string) : string
+trainProcess(in processId : string, in instances : IList)
+makePrediction(in processId : string, in futureXReg : string) : string
+deleteProcess(in processId : string)
ForecastingService
«uses»
Forecasting Database
ProcessDao
+getProcess(in processId : string) : TSProcess
+persist(in process : TSProcess) : TSProcess
+getBySloId(in sloId : int) : TSProcess
+delete(in processId : string)
SimpleGenericDao
+getByID()
+saveOrUpdate()
+delete()
+getByCriteria()
TSProcess
-Id : GUID
-sloId : int
-modelName : string
-instances : IList
-predictionValue : string
+trainProcess()
+doPrediction(in futureXRegString : string)
+updateInstance(in instance : TSInstance)
«interface»
Persistable
+setId()
+getId()
1
1
TSValue
-Id : long
-value : string
-instanceIdentifier : string
1
*
TSInstance
-Id : long
-field : TSField
-realizationValues : IList
-isDependentField : bool
-
+addValue(in value : TSValue)
+getDoubleSeriesList() : IList
+getDoubleSeries() : double[]
+mergeInstance(in update : TSInstance)
1
1
TSField
-Id : long
-fieldType : TSFieldType
-fieldName : string
-possibleNominals : Hashtable
+getValue(in val : string) : double
+getValueForNominal(in val : string) : double[]
1
1
«enumeration»
TSFieldType
+DECIMAL
+NOMINAL
+INTEGER
Forecasting Engine
1
Statistical Model
ARIMAModel
-dimensions : int[]
«uses»
RCommunicator
- : IList
- : byte[,]
- : ARIMAModel
+getVARSimulation(in dimensions : int, in ar : int, in size : int) : IList
+trainModel(in model : ARIMAModel) : byte[,]
+doPrediction(in model : ARIMAModel, in futureXReg : double[]) : ARIMAModel
«interface»
IForecastModel
+getId() : long
+setId(in Id : long)
+setSeries(in series : IList)
+getSeries() : IList
+setExternalRegressors(in externalRegressors : IList)
+getExternalRegressors() : IList
+setPrediction(in prediction : double[])
+getPrediction() : double
+setAIC(in aic : double)
+getAIC() : double
+setSIGMA2(in sigma2 : double)
+getSIGMA2() : double
+doPrediction(in futureXReg : double[])
+train()

Figure 5.7: Class Diagram of the TSForecasting Project

CHAPTER 6

Evaluation

In the following section of this thesis a comprehensive evaluation of the conducted tests is detailed. The example use case of section 4 was implemented within a Windows Workflow and executed 500 times to collect a necessary amount of data. During execution the monitoring mechanisms of VRESCo were used to measure facts and estimates as well as the resulting SLO values, being present in the end of each work flow execution. As already described in section 4 the following SLO values were used to test the prediction accuracy of the developed model:

1. The *Payment Processing Time* of the customer, when purchasing an ordered item.
2. *Additional Costs* of a product.
3. The overall *Delivery Time*.
4. The total *Shipping Time*.
5. The *Quality* of the product.

The goal of this evaluation is to show that the use of the implemented prediction method opens a new possibility to effectively save costs within the automated execution of service compositions, by performing a sample run and comparing the predicted classifications of the SLOs (*SLA violated / SLA not violated*) to the real outcome. The classifications are based on beforehand specified SLO boundaries. Since all parameters of an SLA are normally negotiated with key customers or arbitrarily chosen by the service provider, they are determined in a way that the violation of an SLA is very unlikely. It is obviously advantageous to proceed like this, because a service provider is normally never willing to sign a contract whose key requirements cannot be fulfilled. Moreover, a service consumer usually chooses reliable services. That is, services that are unlikely to violate the specified contract. In order to simulate business cases as realistic as possible the SLO boundaries for the evaluation were chosen in a way that the violation of a service occurs very rarely. In addition to the evaluation of the prediction accuracy the model training times are presented to demonstrate the applicability in real world cases where the time component may be critical.

All tests have been carried out on a Laptop with 4GB RAM, Intel Core 2 Duo P8400 processor @ 2.26 GHZ, Windows 7 Professional 64-Bit. The core services as well as the work flow were executed on the same machine. IIS 7 was used to host the core services, whereas the contributed work was deployed directly into a process within the *System.ServiceModel.ServiceHost* class. The mechanisms to simulate the behavior of ARIMA models was integrated into the existing project *AtomicAssemblyServices*, which is responsible for publishing the required services.

6.1 Performance

The training processes are growing constantly with the time. As the plots from of figures 6.1 - 6.3 show, the duration to train a specific model is growing linearly with the number of available past realization values. The number of instances ranges from 1 to 500 whereas the duration has its minimum at below one and its maximum under 45 seconds. As all tests were conducted on the same physical machine, the performance of training times is strongly fluctuating. However, they are expected to be more constant when the *TSForecasting* project is published on a single machine. Another performance influencing factor is the number of external regressors. When a current value of an estimate or fact is unknown, the prediction mechanism is excluding all other external regressors as well (the prediction process cannot be completed without the presence of all current external regressors) resulting in a significant shorter training time. Although the five performance plots may seem different on the first look, the graphics reveal that all training times possess a similar behavior.

Figure 6.1 is showing the training times for the SLOs *Delivery Time* and *Additional Costs*. The duration for training a current model is growing constantly for both. The strong fluctuations at a relatively low level of instances are indicating that the most influencing part is the current workload of the computer itself. That is, the overhead of creating a new connection to the *Forecasting Service* as well as persisting and fetching data to or from the database is the main time waster. The difference in later time points clarifies this, as the stabilization of the training times for the *Delivery Time* is less fluctuating than that of the *Additional Costs*, indicating that the prediction process for the first is outweighing the overhead as opposed to the latter.

In Figure 6.2 the performance measurements for the prediction of the SLOs *Delivery Time* and *Quality* are presented. In this case, the quick training of the *Quality* in earlier execution phases is arising from the condition that categorical variables need at least 10 times more historical instances than the number of their possible classifications. As 15 categories are possible, the *Quality* is not predicted for the first 150 instances and therefore doesn't consume a lot of resources. However, the later fluctuations have a different reason than that of figure 6.1. In this case, a special fallback mechanism of the prediction process is the origin. Every time when a fact or estimate for a current SLO instance is missing the *Forecasting Engine* is ignoring *all* of the additional information. This is because the mathematical model, considering external re-

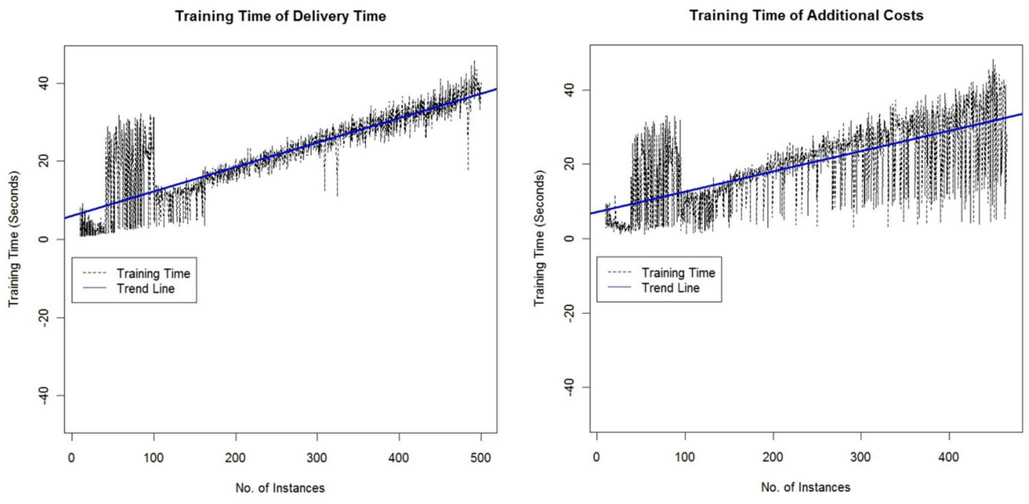

Figure 6.1: Training Times for the SLOs: *Delivery Time* and textitAdditional Costs

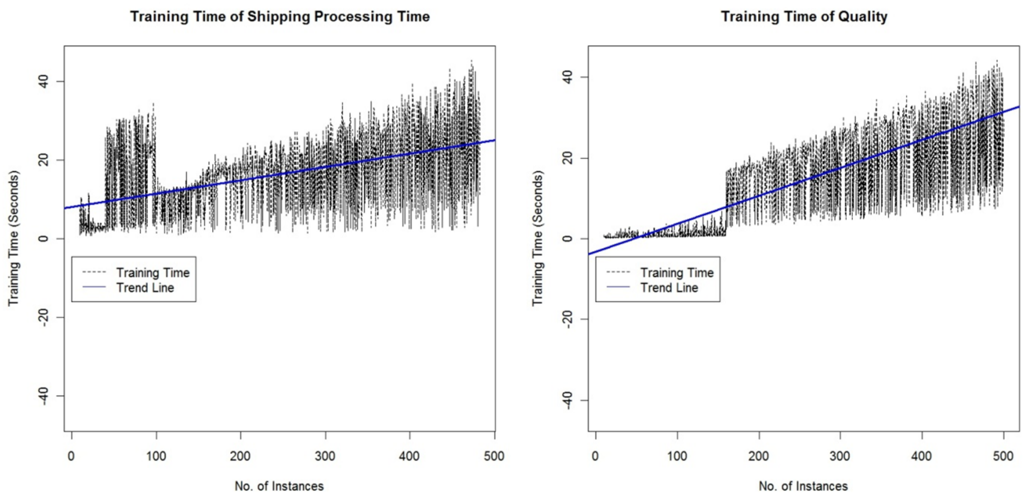

Figure 6.2: Training Times for the SLOs: *Shipping Processing Time* and textitQuality

gressors as well, would fail as a result of missing values. At this point, to prevent failures, the prediction process focuses on the historical SLO values only, which results in a faster prediction time. In the case of the *Quality*, which uses 2 facts and 2 estimates, the statistical model would be reduced from 6 to 2 coefficients that have to be estimated. In the case of an $AR(2)$ model, the amount of information is reduced to 33%, which in turn means a speed up of 67%. The training times for the *Shipping Time* are, like for the *Additional Costs*, relatively small compared to the communication overhead of the *Forecasting Service*.

The last performance plot of figure 6.3, which shows the results for the *Payment Processing Time* is showing the same behavior as the *Additional Costs*. The communication and preparation overhead is much more deciding than the prediction process itself.

The scaling behavior of the *Forecasting Service* is constant with a flat increase as the number of instances is growing. The measured training times therefore show that this method is applicable in real world business cases, where the time component is an important factor.

6.2 Simple Cases

When not dealing with trends or seasonality, the creation of a simulator is relatively simple. After setting a fixed number of AR and MA terms the coefficients are selected randomly. The formulas of Chapter 2.2 are then simply applied to the white noise, which is created, by taking the expected value of several realizations of a random process. Additionally the white noise is normalized into specified boundaries. All required parameters can be applied by using an XML based simulator configuration file.

As described in Chapter 4, the *Payment Processing Time* follows a simple AR process and the *Additional Costs* an ARMA process. Therefore, the simulator configuration for the *Banking Service* is as follows:

Banking Service Configuration	
Minimum Value	200
Maximum Value	500
AR Terms	2
MA Terms	0
Differences	0
Seasonality	0

Table 6.1: *Banking Service* Configuration

As the payment processing time is coupled with the *Billing Service* and *Customer Service* as well, their settings are nearly the same. Only their differing minimum and maximum values are therefore listed in the following:

Billing Service Configuration	
Minimum Value	50
Maximum Value	250

Table 6.2: *Billing Service* Configuration

Customer Service Configuration	
Minimum Value	100
Maximum Value	200

Table 6.3: *Customer Service* Configuration

Additional Costs do not require a special simulator since they are dependent on the quality of a product (detailed in Section 6.4) and the general workload of the reseller.

Figure 6.4 shows the results for *Additional Costs*. The real values for this SLO are ranging from 2770 to 6340 as the green line depicts. The value range of the prediction - shown as red dashed line - is similar with a lowest value of 2896 and a highest value of 6340. The prediction accuracy is quite good, as the standard deviation is beyond 140. A closer look at the plot discloses peaks that are not predicted very precisely. As the model fitting process is done automatically it is not possible to avoid these prediction inaccuracies. However it is possible to reduce the impact by choosing a lower SLA violation boundary under the condition of accepting an occasional false positive alert. The chosen boundary in this case is 1100, resulting in a confusion matrix shown in Table 6.2.

		Predicted	
		Violated	**Not Violated**
Actual	**Violated**	0	3
	Not Violated	1	574

Table 6.4: Confusion Matrix for *Additional Costs*

The prediction accuracy for *Additional Costs* is 99.31% providing a very good ex-ante SLA violation detection. Additionally a Student t-test was performed to confirm the statistical significance. The resulting p-value of 0.7281 is far above the requested value of 0.05 (a 95% confidence interval was chosen) indicating that the Null-Hypothesis H_0 = *The result is statistical significant* should be accepted.

The *Payment Processing Time* depicts an important factor of success when modeling time series. Namely the careful selection of additional regressors. There are 2 ways in which the selection of external regressors can be done wrong:

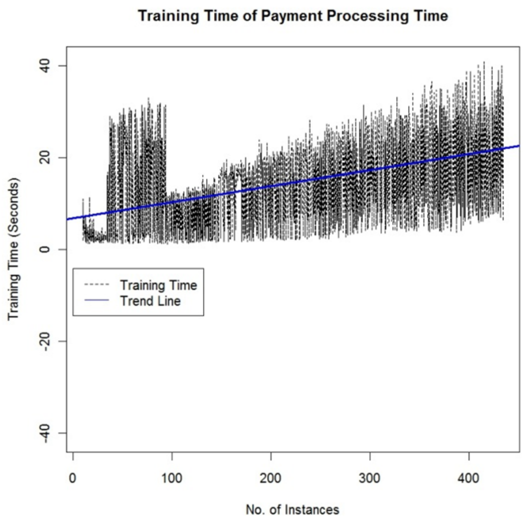

Figure 6.3: Training Times for the SLO: *Payment Processing Time*

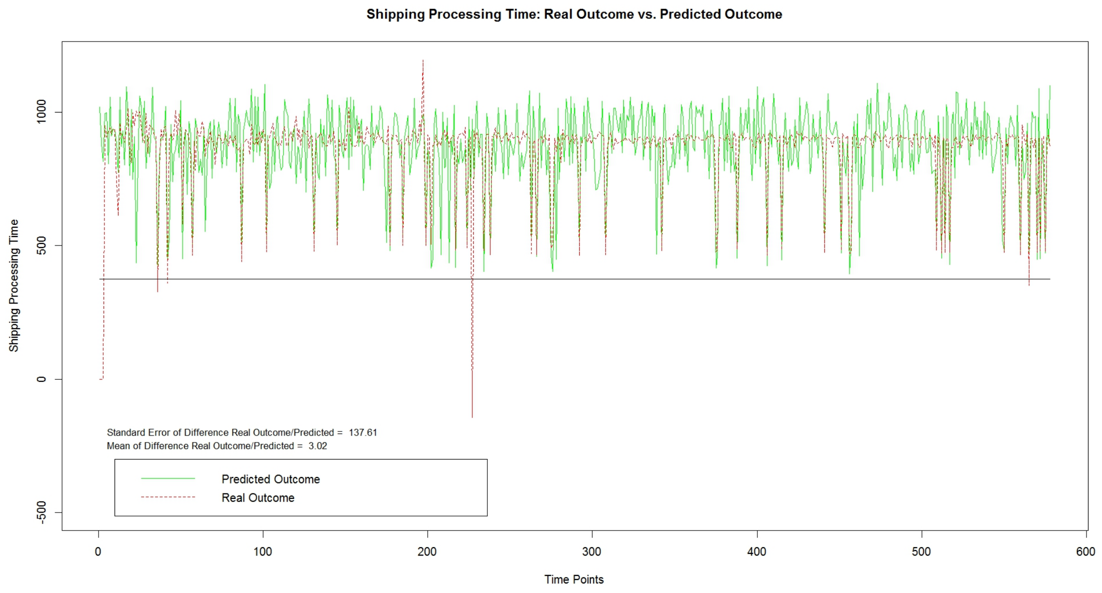

Figure 6.4: Prediction Accuracy for *Additional Costs*

1. Taking *not enough* regressors into consideration
2. Over fitting the model by choosing *to many* additional variables

The first case is shown in figure 6.5, where only the target SLO is considered itself. Here, the information that comes from the time series is not sufficient because it is influenced not only by its own past, but also depends on some unknown additional variables.
With an SLA boundary of 4000 the confusion matrix of the corresponding Figure 6.5 is as depicted in Table 6.2

		Predicted	
		Violated	**Not Violated**
Actual	**Violated**	2	1
	Not Violated	0	435

Table 6.5: Confusion Matrix for the *Payment Processing Time* without External Regressors

Although the prediction accuracy of 99.77% and a p-value of 0.9077 is a sufficient result, the addition of more information can improve the results a little further.

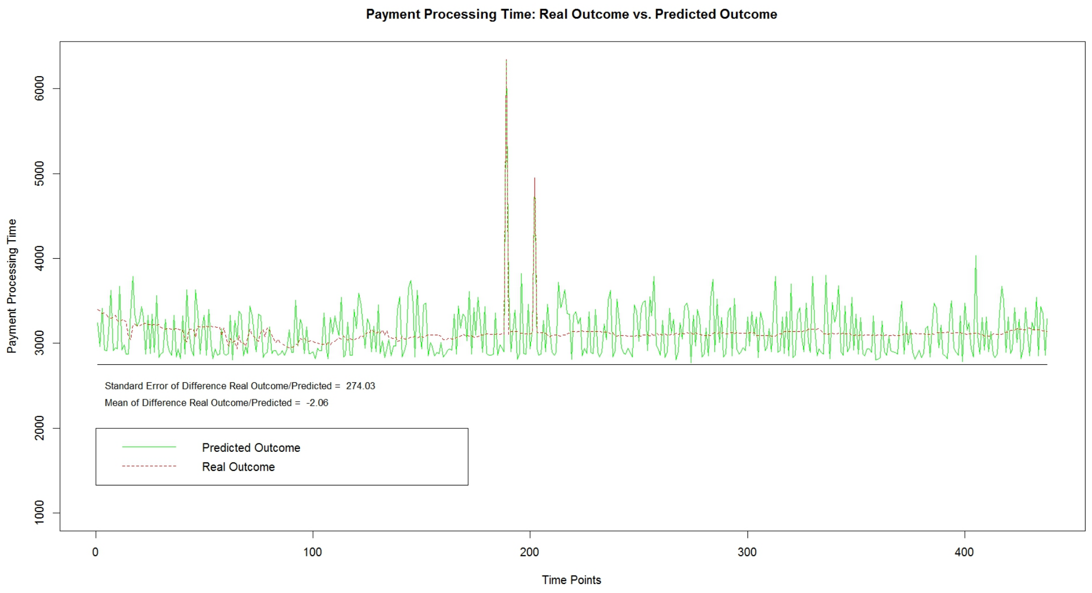

Figure 6.5: Prediction Accuracy for the *Payment Processing Time* without External Regressors

On the other hand, it is almost always worse to use additional, but unnecessary information. In this case, the prediction accuracy not only does not improve, but sometimes leads to a wrongly fitted model, resulting in a much inferior prediction accuracy. As one can see in Figure 6.6, the prediction quality of the *Payment Processing Time* is very bad, which arises from the unnecessary estimates *RESPONSE_TIME_credit_cc* and *RESPONSE_TIME_request_cc_authorization*

of the *Banking Service*.
The confusion matrix of table 6.2 confirms this first impression giving a prediction accuracy of 0.68%. The p-value being lower than $2.2e^{-16}$ is indicating a clear rejection of the null hypothesis $H_0 =$ *The result is statistical significant*.

		Predicted	
		Violated	**Not Violated**
Actual	**Violated**	3	0
	Not Violated	435	0

Table 6.6: Confusion Matrix for the *Payment Processing Time* with Additional Estimates: *RESPONSE_TIME_credit_cc* and *RESPONSE_TIME_request_cc_authorization*

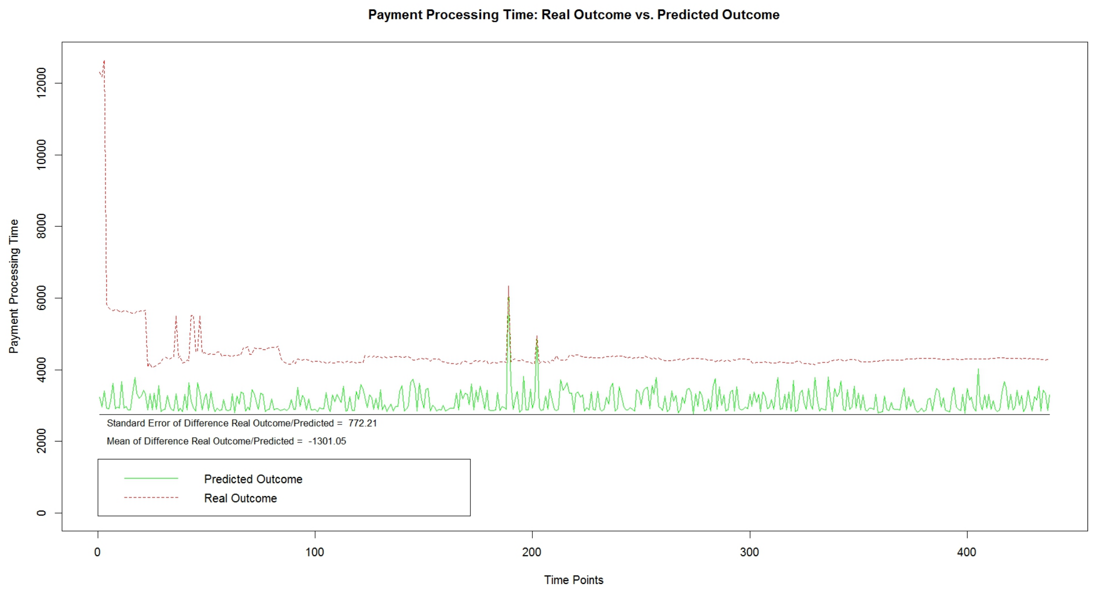

Figure 6.6: Prediction Accuracy for the *Payment Processing Time* with Additional Estimates: *RESPONSE_TIME_credit_cc* and *RESPONSE_TIME_request_cc_authorization*

Finally, the model that was found to be the best, is depicted in Figure 6.7. It does not only consider the *Payment Processing Time* itself, but also the estimates *RESPONSE_TIME_get_cc_details* and *RESPONSE_TIME_get_payment_prefs* from the *Customer Services*. The inclusion of the 2 estimates leads to a standard prediction error of 275.89 which is slightly better than the first, and much better than the second model. The resulting p-value of 0.8066 and a prediction accuracy of 99.77% confirm the statistical significance. The corresponding confusion matrix can be found in Table 6.2.

		Predicted	
		Violated	**Not Violated**
Actual	**Violated**	2	1
	Not Violated	0	435

Table 6.7: Confusion Matrix for the *Payment Processing Time* with Additional Estimates: *RESPONSE_TIME_get_cc_details* and *RESPONSE_TIME_get_payment_prefs*

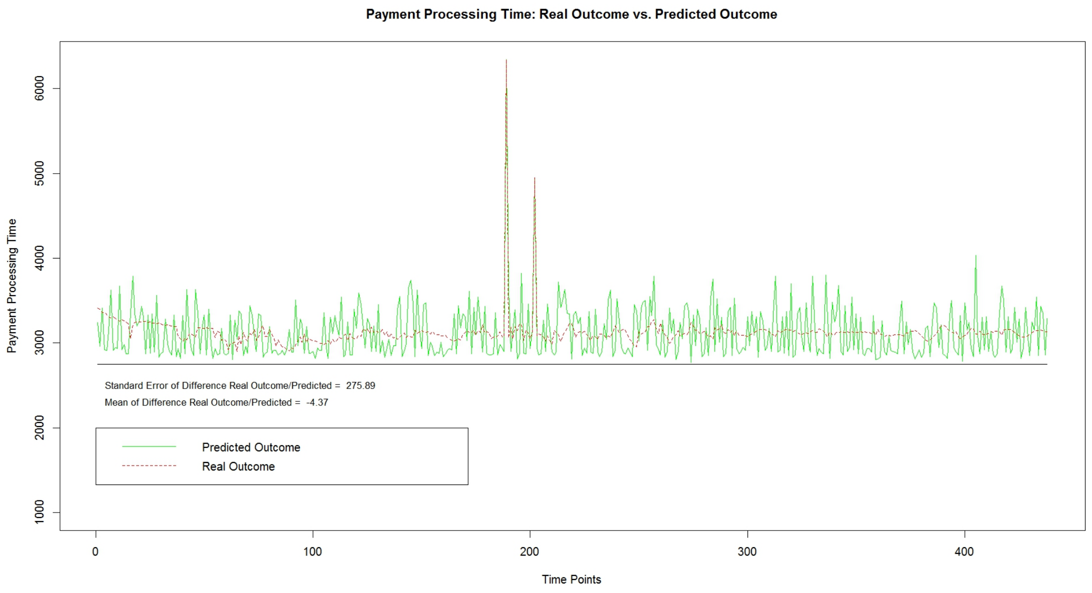

Figure 6.7: Prediction Accuracy for the *Payment Processing Time* with Additional Estimates: *RESPONSE_TIME_get_cc_details* and *RESPONSE_TIME_get_payment_prefs*

6.3 Time Series with Trend

The *Delivery Time* is influenced by all other SLOs. It possess an unknown trend that is estimated during the prediction process.
As figure 6.8 shows the *Delivery Time* possesses a linearly growing trend. That is, the long-term behavior of the process is increasing with time. It is not obvious to see, that the prediction of this time series is quite good. Although the standard deviation of the residuals is above 15.000 one has to consider that it has to be compared to the maximum value of the delivery time (above 150,000). Since SLO values are always compared to predefined boundaries, the violation of an SLA can be often prevented by choosing the boundaries a little bit lower (e.g. if the real SLA violation occurred at a value > 120,000 the boundary will be set to 100,000).

Moreover, the p-value of 0.608 and a prediction accuracy of 99.52% prove that the results are statistical significant. The confusion matrix, showing the correct/incorrect classifications with an SLA boundary of $155,000$, is found in Table 6.3.

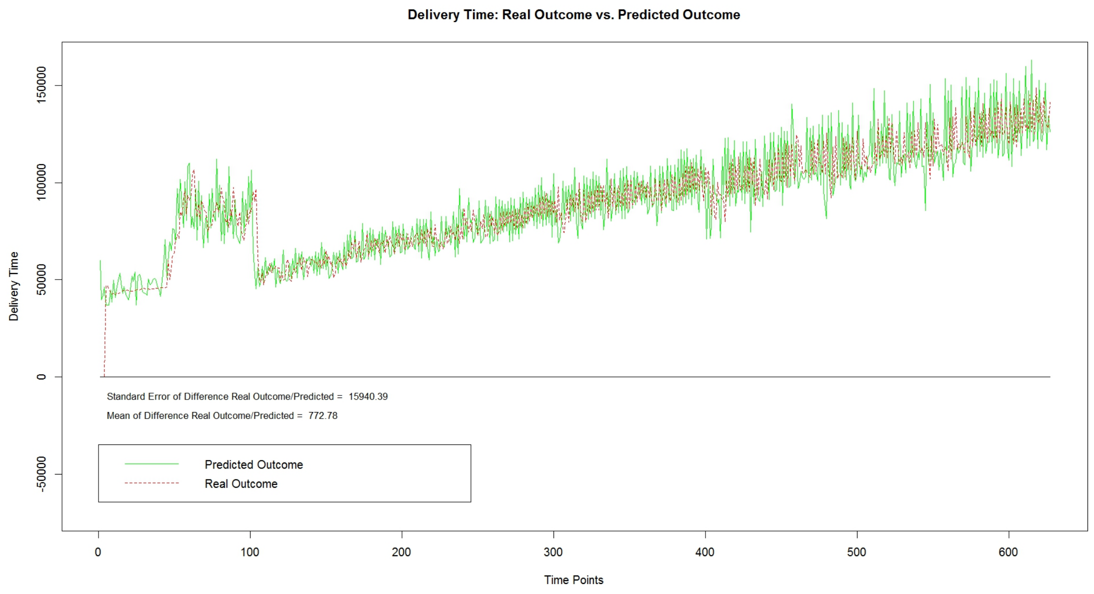

Figure 6.8: Prediction Accuracy for the *Delivery Time*

		Predicted	
		Violated	**Not Violated**
Actual	**Violated**	0	3
	Not Violated	0	624

Table 6.8: Confusion Matrix for the *Delivery Time*

6.4 Saisonality

The trend component was simulated with the help of sinusoidal waves. As described by [60], a cyclic behavior of time series can be described by the sine or cosine function accordingly. As both functions can be used, the simulation technique for the seasonality is - as opposed to [60] - based on the sine function. The equation for describing a seasonal process is as follows:

$$x_t = A * sin(\frac{2\pi}{w} * t + \Phi) \tag{6.1}$$

Where A denotes the amplitude, that is, the boundary in which x_t is oscillating. In the case of the simulation, A is given by the maximum value that a certain series can take, divided through three (to damp the oscillations). W describes the cycle, by specifying the time steps that are necessary to complete one cycle. While t denotes the current time step, Φ is called the phase and determines the starting point of the sine function. Φ is set to zero, since the determination of the exact starting point is not necessary during the simulations.

The *Shipping Processing Time* is expected to follow a seasonal trend which oscillates within a 12 month cycle. Because the main influencing factor is given by the *Shipping Service*, it was configured as follows:

Shipping Service Configuration	
Minimum Value	500
Maximum Value	1000
AR Terms	1
MA Terms	0
Differences	0
Seasonality	12

Table 6.9: *Shipping Service* Configuration

		Predicted	
		Violated	**Not Violated**
Actual	**Violated**	0	3
	Not Violated	0	606

Table 6.10: Confusion Matrix for the *Shipping Processing Time*

Figure 6.9 depicts the prediction process of the *Shipping Processing Time*. One can see, that the series possesses a cycle of 12 (month). The Standard Error of 203 and a prediction accuracy of 99.51% indicate, that the seasonal component of this process is well depicted by the automatically fitted model. With an SLA boundary of $2,400$ the p-value of the t-test is 0.4469 which indicates a clear statistical result. The corresponding confusion matrix is found in Table 6.4

6.5 Categorical Cases

	Instances
Correct	45
Not Correct	582

Table 6.11: Confusion Matrix for the *Quality*

The simulation of categorical time series was done with the help of the R package *dse* [22]. Thereby, the simulator makes use of the *RCommunicator* class of the *TSForecasting* package. A special method *simVAR* takes as arguments the number of dimensions, AR terms and the sample size. The method then creates a new model with the according dimensions, and assigns random

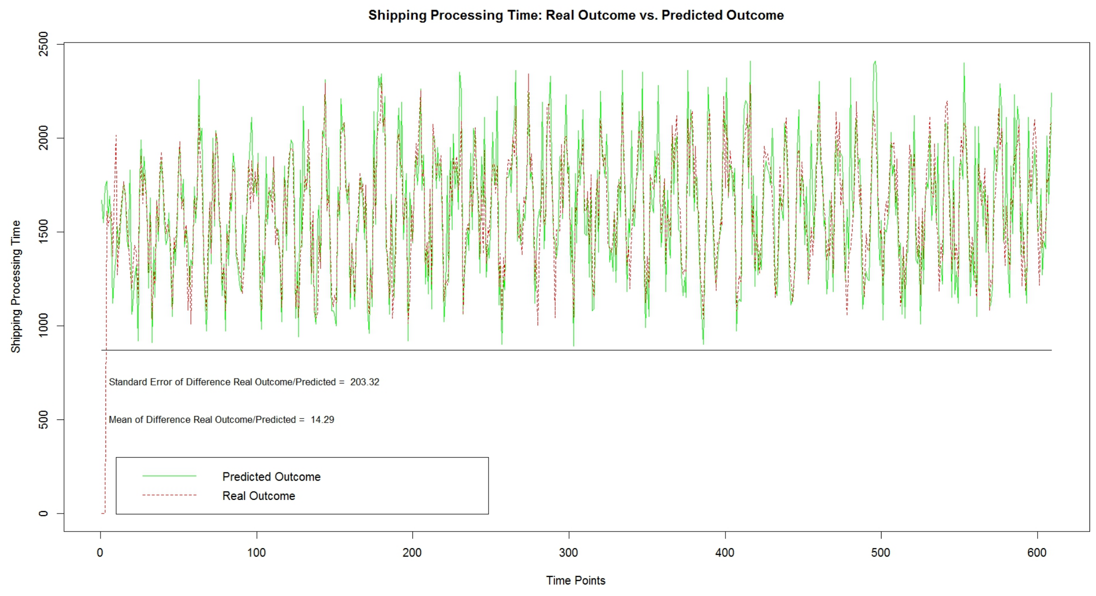

Figure 6.9: Prediction Accuracy for *Shipping Processing Time*

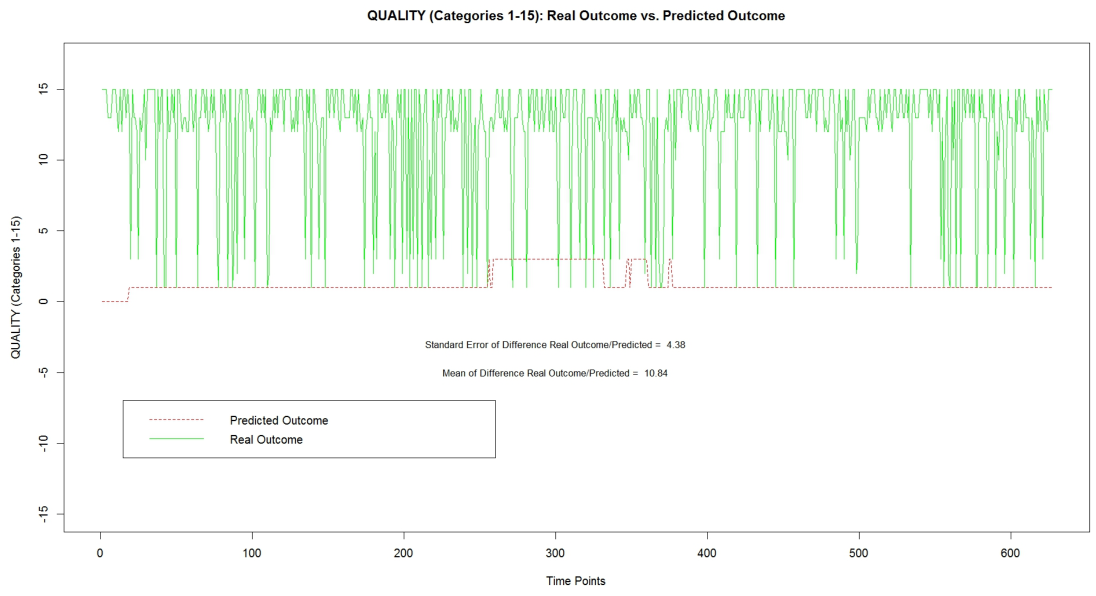

Figure 6.10: Prediction Accuracy for *Quality*

coefficients. When the model is built, the simulation returns a list of the specified size, which contains vectors of dichotomous variables. The resulting vectors can then be translated into categorical variables.
As the *QUALITY* of the product is strongly influenced by the customer's choice of priority, the chosen priorities of the customers are generated by a random $AR(2)$ process with categories reaching from 1 up to 15.

Figure 6.10 shows the prediction process of the *QUALITY*. As categorical variables are converted into dichotomous vectors the AR approach does not perform very well. This is due to the fact, that the AR models depict linear dependencies between current and past values. As the dichotomous vectors contain but zeros and ones, the estimated coefficients do not optimally describe the behavior of the time series. A further t-test with a resulting p-value of lower than $2.2e^{16}$ and a prediction accuracy of only 7.18% confirms the bad prediction performance. As Table 6.5 shows, only 45 out of 627 instances were classified correctly.

In the case of categorical variables, one might use other techniques such as decision trees proposed by [35].

6.6 Results Review

The evaluation of this chapter is showing that the prediction performance of the contributed work is quite accurate. In all numerical cases, the performance results are very satisfying. The calculated prediction accuracy is consistently above 99% (except for the categorical and overfitted model) which indicates that the *Forecasing Service* is of great value when composing services automatically. Furthermore the example of the *Payment Processing Time* shows that the careful selection of facts and estimates, which should be included, is crucial for the prediction performance. The main drawback of the proposed model is the prediction for categorical SLO values. The prediction accuracy of 7.18% leads to the suboptimal number of correct classifications of 45 instances out of 627.
The main message of this chapter is that the contributed work provides an accurate way of predicting SLA violations with SLOs of numerical nature. In the case of categorical values more research work is required. Thereby, statistical methods like HMM or SVM are promising.

CHAPTER 7

Conclusion

7.1 Summary

The prediction and prevention of SLA violations is becoming more and more important within the context of service compositions. Especially when whole business processes are to be performed without human intervention, the automatic building and training of adequate statistical models is fundamental. As current solutions lack of the involvement of historical data - or solely focus on the detection of SLA violations - this thesis aims to close this specific gap. The first part of this work served as introduction and constituted the concrete scientific questions and issues that justify the utilization of statistical models for the prediction of SLA violations. While the chapters 3 and 4 outlined the technical background and related research work, chapter 5 presented an example scenario that emphasized the need for the contribution of this thesis. A concrete methodology and algorithms that address the issues of chapter 1 were established in chapter 6. Thereby, the statistical language *R* was used to implement procedures for the automatic creation of ARIMA models. Based on the example use case chapter 7 revealed the potentials of the contributed work, by showing, that its deployment can possibly predict and, more importantly, prevent many possible SLA violations.

7.2 Future Work

A fundamental shortcoming of the contributed work is the poor performance when dealing with nominal SLO values. As it is out of scope, further research work should be conducted, to explore the prediction quality of other statistical models, such as HMM or SVM. Especially the generalized class of state-space models will serve as an interesting starting point.
As the evaluation of the contributed work was realized by a set of simulators further tests need to be performed. Thereby, comprehensive test within real business cases is desirable as this ensures that the developed algorithms appropriately perform in a constantly changing environment with unexpected side-effects.

APPENDIX A

Diagrams

A.1 Complete Workflow Diagram of the Use Case

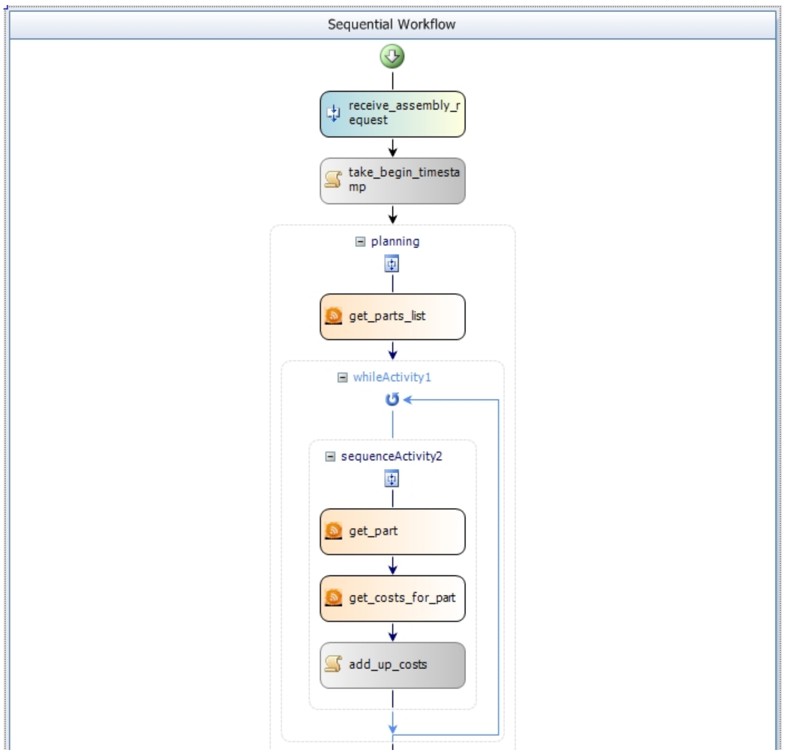

Figure A.1: Complete Workflow Diagram of the Use Case Part1 (taken from [51])

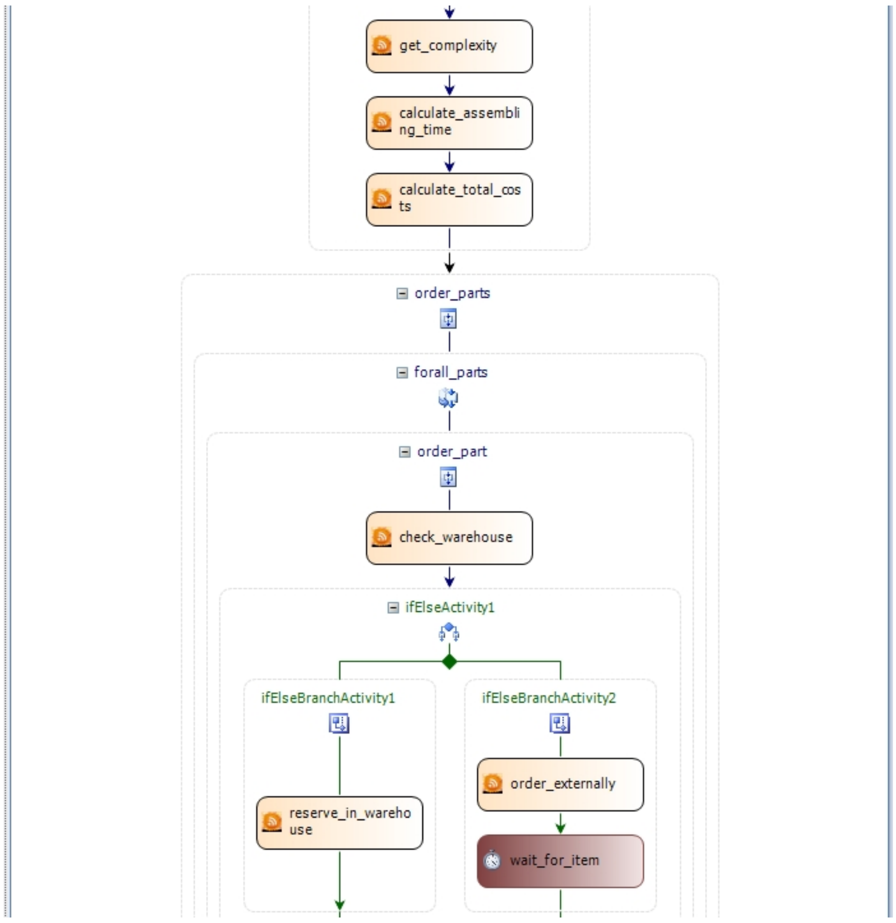

Figure A.2: Complete Workflow Diagram of the Use Case Part2 (taken from [51])

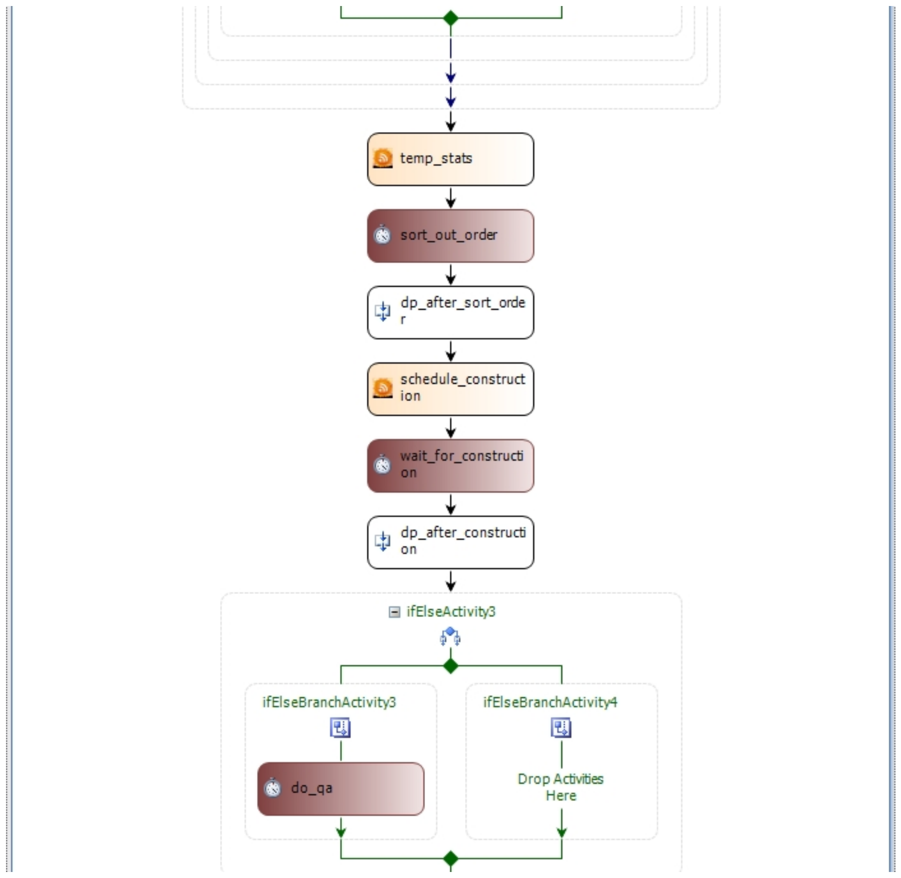

Figure A.3: Complete Workflow Diagram of the Use Case Part3 (taken from [51])

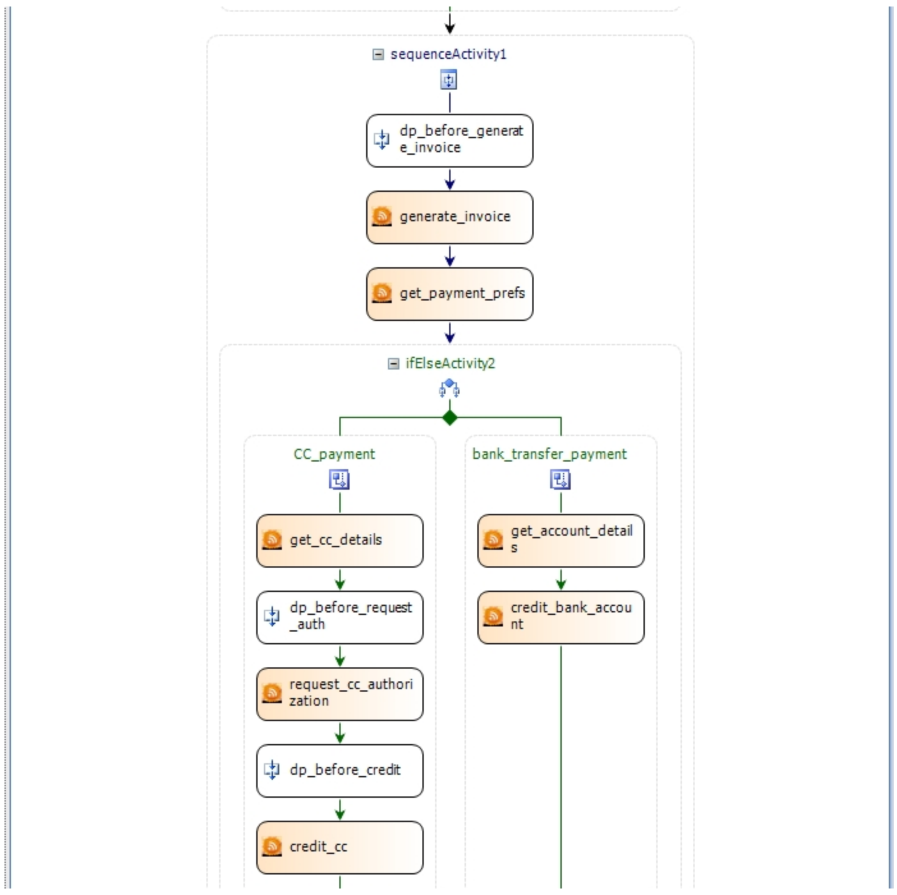

Figure A.4: Complete Workflow Diagram of the Use Case Part4 (taken from [51])

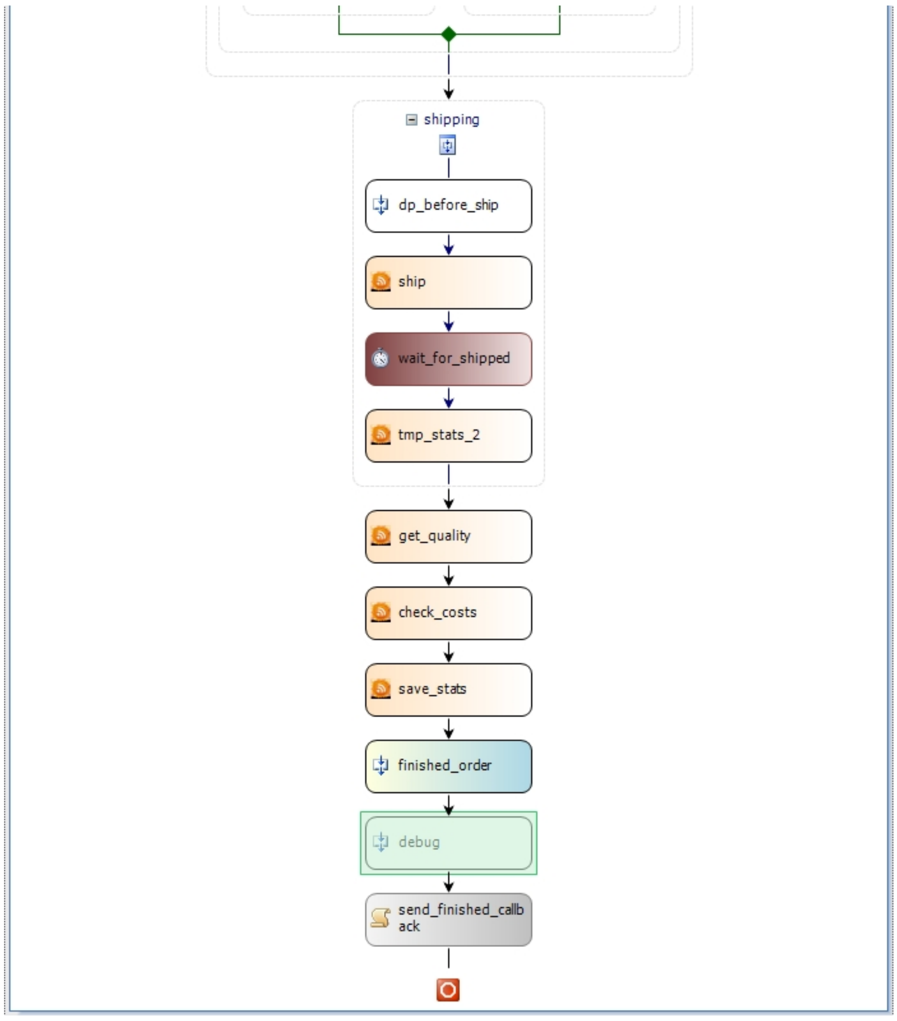

Figure A.5: Complete Workflow Diagram of the Use Case Part5 (taken from [51])

APPENDIX B

Listings

B.1 Listings of Implemented R Code

```
trainTSModel <- function(d, xreg = NULL) {
  model = list()
  model$xreg = NULL
  model$noxreg = NULL
  categorical = FALSE

  #fix the NA values in xreg columns
  if(!is.null(xreg)) {
    # remove columns with only NA values
    xreg <- xreg[,colSums(is.na(xreg))<nrow(xreg)]
    xreg <- na.approx(xreg,rule=2)
  }

  if(is.matrix(d)) {
    #do it with the dse package
    if(!is.null(xreg)) {
      currentModel = NULL
      bestAIC = NULL
      for(i in 1:6) {
        suppressWarnings({
          tryCatch({
            tsd <- TSdata(input=xreg,output=d)
            tmpModel <- estVARXls(tsd, max.lag=i)
            size = length(d[,1])
            tmpAIC <- tmpModel$estimates$like[1] + ((size + i)/(size - i - 2)
                )

            if(is.null(currentModel) || bestAIC > tmpAIC) {
              currentModel <- tmpModel
              bestAIC <- tmpAIC
            }
        }, error = function(e) {
          currentModel <- NULL
        })
      })
    }
    model$xreg <- currentModel
  }

  currentModel <- NULL
  bestAIC <- NULL
  for(i in 1:6) {
  suppressWarnings({
    tryCatch({
      tsd <- TSdata(output=d)
      tmpModel <- estVARXls(tsd,max.lag=i)
      size = length(d[,1])
      tmpAIC <- tmpModel$estimates$like[1] + ((size + i)/(size - i - 2))
      if(is.null(currentModel) || bestAIC > tmpAIC) {
            currentModel <- tmpModel
            bestAIC <- tmpAIC
          }
    }, error = function(e) {
      currentModel <- NULL
    })
  })
  }
  model$noxreg <- currentModel

  categorical = TRUE
} else {
  #do it with the forecast package
  if(!is.null(xreg)) {
    suppressWarnings({
      tryCatch({
        model$xreg <- auto.arima(x=d,xreg=xreg)
      }, error = function(e) {
        model$xreg <- NULL
      })
    })
  }

  suppressWarnings({
    tryCatch({
      model$noxreg <- auto.arima(x=d,xreg=NULL)
    }, error = function(e) {
      model$noxreg <- NULL
    })
  })

  categorical = FALSE
}

if(is.null(model$xreg) && is.null(model$noxreg)) {
```

```r
    stop("No model could be found")
  }

  return(list(model=model, categorical=categorical))
}
```

Listing B.1: R Function to Train a Suitable Model for a concrete Time Series

```r
getTSPrediction <- function(model, f_xreg=NULL) {
  if(model$categorical == TRUE) {
    fc <- NULL
    if(!is.null(f_xreg)) {
      suppressWarnings({
        tryCatch({
          m <- model$model$xreg$model
          newModel <- ARMA(A=m$A,B=m$B,C=m$C,TREND=m$TREND, constants=m$
              constants)
          fc <- l(newModel, model$model$xreg$data)
        }, error = function(e) {
          print("Skipping exogenous input")
          fc <- NULL
        })
      })
    }
    if(is.null(fc)) {
      suppressWarnings({
        tryCatch({
          m = model$model$noxreg$model
          newModel = ARMA(A=m$A,B=m$B,C=m$C,TREND=m$TREND, constants=m$
              constants)
          fc <- l(newModel,model$model$noxreg$data)
        }, error=function(e) {
          fc <- NULL
        })
      })
    }
    if(is.null(fc)) {
      stop("Model could not be predicted")
    }

    #if prediction received, get results
    dimensions = c(length(fc$model$A[,1,1]),0,length(fc$model$B[,1,1]))
    sigma2 = mean(fc$estimates$cov)
    prediction = fc$estimates$pred[(length(fc$estimates$pred[1,])+1),]
    prediction = toResult(prediction)
  } else {
    fc <- NULL
    if(!is.null(f_xreg)) {
      suppressWarnings({
        tryCatch({
          m <- model$model$xreg
          if(is.null(m)) {
            fc <- NULL
            print("Skipping exogenous input")
          } else {
            fc <- forecast.Arima(object=m, xreg=f_xreg)
          }
        }, error = function(e) {
          fc <- NULL
          print("Skipping␣exogenous␣input")
        })
      })
    }

    if(is.null(fc)) {
      suppressWarnings({
        tryCatch({
          m <- model$model$noxreg
          fc <- forecast.Arima(object=m, xreg=NULL)
        }, error=function(e) {
          fc <- NULL
        })
      })
    }

    if(is.null(fc)) {
      stop("Model␣could␣not␣be␣predicted")
    }

    m <- model$model$noxreg
    dimensions <- m$arma[(length(m$arma)-2):length(m$arma)]
    sigma2 <- m$sigma2
    prediction <- fc$mean[1]
  }

  return(list(dimensions=dimensions, sigma2=sigma2, prediction=prediction))
}
```

Listing B.2: R Function to get a Prediction from a fitted Model

```
toResult <-
function(vector) {


vector[vector >= max(vector)] <- 1
vector[vector < max(vector)] <- 0

  if(sum(vector) > 1) {
     for(i in 1:length(vector)) {
if(vector[i] == 1) {
vector[] = 0
vector[i] = 1
return(vector)
}
 }
}

return(vector)
}
```

Listing B.3: R Function to normalize a categorical output vector

```
simVAR <- function(dim, ar, N) {
  mu = 1/(ar+dim)
  AR = matrix(rnorm(dim^2, mean=mu, sd=(mu/3)),dim)

  if(ar > 1) {
    for(i in 2:ar) {
      AR = cbind(AR, matrix(rnorm(dim^2,mean=mu, sd=(mu/3)),dim))
    }
  }

  tmp <- matrix(t(cbind(diag(dim),AR)), ncol = dim, byrow=TRUE)
  m <- ARMA(A=array(tmp, c((ar+1),dim,dim)), B=diag(dim), TREND=rnorm(dim))

  start <- vector(length=dim, mode="numeric")
  start[] = 1

  res <- simulate(m, sampleT=N)

  res <- res$output

  for(i in 1:length(res[,1])) {
    vec <- res[i,]
    vec[vec >= max(vec)] <- 1
    vec[vec < max(vec)] <- 0
    vec[vec >= max(vec)] <- 1
    res[i,] <- vec
  }

  return(res)
}
```

Listing B.4: R Function to Simulate a Multivariate AR Model

Acronyms

ACF Autocorrelation Function 10, 14

AICc Akaike's Information Criterion, Bias Corrected 14, 20, 21, 33, 34, 36, 37

AR Autoregressive 3, 12, 14, 21, 23, 27, 28, 33–36, 41, 46, 53, 55, 69

ARIMA Autoregressive Integrated Moving Average v, vii, ix, 3, 11–14, 20, 21, 28, 29, 33, 39, 41, 44, 57

ARMA Autoregressive Moving Average 12, 13, 23, 46

BIC Bayesian Information Criterion 14

BPEL Business Process Execution Language 16

CBR Case-based Reasoning 19

CLR Common Language Runtime 6

DeSVi Detecting SLA Violation infrastructure 18

DLL Dynamic Link Library 37

DSL Domain Specific Language 19

FoSII Foundations of Self-governing Infrastructures 18

GUID Globally Unique Identifier 37

HMM Hidden Markov Model 22, 55, 57

IIS Internet Information Services 7, 44

IPAT Inter Packet Arrival Time 17, 18

JMS Java Messaging Service 19

KLD	Kullback Leibler Divergence 17
LoM2His	Low Level Metrics to High Level SLA 19
MA	Moving Average 12, 14, 21, 28, 33–36, 46, 53
ML	Maximum Likelihood 11, 12, 14, 37
OLS	Ordinary Least Squares 11, 12
PACF	Partial Autocorrelation Function 14
QoS	Quality of Service 9, 15–17
RMSE	Root Mean Squared Error 21, 23
SIC	Schwarz Information Criterion 14
SLA	Service Level Agreement v, vii, ix, 1–3, 9, 10, 16–19, 21, 26, 29–33, 43, 47, 49, 51, 53, 55, 57
SLO	Service Level Objective v, vii, 2, 3, 9, 10, 18, 19, 21, 26, 28–33, 37–40, 43–49, 51, 55, 57
SOA	Service Oriented Architecture ix, 1, 3, 5, 6, 9, 15
SVM	Support Vector Machine 22, 23, 55, 57
UML	Unified Modelling Language 41
VID	Valid IPAT Distribution 17
VM	Virtual Machine 18
VQL	VRESCo Query Language 15
VRESCo	Vienna Runtime for Service Compositions v, vii, 2, 3, 15, 16, 18, 26, 29, 32, 37, 43
WAS	Windows Activation Service 7
WCF	Windows Communication Foundation ix, 3, 6–9, 25, 37, 40
WSDL	Web Service Description Language 5, 6

Bibliography

[1] Kosei ABE. R.net. `http://rdotnet.codeplex.com/`, 2010-2011.

[2] Philipp Leitner Anton Michlmayr, Florian Rosenbertg and Schahram Dustar. Comprehensive qos monitoring of web services and event-based sla violation detection. *MW4SOC*, 2009.

[3] Christoph Beierle and Gabriele Kern-Isberner. *Methoden wissensbasierter Systeme. Grundlagen – Algorithmen – Anwendungen.* Vieweg, Wiesbaden, 3 edition, 2006.

[4] Martin G. Bernhard. *Praxishandbuch Service-Level-Management: Die IT als Dienstleistung organisieren.* Symposium Publishing GmbH, 2006.

[5] George E. P. Box and Gwilym M. Jenkins. *Time Series Analysis - Forecasting and Control.* Holden-Day, 1976.

[6] Fabio Canova and Bruce E. Hansen. Are seasonal patterns constant over time? a test for seasonal stability. *Journal of Business & Economic Statistics*, 13, 1995.

[7] David Chappell. Introducing windows communication foundation in .net framework 4. `http://msdn.microsoft.com/library/ee958158.aspx`, January 2010.

[8] Tony Chau, Vinod Muthusamy, Hans-Arno Jacobsen, Elena Litani, Allen Chan, and Phil Coulthard. Automating sla modeling. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, CASCON '08, pages 10:126–10:143, New York, NY, USA, 2008. ACM.

[9] Hugo Haas et al. David Booth. Web services architecture. `http://www.w3.org/TR/ws-arch/`, February 2004.

[10] Peter Schmidt Dennis Kwiatkowski, Peter C.B. Phillips and Yongcheol Shin. Testing the null hypothesis of stationarity against the alternative of a unit root. *Journal of Econometrics*, 54:159 – 178, 1992.

[11] Alexis Dimitriadis. Modeling time series with auto-regressive markov models. Master Thesis at the Department of Mathematics, State University of Portland, August 1992.

[12] Norman R. Draper and Harry Smith. *Applied Regression Analysis (Wiley Series in Probability and Statistics).* Wiley-Interscience, third edition, April 1998.

[13] Vincent C. Emeakaroha, Ivona Brandic, Michael Maurer, and Schahram Dustdar. 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*, pages 48–54, 2010.

[14] Vincent C. Emeakaroha, Marco A. S. Netto, Rodrigo N. Calheiros, Ivona Brandic, Rajkumar Buyya, and César A. F. De Rose. Towards autonomic detection of SLA violations in cloud infrastructures. *Future Generation Computer Systems*, November 2011.

[15] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[16] Heiko Ludwik et. al. Web service level agreement (wsla) language specification. `http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf`, January 2003.

[17] Henry S. et. al. (editors). Xml schema part 1: Structures second edition. `http://www.w3.org/TR/2004/REC-xmlschema-1-20041028`, 10 2004.

[18] R. Fielding et. al. (editors). Hypertext transfer protocol – http/1.1. `http://www.ietf.org/rfc/rfc2616.txt`, June 1999.

[19] Tim Bray et. al. (editors). Extensible markup language (xml) 1.0. `http://www.w3.org/TR/REC-xml/`, November 2008.

[20] NHibernate Forge. Nhibernate. `http://nhforge.org/Default.aspx`, 2007-2011.

[21] Berger T. G. *Thomas G. Berger: Service Level Agreements*. VDM Verlag, 2007.

[22] Gilbert and P. D. *Brief User's Guide: Dynamic Systems Estimation*, 2006 or later.

[23] GNU. Gnu general public license, version 2. `http://www.gnu.org/licenses/gpl-2.0.html`, June 1991.

[24] Object Management Group. Unified modelling language. `http://www.uml.org/`.

[25] D Hajjar and S AbouRizk. *SIMPHONY: AN ENVIRONMENT FOR BUILDING SPECIAL PURPOSE CONSTRUCTION SIMULATION TOOLS*, volume 2, pages 998–1006. IEEE, 1999.

[26] James Douglas Hamilton. *Time Series Analysis*. Princeton University Press, 1994.

[27] David J. Hand. Forecasting with exponential smoothing: The state space approach by rob j. hyndman, anne b. koehler, j. keith ord, ralph d. snyder. *International Statistical Review*, 77(2):315–316, 2009.

[28] Sayed Hashimi. Service-oriented architecture explained. `http://oatv.com/pub/a/dotnet/2003/08/18/soa_explained.html`, August 2003.

[29] Rob J Hyndman. *forecast: Forecasting functions for time series*, 2011. R package version 3.08.

[30] Rob J. Hyndman and Yeasmin Khandakar. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software*, 27(3):1–22, 7 2008.

[31] Schahram Dustar Ivar Jorstad and Do Van Thanh. A service oriented architecture framework for collaborative services. *Proc*, 14, 2005.

[32] Nicolai M. Josuttis. *SOA in Practice: The Art of Distributed System Design (Theory in Practice)*. O'Reilly Media, 1 edition, August 2007.

[33] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[34] D.D. Lamanna, J. Skene, and W. Emmerich. Slang: a language for defining service level agreements. In *Distributed Computing Systems, 2003. FTDCS 2003. Proceedings. The Ninth IEEE Workshop on Future Trends of*, pages 100 – 106, may 2003.

[35] Philipp Leitner, Branimir Wetzstein, Florian Rosenberg, Anton Michlmayr, Schahram Dustdar, and Frank Leymann. Runtime prediction of service level agreement violations for composite services. In *Proceedings of the 2009 international conference on Service-oriented computing*, ICSOC/ServiceWave'09, pages 176–186, Berlin, Heidelberg, 2009. Springer-Verlag.

[36] Juval Lowy. *Programming WCF Services*. O'Reilly Media, Inc., 2007.

[37] Y. Lu and S. M. Abourizk. Automated Box-Jenkins forecasting modelling. *Automation in Construction*, 18(5):547–558, August 2009.

[38] Claus von Riegen Luc Clement, Andrew Hately and Tony Roges (editors). Uddi version 3.0. Technical report, OASIS, 2004.

[39] Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter Brown, and Rebekah Metz. Reference model for service oriented architecture 1.0. Technical Report wd-soa-rm-cd1, OASIS, February 2006.

[40] Qusay H. Mahmoud. Getting started with the java message service (jms). `http://java.sun.com/developer/technicalArticles/Ecommerce/jms/`, November 2004.

[41] E. Marilly, O. Martinot, S. Betge-Brezetz, and G. Delegue. Requirements for service level agreement management. pages 57–62, 2002.

[42] Eric A. Marks and Michael Bell. *Service-Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology*. John Wiley & Sons, June 2006.

[43] MathWorks. Matlab. `http://www.mathworks.de/products/matlab/index.html`, 2010.

[44] Microsoft. Internet information services. http://www.iis.net/.

[45] Microsoft. Microsoft .net framework. http://www.microsoft.com/net.

[46] Microsoft. The common language runtime and framework class library. http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=16272, November 2005.

[47] Douglas C. Montgomery, Elizabeth A. Peck, and Geoffrey G. Vining. *Introduction to Linear Regression Analysis (4th ed.).* Wiley & Sons, Hoboken, July 2006.

[48] A. Mosincat and W. Binder. Automated performance maintenance for service compositions. In *Proc. 11th IEEE Int Web Systems Evolution (WSE) Symp*, pages 131–140, 2009.

[49] John Neter, Michael H. Kutner, William Wasserman, and Christopher J. Nachtsheim. *Applied Linear Regression Models.* McGraw-Hill/Irwin, January 1996.

[50] Chair of Statistics University of Würzburg. A first course on time series analysis. http://www.statistik-mathematik.uni-wuerzburg.de/fileadmin/10040800/user_upload/time_series/the_book/2011-March-01-times.pdf, 2011.

[51] Technical University of Vienna. Vresco - vienna runtime environment for service compositions. http://www.infosys.tuwien.ac.at/prototypes/VRESCo, 2006 - 2011.

[52] Mike P. Papazoglou and Willem-Jan Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16:389–415, July 2007.

[53] R. Perrey and M. Lycett. Service-oriented architecture. pages 116–119, 2003.

[54] Waldemar Hummer Philipp Leitner and Schahram Dustar. Cost-based optimization of service compositions. Technical report, Technical University of Vienna, 2011.

[55] Helmuth Pruscha and Axel Göttlein. Forecasting of categorical time series using a regression model. *Economic Quality Control*, 18(2):223 – 240, 2003.

[56] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.

[57] Stefan Rüping. Svm kernels for time series analysis. http://www.stefan-rueping.de/publications/rueping-2001-a.pdf, 2001.

[58] R. M. Sakia. The box-cox transformation technique: a review. *The Statistician*, 41:169 – 178, 1992.

[59] R. Serral-Gracia, Y. Labit, J. Domingo-Pascual, and P. Owezarski. Towards an efficient service level agreement assessment. In *Proc. IEEE INFOCOM 2009*, pages 2581–2585, 2009.

[60] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications*. Springer, 2010.

[61] James Skene, D. Davide Lamanna, and Wolfgang Emmerich. Precise service level agreements. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 179–188, Washington, DC, USA, 2004. IEEE Computer Society.

[62] Justin Smith. *Inside microsoft windows&#174; communication foundation*. Microsoft Press, Redmond, WA, USA, first edition, 2007.

[63] Wilkes L. Sprott D. Understanding service-oriented architecture. `http://msdn.microsoft.com/en-us/library/aa480021.aspx`, January 2004.

[64] Jos J. M. Trienekens, Jacques J. Bouman, and Mark Van Der Zwan. Specification of service level agreements: Problems, principles and practices. *Software Quality Control*, 12:43–57, March 2004.

[65] W3C. Web services description language (wsdl) 1.1. `http://www.w3.org/TR/wsdl`.

[66] W3C. Soap version 1.2. `http://www.w3.org/TR/soap12-part1/`,`http://www.w3.org/TR/soap12-part2/`,`http://www.w3.org/TR/soap12-part3/`, July 2007.

[67] William W. S. Wei. *TIme series analysis: univariate and multivariate methods*. Pearson Addison Wesley, 2006.

[68] Weka Machine Learning Project. Weka. `http://www.cs.waikato.ac.nz/~ml/weka`.

[69] Jun Yan, Ryszard Kowalczyk, Jian Lin, Mohan B. Chhetri, Suk Keong Goh, and Jianying Zhang. Autonomous service level agreement negotiation for service composition provision. *Future Generation Computer Systems*, 23(6):748 – 759, 2007.