Otto-von-Guericke-University Magdeburg

Faculty of Computer Science
Department of Technical and Operational Information Systems

Internship Report

Multivariate Data Analysis of Pharmaceutical Spectrometric and Process Data

Author:
Christian Moewes

January 22, 2007

Supervisors:

Prof. Dr. Eyke Hüllermeier
Universität Magdeburg
Fakultät für Informatik
Postfach 4120, 39016 Magdeburg
Germany

Dipl.-Inf. Jürgen Beringer
Universität Magdeburg
Fakultät für Informatik
Universitästplatz 2, 39106 Magdeburg
Germany

Moewes, Christian:
Multivariate Data Analysis of Pharmaceutical Spectrometric and Process Data
Studienarbeit, Otto-von-Guericke-Universität
Magdeburg, 2007.

Abstract

This report is related to my project which I was working on during my internship with the Intelligent Vision and Reasoning Department at Siemens Corporate Research (SCR) in Princeton, NJ from March 6th until September 6th, 2006.

Real world systems like the manufacturing of of pharmaceutical drugs, semiconductors, petroleum chemicals or metallurgy are much too complex to be understood down to the ground. That is why experiments with batteries of sensors are a necessity. Those detectors (spectrometer, thermometer, barometer, ...) are used for any kind of physical measuring value like spectra, temperature, pressure etc.

Nowadays, the modern measurement technique makes it possible to collect a huge amount of data over time for every sensor. The need of model that is as easy as possible describing the behavior of this multivariate system is big. Nearly every theoretical approach to model a complex system has its limits. The idea of an empirical model is to observe intrinsic, latent variables in an indirect way.

Every measurement contains defects like noise, uncertainties, vagueness. With the help of statistics one can induce knowledge from noisy data. So it is natural to use statistical methods to tackle noisy real-world applications. Multivariate models like PCA or PLS consider the joint effect of all measured variables that maybe reveals unrecognized features. They are also more resistant to noise than classical multivariate regression methods.

My task was to extend the software CAP (System for Condition Assessment & Prognosis) in order to handle multivariate pharmaceutical spectral and process data which were ascertained by sensor measurement of bioreactors. I will give an insight into the software architecture of CAP and describe the utilised methods for multivariate data analysis (MVDA). Moreover I will outline my contributions for CAP which cover graphical user interfaces, hardware interfaces and data mining algorithms as well.

Zusammenfassung

Diese Arbeit bezieht sich auf mein Projekt, an dem ich whrend meines Praktikums mit der Intelligent Vision and Reasoning Abteilung von Siemens Corporate Research (SCR) in Princeton, NJ vom 6. März bis 6. September 2006 gearbeitet habe.

Reale Systeme wie z.B. die Erzeugung von Pharmazeutika, Halbleitern, Petroleumchemikalien oder die Metallverarbeitung sind zu komplex um hundertprozentig verstanden zu werden. Deswegen sind Experimente mit unzähligen Sensoren eine Notwendigkeit. Diese Detektoren (Spektrometer, Thermometer, Barometer, ...) werden für jede Art von physikalischer Messgröße wie Spektren, Temperatur, Druck etc. eingesetzt.

Heutzutage macht die moderne Messtechnik es möglich, eine riesige Menge an Daten über die Zeit für jeden Sensor zu sammeln. Der Bedarf eines Models, dass so einfach wie möglich das Verhalten dieses multivariaten Systems beschreibt, ist groš̈s. Fast jeder theoretische Ansatz, ein komplexes System zu modellieren, hat seine Grenzen. Die Idee eines empirischen Models ist, innere, verborgene Eigenschaften auf indirekter Art und Weise aufzudecken.

Jede Messung ist behaftet mit Fehlern wie Rauschen, Unsicherheit, Unschärfe. Mithilfe von Statistik kann man Wissen von verrauschten Daten ziehen. Also ist es natürlich, statistische Verfahren zu nutzen um reale Anwendungen in Angriff zu nehmen. Multivariate Modelle wie PCA oder PLS berücksichtigen den Effekt des Zusammenschlusses aller Variablen, der vielleicht unerwartete Eigenschaften aufdeckt. Sie sind auch widerstandsfähiger gegen Rauschen als klassische multivariate Regressionsmethoden.

Meine Aufgabe bestand darin, die Software CAP (System for Condition Assessment & Prognosis) zu erweitern, um multivariate pharmazeutische Spektral- und Prozessdaten zu verarbeiten, die durch Sensormessung an Bioreaktoren erhoben wurden. Ich werden einen kurzen Einblick in die Software-Architektur geben und die benutzten Methoden zur multivariaten Datenanalyse (MVDA) beschreiben. Ferner werde ich meine Beitrge für CAP darlegen, die graphische Nutzerschnittstellen, Hardware-Schnittstellen und Algorithmen zur intelligente Datenanalyse überspannen.

Acknowledgement

I would like to thank my supervisor Dr. Claus Neubauer from SCR who made my internship possible at all. I would also like to thank my parents, Günther and Carola, for encouragement, support and help during my stay in the US. Without them, it would have been very hard for me to be there where I am. Special thanks are extended to my "coffee break group" and "ChiFaCC", in particular Diran Guiliguian, Susann Bucher, Carlos Melo Jr, Vassilis Athitsos, Lauren Spingler, Markus Gipp and Ying Li. I would also like to thank Jürgen Beringer and Prof. Eyke Hüllermeier who read this report prior to publication and were kind enough to offer their comments. My grateful thanks go to all the people at the Data Analysis and Modeling Group who had always time to answer my questions in design, implementation and mathematics, particularly Bernhard Glohmann, Yuan Chao, Sheikh Abdul-Saboor, Christoph Lapp-Emden, Marc Holger Uhlmann and Rong-En Fan. Finally, I want to thank Pavla Adamcová who encouraged me to complete my internship and became my best friend.

”Make everything as simple as possible – but not simpler.”

Albert Einstein

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

CAP	System for Condition Assessment and Prognosis
DLL	Dynamic Linked Library
EVD	Eigenvalue Decomposition
FReND	Framework for Relational Numerical Data
GUI	Graphical User Interface
IVR	Intelligent Vision and Reasoning Department at SCR
KDD	Knowledge Discovery in Databases
MSC	Multiplicative Scatter Correction
MVDA	Multivariate Data Analysis
NIR	Near Infrared
OLE	Object Linking and Embedding
OPC	OLE for Process Control
PAT	Process Analytical Technology
PCA	Principal Component Analysis
PLS	Partial Least Squares Projection to Latent Structures
SCR	Siemens Corporate Research
SVD	Singular Value Decomposition
UML	Unified Markup Language
USP	Umetrics SIMCA Project File
XML	Extended Markup Language

Chapter 1

Introduction

This chapter begins with an explanation of the motivation of my project. After this an overview on the problem of MVDA is given. In Section 1.2 I will introduce basic facts about Siemens and SCR in particular as well. In Section 1.3 the system framework is described briefly. Afterwards the task is defined in Section 1.4. The chapter closes with an outline of the remaining chapters.

1.1 Motivation

Nowadays, the amount of data that is collected during measuring a real process is huge. Decades ago, the aggregation of data was poor compared to today. The development of the costs and the qualities of different sensors reached a point where data analysis tools are essential.

Especially in the field of Chemometrics which was defined by Wold [20] we can find such processes where one tries to find the underlying physical laws that describe the behavior of the system. Usually, real-world systems are too complex to be fully understood. Therefore, it is necessary to run different experiments to collect data. This leads to statistical empirical models which are multivariate like PCA or PLS. These approaches have some advantages. They maybe reveal unexpected patterns and take the joint effect of all variables into account as well. Furthermore, the usage of statistical methods is robust against the presence of noise, non-linearities and uncertainties.

Multivariate data analysis (MVDA) can especially be found in pharmaceutical industries. Since data exists in abundance, the main goal is to make sense of huge data sets. In research and development, one tries to find useful information in the data by designing experiments and using MVDA methods to gain this precious knowledge.

On the other hand, pharmaceutical manufacturers can demonstrate understanding of their processes with MVDA. If they do have more knowledge about their processes, the possibility of making a bad product will be low. Very often regulatory barriers have inhibited adoption of state-of-the-art manufacturing practices within the pharmaceutical industry.

Hence the United States Food and Drug Administration (FDA) developed a new risk based approach for Current Good Manufacturing Practices (cGMP). It is called Process Analytical Technology (PAT) and will modernize the regulation of pharmaceutical manufacturing. Pharmaceutical industry can also implement improvements based on their process knowledge without the need for regulatory review.

Thus the implementation of clever experimental setups and the iterative optimization of process parameters will improve the product quality, lower the costs and update the knowledge about the process itself. MVDA methods can help to reach this goal in a very effective way.

The Data Analysis and Modeling Group of Dr. Claus Neubauer (claus.neubauer@siemens.com) at Siemens Corporate Research in Princeton, NJ implemented a framework called CAP that is supposed to serve online monitoring of machines to predict outages. This framework should be extended to solve PAT problems including batch analysis for pharmaceutical data.

Since the number of variables of spectra is huge, projection methods like principal component analysis and partial least squares as multivariate regression are needed to find relevant variables. Both methods are state-of-the-art techniques which usually perform very well.

Before exposing the used framework of CAP and the task formulation, I will mention some facts about Siemens globally. After that I will lead over from Siemens CT to SCR where I was working for six months.

1.2 Siemens Corporate Research

In 2005, Siemens was the world's 21st largest company with a net income of \$2.8 billion and sales of \$98.2 billion. The global budget for research and development (R&D) amounted \$6.6 billion and was partly dedicated to 47,000 R&D employees. Siemens basically covers 6 industrial sectors as shown in Figure 1.1:

- Information and Communications
- Automation and Control
- Power
- Transportation
- Medical
- Lighting

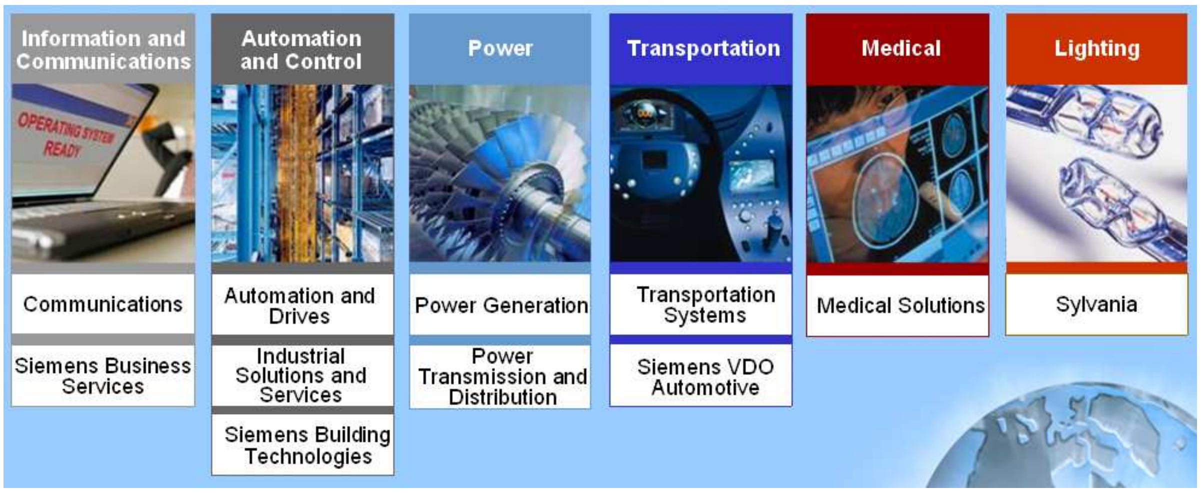


Figure 1.1: The six basic industrial sectors of Siemens at a glance.

Siemens is the world's third highest contributor to Research and Development (R&D). Three quarters of Siemens' revenue is derived from products and services that are less than five years old. Globally, Siemens has more than 30,000 software engineers which means more than SAP, Oracle or Microsoft. In total, Siemens dedicates around 6,500 employees to U.S. R&D daily. Siemens is among the top ten U.S. patents holders. Worldwide, Siemens filled and received nearly 5,000 patent applications in 2003. It currently holds 45,000 active patents.

Siemens Corporate Technology (CT) is a part of Siemens that tries to secure the technological future. Furthermore, Siemens CT wants to increase the competitiveness of the company in close cooperation with the business group and regional units. It serves as a global network as shown in Figure 1.2 in Bangalore, Tokyo, Shanghai, Beijing, Moscow, St. Petersburg, Berlin, Munich, Erlangen, Romsey, Berkley and Princeton. All in all 2,300 researchers and developers are employed at Siemens CT.

Siemens Corporate Research (SCR) was established in the United States in 1977. It is one out of five Corporate Technology R&D centers worldwide. SCR consists of 8 departments as shown in Figure 1.3. Every department runs several programs. I was employed by the department for Intelligent Vision and Reasoning (IVR) which basically pursues the following programs:

- Knowledge Based Image Analysis
  - Robust Image Analysis based on prior knowledge
  - Computer-aided detection and diagnosis
  - Smart Image Acquisition and Composition
- Adaptive Techniques
  - Adaptive Filtering for Image Enhancement

Figure 1.2: Siemens Corporate Technology centers in the world.

  - Geometric Computing and Modeling
  - Intelligent learning, Feature Analysis, Optimization

- Bio-Informatics
  - Biomedical Informatics and Data Mining

- Data Analysis and Modeling
  - Predictive Maintenance
  - Machine Condition Monitoring
  - Medical and Industrial Decision Support

The responsible program for CAP where I worked on is Machine Condition Monitoring. Fundamentally, all technology departments of SCR shall predevelop their programs to productization and salable solutions. The research at SCR shall be initiating for new competencies.

In the next section I will outline the system framework of CAP. Some fundamental overview is needed to understand the concept of the software I was working on.

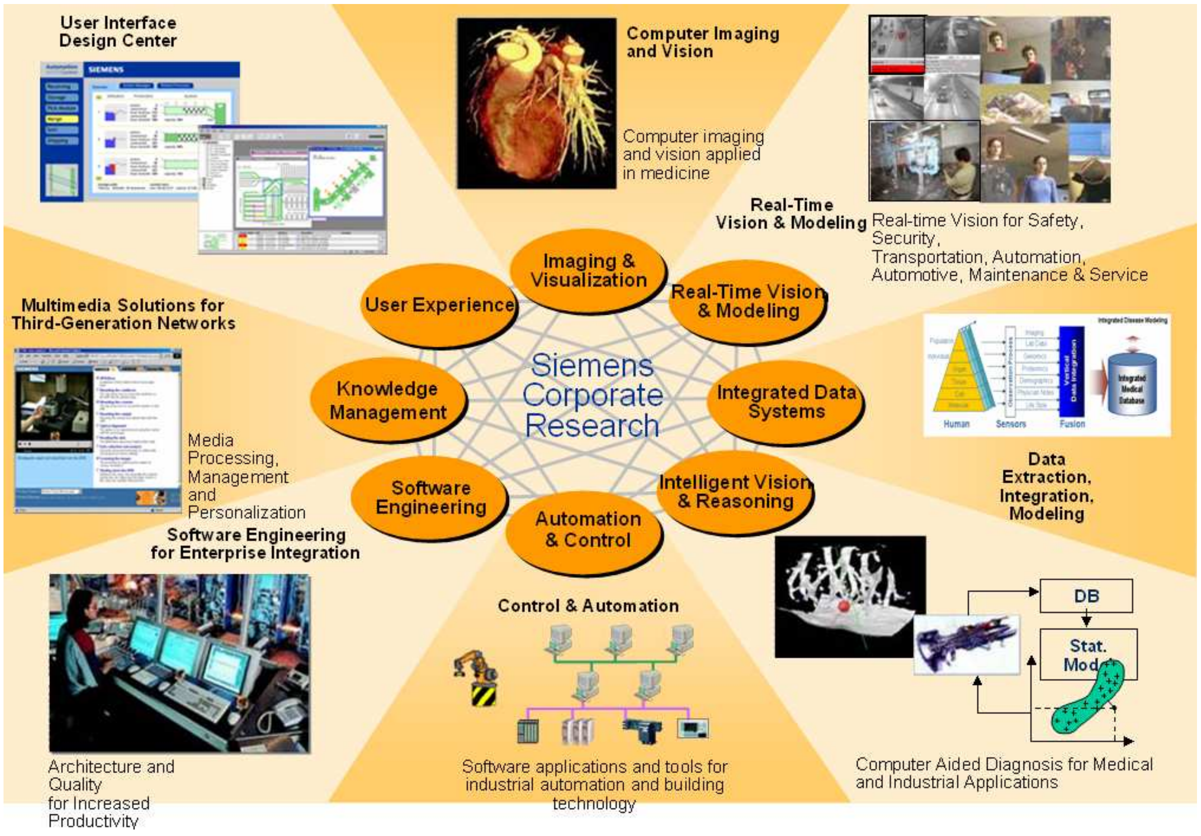

User Interface Design Center
Computer Imaging and Vision
Computer imaging and vision applied in medicine
Real-Time Vision & Modeling
Real-time Vision for Safety, Security, Transportation, Automation, Automotive, Maintenance & Service
Multimedia Solutions for Third-Generation Networks
Media Processing, Management and Personalization
Imaging & Visualization
User Experience
Real-Time Vision & Modeling
Knowledge Management
Siemens Corporate Research
Integrated Data Systems
Software Engineering
Intelligent Vision & Reasoning
Automation & Control
Data Extraction, Integration, Modeling
Software Engineering for Enterprise Integration
Control & Automation
DB
Stat. Mod
Architecture and Quality for Increased Productivity
Software applications and tools for industrial automation and building technology
Computer Aided Diagnosis for Medical and Industrial Applications

Figure 1.3: The eight departments of SCR.

1.3 System Framework

Basically, CAP consists of different Java projects which serve as modules. Thus one can easily add or remove single modules. The design goal was to separate all program logics from the GUI. The following list of projects contains on the one hand relevant modules for PAT and on the other hand projects where most of my contributions can be found:

Algo	contains all classes for the different algorithms and models that are used
Data	handles the user management
FReND	offers methods to model abstract data structures
GUI	contains the GUI for CAP
OPUS	includes a wrapper JNI DLL as interface for OPUS.dll
PATClient	represents the GUI implementation for PAT
Plotting	offers graphical routines for plotting
SIMCA	implements methods of a MVDA tool of Umetrics

I will explain more in detail the different structures of CAP in chapter 3. In the introduction, I will just give a brief description about the single projects.

The Java project ***Algo*** contains all Java classes that represent the concept of a model with its specific algorithms. A model shall describe given data by a certain mathematical abstraction. This approach makes it easy to predict the behavior in future out of historical data (so-called training data). This means that a model has to be trained before it is applied to new data sets to predict a certain outcome.

Algo basically offers two models which are used for regression problems and the reduction of data dimensions, respectively. I refer to Section 3.3 for detailed information about this Java project.

Whereas I worked a lot on the latter one, my contribution to the project ***Data*** was little. It did not exceed the correction of implementation errors and the design of small graphical user interfaces. Hence I will just give a short introduction of its basic idea. *Data* shall handle the user management of CAP. Different rights shall be assigned to different user groups. Super users, e.g. are allowed to create or change models. Normal users can only apply the stored models to data.

The most important project without any doubt is ***FReND*** (Framework for Relational Numerical Data). It contains every used data structure such as *Relation* or *TimeStamp*, e.g. Again, my work field did not touch this project a lot. More or less, my implementations were based on the given concepts in *FReND*. Thus, some UML diagrams that are presented in chapter 3 may contain classes of this project.

CAP's graphical user interfaces are mainly located in the following project. ***GUI*** basically serves as the graphical interface between the user and CAP's underlying hidden algorithms. Its task is to convert the processed user input into information which can start, change or stop CAP processes. In Section 3.4 I will give a short outline of my contributions to this project.

The project ***PATClient*** was designed to have a second GUI for a different customer. Basically, its classes override certain Java functions that are distributed over several projects like *GUI* or *Algo*, e.g. Since the design of CAP allows to change components easily without effecting other components, customer's demands can be designed and implemented as an outstanding project. For further information about *PATClient* I refer to section 3.2.

SCR and in particular PhD Yuan Chao (yuan.chao@siemens.com) once developed a tool called *PowerPlot* for the visualization of scatter and function plots. Its source code can be found in CAP's project ***Plotting***. It was already used in the previous version of CAP, called *PowerMonitor*. This software was developed to monitor the past and future behavior of turbines in several U.S. power plants. *PowerMonitor* still runs successfully. Nevertheless its old software architecture is not maintained any more.

Hence, the idea for *CAP* as a totally new concept of *PowerMonitor* was born. Moreover, *CAP* shall also fulfill not only the needs of power plant customers. It was also developed to be flexibly extended or changed in order to serve a bigger group of customers. Nearly every real-world application producing online time series with the necessity of a describing model can be a potential challenge for *CAP*.

In the second half of my internship at SCR, my main task was to develop a Java Native Interface (JNI) between *CAP* and the Umetrics software Simca-P. Hence, I designed, implemented and tested the Java project ***SIMCA*** which mainly contains one JNI class to call the DLL functions of Simca-P. Since one of our customers is already familiar with Umetrics, CAP shall be able to use their concepts of data analysis tools instead of designing an own solution. However, I did implement certain functions on my own to compare the given software in a qualitative way. In Section 3.5.1 and Section 3.5.2 I will shortly discuss the basic concepts of *SIMCA* and its JNI Implementation, respectively.

Moreover CAP shall be able to connect to hardware devices for spectrometry in order to acquire online data. The German company *BRUKER OPTIK GmbH* (http://www.brukeroptics.de) developed a DLL called OPUS to read out spectrometers via TCP/IP. With the given DLL my task was to implement a JNI for CAP. I will give further information in Section 3.5.3.

1.4 Task Formulation

Not surprisingly, there are already established professional software tools for multivariate data analysis (MVDA). Recently, the company Umetrics (http://www.umetrics.com) brought out their new version of SIMCA-P. Although the software CAP was originally developed for the monitoring of machines to predict their maintenance, it is flexibly designed enough to serve as a MDVA tool.

My task was to extend CAP in multiple ways to tackle common MVDA problems. To solve this task I had access to different real-world data sets and several DLLs from miscellaneous companies. Before I will explain my subtasks in detail, let me give a short list of them:

- development of a graphical user interfaces for spectrometric data
- implementation of preprocessing techniques (compression, filtering, correction)
- realization of PCA and PLS as projection methods
- conception of a software interface to call *SIMCA-P* DLL functions for PLS
- design of hardware connection to data source with OPUS

First of all I was supposed to develop, implement and test a graphical user interface for a client/server application. It shall be used to visualize the incoming PAT data for the user. Normal sensor values of a variable in CAP are shown as line charts. Remember that a line chart is the general representation of a time series in most cases.

PAT data sets are usually composed of an arbitrary number of univariate variables (temperature, pressure, concentration, etc.) and one multivariate variable, the spectrum. The latter has to be plotted in a totally different way since it consists of several hundred frequencies. A false-color picture shall represent the spectrum such that it is easy for the user to see unregularities or features. In Section 3.4.3 I will state my approach more precisely. Furthermore you will find some exemplarily charts which illustrate the utilized ideas.

The next step shall cover common PAT preprocessing techniques for compression, filtering and data correction. Several approaches can be found e.g. in [3, 5, 8, 13, 15, 16, 18, 21] which empirically show good results. Usually the preprocessing is the most important step in the process of Knowledge Discovery in Databases (KDD) since there might be many different scenarios in the raw data. Unprocessed data may bias the model heavily such that good predictions cannot be achieved.

Due to the fact that we know where the data come from, it is advisable to use this knowledge in order to transform or preprocess the data. If we exploit the particularity of the process of data acquisition, we can effectively remove noise, outliers and replace missing values as well. I refer to Section 2.1 for further details.

After visualization and preprocession of the data, projection methods like PCA and PLS shall be implemented. They fulfill the actual term of a model since they abstract knowledge from the data. Hence both data mining methods shall be represented in CAP such that model parameters can be stored and future predictions for new instances can be computed.

PCA shall be used as data dimension reduction [2, 14]. It will find the most prominent structures in the data to a certain degree of dimensions which is chosen by the user. Usually this degree will be fixed to two as the user wants to plot the most principal component as a function of the second most principal one. In Chemometrics it is sufficient to have just two dimensions since the biggest part of all information can be projected to them. Thus, PCA compresses the data to a minimum without losing much information.

Partial least squares (PLS) regression is a latent-variable-based method for the linear modeling of the relationship between a set of response variables and a set of predictor variables (see [1, 4, 7, 10, 17, 22] for further details). The objective is to find relations between blocks of data by relating their latent variables.

The motivation to use PLS instead of other regression methods is related to the characteristics of the PAT data. Since the spectrum is nearly continuously recorded in time and frequency, adjacent values are highly correlated to each other. Thus, PLS will find the latent structures like PCA. It will offer a linear model to predict values for the response variables (here, the process variables) out of the given predictor variables (the spectrum) at a certain time stamp. Further insights from the algorithm point of view can be found in Section 2.2.2. I refer to the software architecture of PLS in Section 3.3.2.

Another objective is to combine the PLS modeling part of an already established tool from Umetrics, *SIMCA-P*, together with the power of CAP. Although implementing PLS by oneself does not pose a big problem, a certain customer of SCR, who demanded a GUI tool like CAP with PLS as modeling method, is very familiar with the usage of *SIMCA-P*. Thus this customer does not need to verify the results of an own developed PLS application any more.

SIMCA-P comes as a bunch of dynamic linked libraries (DLLs) including C header files. Since CAP is completely developed in Java, one has to use the Java Native Interface (JNI) to call native machine functions. I will refer to [9] for further information about JNI.

In order to call the given DLL functions from *SIMCA-P*, a JNI interface has to be implemented. Basically there shall be a wrapper DLL developed in C that serves as an interface between the *SIMCA-P* DLL and a Java class which contains all native methods that are needed.

To validate if the functions are called properly, test data sets for benchmark are given. Having a native Java class, there shall be a graphical user interface to call its functions in CAP. The software architecture of this interface is explained in detail, with examples and screen shots in Section 3.5.

Finally, my last objective was the development of an interface between CAP and a hardware device to acquire online data. With the help of the Bruker software *OPUS*, CAP shall be able to process the incoming data from a real-world application.

Hence CAP shall acquire data directly from the data source, like e.g. spectrometer, thermometer or other measurement devices. Thus, an own JNI shall call the given OPUS DLL functions. This approach shall look very similar to the JNI for *SIMCA-P*. A graphical user interface for CAP, however, was not demanded whilst my time at SCR in Princeton, NJ. I will refer to Section 3.5.4 for detailed insights into the hardware interface for OPUS.

1.5 Outline of this Report

The thesis is organized in four further parts. In Chapter 2 I will discuss the preprocessing and data mining techniques I introduced briefly before. In Chapter 3 I will give a short overview of some basic software engineering concepts of CAP. Chapter 4 describes some results of one specific case of spectral data. My conclusions in Chapter 5 will finalize this thesis.

Chapter 2

Applied Data Mining Techniques

In this chapter I want to present the data mining techniques that are used in CAP in order to tackle PAT specific problems. First, I will describe the preprocessing methods in Section 2.1. After that, I will introduce the main data mining algorithms PCA and PLS in Section 2.2. At the end of this chapter I will shortly discuss the analysis of data sets containing several experiments in Section 2.3. It will play an important role in Chapters 3 and 4 since chemometrical data usually consist of different batches.

2.1 Preprocessing Steps

As in nearly every case of data analysis, it may be comfortable to transform the data before applying further preprocessing steps or data mining algorithms. The easiest and most common techniques as centering and normalization are briefly mentioned in Section 2.1.1 and 2.1.2 respectively.

Empirical results have shown that a slightly different transformation than normalization sometimes perform better on PAT data sets. Since Pareto-Scaling is implemented in CAP, this technique is outlined in Section 2.1.3 as well.

There exists very special preprocessing methods for nearly every type of data. For diffuse near-infrared spectroscopy data as I was working on, one can find several specific approaches that empirically perform well. I will discuss one multiplicative transformation that was implemented in CAP in Section 2.1.4.

Of course there are many different methods to correct PAT data and to deal with their peculiarities. I do not want to discuss additional techniques since I did not implement any and they would exceed the bounds of this report. Further literature about this topic can be found in [3, 5, 21].

As in every KDD process, the preprocessing of data takes most of the time. It is necessary since real-world data is a priori not clean. First of all, a general problem occurs due to the measurement. There exists no way to measure the exact true value since every instrument to measure has a certain accuracy.

Especially there is thermic noise which is direct proportional to the temperature of the measurement environment. If one wants to reduce thermic noise to a minimum, one has to cool down the environment to the absolute null point of zero Kelvin (- 273.15 degrees Celsius) which is as good as impossible. Unfortunately, termic noise increases exponentially with increasing temperature.

Very often the amount of resulting elements of a chemical reaction do heavily depend on the environment temperature. For the experiments that were driven to acquire data which I will discuss about in this report, it was mandatory to have temperatures above 300 Kelvin (nearly room temperature) which is far away from the minimum of thermic noise. Hence, the acquired data will for sure be noisy.

Since thermic noise can be considered as white noise, it is Gaussian distributed over the noise. And hence it can be reduced by clever transformation methods like smoothing for example. I will introduce one smoothing algorithm in Section 2.1.5 which performs very well on spectrometric data.

Another serious problem might be the scenario of missing values in data sets. If one or more sensors that collect certain values do not work, one has to apply different techniques to substitute these values by approximations or estimates of the real values. Luckily, there were no missing values in data sets when they were acquired, I will just refer to the article of Nelson [13] that might be a good starting point for further studies.

2.1.1 Centering

Usually normalization (see Section 2.1.2) is applied to the given data. In case that all variables $\mathbf{X} = \{X_1, \ldots, X_p\}$ do have the same units and nearly the same range of numbers, there is maybe no need to divide data sets by their standard deviations. There would be no necessity to dispense with the units and the range since it does not change the correlations among the variables.

As in our case for the spectral data, all variables do have the units of wavelengths which is cm^{-1} and nearly the same range of numbers. However, they usually do not have the same expected value. This implies that the so-called centering as a special form of normalization might be applied as transformation method.

Centering is also known as "subtracting the mean" which implies that data are shifted or transformed to mean value of zero. It should be sufficient enough to outline the formula, since centering is probably the easiest technique and does not need any additional explanations.

First, one has to compute the mean $\bar{x}_k$ for every dimension or variable X_k, $k = 1, \ldots, p$ which is defined by

$$\bar{x}_k = \frac{1}{n} \sum_{i=1}^{n} x_{ik}. \tag{2.1}$$

Finally every value x_{ik} out of X_k is subtracted by its mean $\bar{x}_k$:

$$x'_{ik} = x_{ik} - \bar{x}_k \tag{2.2}$$

2.1.2 Normalization

If the variables do cover several units or substantially different ranges of numbers, then centering is in general not enough to transform data. Moreover data's standard deviation is transformed to one. This technique is called normalization to unit variance. If there exists no prior knowledge about the variables, it can be a very popular method.

After normalization is applied, the new transformed data do not carry any information about the ranges since all are equal. Thus every variable will have the same influence on the model.

Suppose that we already estimated the expected value or so-called mean $\bar{x}_k$. Then one has to estimate the standard deviation s for each variable X_k by

$$s_k = \frac{1}{n-1}\sqrt{\sum_{i=1}^{n}(x_{ik} - \bar{x}_k)^2}. \tag{2.3}$$

The normalization is performed in two steps. First, one has to subtract the mean $\bar{x}_k$ of each value x_{ik} whereas $X_k = (x_{1k}, \ldots, x_{nk})$. Then, this result is divided by the empiric standard deviation s_k

$$x''_{ik} = \frac{x_{ik} - \bar{x}_k}{s_k}. \tag{2.4}$$

2.1.3 Pareto-Scaling

In spectral analysis, frequently no scaling is used since variables share the same units and nearly the same ranges of numbers. In [8] we can find an explanation why one should find a trade off between no scaling and normalization.

The largest spectral variables might not be the most informative. Thus, in the absence of any scaling, these variables may dominate and obscure variation of interest in dimensions with lower intensity.

Due to setting changes meanwhile the experiment runs, it is possible that the measured intensities are directly proportional to these changes. This may create features that might disturb the actual process of data mining. Normalization helps to prevent a domination of those undesired features. However, it destroys the dominance of maybe most important variables.

To use an intermediate between these two extremes of no scaling and s_k scaling, we can apply Pareto-Scaling that uses $\frac{1}{\sqrt{s_k}}$ as scaling factor by the following formula

$$x^p_{ik} = \frac{x_{ik} - \bar{x}_k}{\sqrt{s_k}}. \tag{2.5}$$

2.1.4 Multiplicative Scatter Correction

In [5] one can find simple reasons for using a linearization of spectral data instead of the raw data. One prominent linearization is the multiplicative scatter correction (MSC) that was introduced by Martens et al. [11].

First of all, we can linearize the given data such that linear regression methods like PCA or PLS produce a better model. Second, physical particles of different sizes can scatter the light that is needed to get the spectral analysis.

Helland [5] also points out that MSC is motivated empirically. The plot of spectral values for a given set of similar samples against the mean spectral values very often results in a straight line. Of course, this assumption does not hold true for all spectra.

Assume that you want to transform spectral values by additive and multiplicative constants such that this transformation

$$\mathbf{x}_i^* = a_i \mathbf{1} + m_i \mathbf{x}_i \tag{2.6}$$

mainly decimate additive and multiplicative sample-dependent scattering caused by the particle problem. After some basic mathematical manipulations, the transformation formula (2.6) has the following form

$$x_{ik}^* = \bar{x} + m_i (x_{ik} - \bar{x}_i) \tag{2.7}$$

whereas $\bar{x}_i = 1/n \sum_k x_{ik}$ is the mean spectral value for a certain sample i and m_i for $i = 1, \ldots, n$ can be an arbitrary positive number. Hence, everything that is missing to apply the correction is the choice of the m_i.

Since the regression of spectral values against the mean spectral values is nearly linear, one can propose the following regression equation

$$x_{ik} = c_i + b_i \, \bar{x}_{.k} + e_{ik} \tag{2.8}$$

where $\bar{x}_{.k} = 1/n \sum_{i=1}^{n} x_{ik}$ are the values of the mean spectrum and e_{ik} is an error term. Solving (2.8) by partial derivation after c_i and b_i and using formula (2.7) leads to the missing values

$$(m_i)_{MSC} = \frac{1}{\hat{b}_i} = \frac{\sum\limits_{k=1}^{p} (\bar{x}_{.k} - \bar{x})^2}{\sum\limits_{k=1}^{p} (x_{ik} - \bar{x}_{i.})(\bar{x}_{.k} - \bar{x})}. \tag{2.9}$$

2.1.5 Savitzky-Golay Filter

In [15] one can find an efficient implementation for a low-pass filter that was developed by Savitzky and Golay [16]. These filters are mainly used to render the width and the height of noisy spectral lines.

First of all, let us recall the basic formula of digital filters for a given time series of values $f_i \equiv f(t_i)$ where $t_i \equiv t_0 + i\Delta$. This filter is given by

$$g_i = \sum_{n=-n_L}^{n_R} c_n \, f_{i+n} \tag{2.10}$$

whereas f_i is certain data with time stamp t_i. Thus, each original value f_i is mapped to a linear combination g_i of itself and certain neighbor values. It is clear that n_L represents the number of values before f_i in time and n_R after it, respectively.

The easiest averaging method is the so-called *moving window averaging.* Here the window size is fixed for some $n_L = n_R$ and each g_i is computed as average from f_{i-n_L} to f_{i+n_R} with constant $c_n = 1/(n_L + n_R + 1)$. Thus if f is constant or linear decreasing or increasing, then the filter won't enclose a bias. However, if the second derivative of f is not zero, there will be a bias. A local extremum e.g. will be reduced in its functional value.

Hence, narrow spectral lines would be flattened and wider. This bias is undesirable because those features are of physical interest. Moving window averaging does preserve the zeroth moment which is the area under the curve and it also maintains the first moment which is the mean position in time. Only the second moment which represents the width of the line is broken.

Now it is clear that one is interested in filter coefficients c_n that preserves even higher moments. Therefore one approximates the function in the moving window by a polynomial (usually quadratic or quartic). Thus, each point f_i is least-square fitted by a polynomial $a_0 + a_1 i + \cdots + a_M i^M$ of degree M to the values $f_{i-n_L}, \ldots, f_{i-n_R}$.

The computational effort would be huge if we had to compute the least-square fit for each point as described above. Fortunately, the coefficients of the fitted polynomial are themselves linear in the data values. Thus, they can be computed before the fitting which is very effective.

One can show that the coefficients c_n with $n = -n_L, \ldots, n_R$ can be computes as follows

$$c_n = \sum_{m=0}^{M} \left\{ (\mathbf{A}^T \cdot \mathbf{A})^{-1} \right\}_{0m} n^m \tag{2.11}$$

whereas

$$A_{ij} = i^j, \qquad i = -n_L, \ldots, n_R, \qquad j = 0, \ldots, M. \tag{2.12}$$

If we are only interested in data fitting, then a_0 as moment is enough and just the first row of the matrix $(\mathbf{A}^T \cdot \mathbf{A})^{-1}$ is necessary. For the computation of the k-th derivative $(0 \leq k \leq M)$, one has to return higher moments like $\left\{ (\mathbf{A}^T \cdot \mathbf{A})^{-1} \right\}_{k,m}$. Then, the degree of the polynomial is usually set to $M = 4$ or larger.

All in all, one can say that the performance of Savitzky-Golay filters heavily depends on the assumption that the underlying function can somehow be locally fitted by a polynomial. Only then this type of filter is very powerful. Fortunately, this assumption empirically holds true for nearly every spectral database.

2.2 Projection to Lower Dimensional Structures

In MVDA the amount of data that has to be analyzed is huge usually. Especially in Chemometrics where we can find spectral data, the number of dimensions is extremely large. Since a spectrum in PAT is recored almost continuously over time and frequency band, we can assume that most of the frequencies are highly correlated to each other. There exists several algorithms that exploit this knowledge. In this section I will present the two most prominent representatives of MVDA.

First, imagine that we want to visualize the most important structures in order to find dependencies. Hence, PCA will compute a projection matrix which is equivalent to a linear combination of a subset of all dimensions. This representation will be much smaller and it will also keep most of the information after reprojection to the original data. In Section 2.2.1 I will explain how this algorithm is motivated and give a pseudo-code implementation in CAP.

If we are supposed to solve a regression problem in PAT, an extension of PCA can be used. Instead of finding the maximum variation in the measured spectral data $\mathbf{X}$, which is done for PCA, PLS derives latent variables that maximize the covariance between $\mathbf{X}$ and the response variables $\mathbf{Y}$. PLS is realized in CAP as well. Further insights and its pseudo-code can be found in Section 2.2.2.

2.2.1 Principal Component Analysis

PCA is a very well-known technique in signal processing. It is used to find the principal components of given data. In 1901, Pearson [14] introduced this method and since then its applications are manifold. The power of PCA is that we can linearly transform the original system of coordinates such that the first new dimension will be the carrier of the biggest variance. The second new dimension will carry the second biggest variance and so on.

Assume that the expected value E of the given data $\mathbf{X}$ is 0. This we can easily achieve by applying centering 2.1.1 to the raw data.

The transformation equation

$$\mathbf{Y} = \mathbf{X}\mathbf{W} \tag{2.13}$$

will project the original data $\mathbf{X}$ to a different representation $\mathbf{Y}$ in a new system of coordinates by the projection matrix $\mathbf{W}$. This transformation is equivalent to the singular value decomposition (SVD) which is not explain in my report. SVD is just needed to obtain the matrix **W** that transforms the data without any loss of information.

PCA is optimal for keeping the subspace with the biggest information. It approximates the data by using the transformation matrix $\mathbf{W_l}$ which will transform the original data into a subset of h new dimensions having the biggest variances. I will briefly outline how to compute this transformation **W**.

A singular value decomposition of data matrix **X** leads to

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{2.14}$$

whereas the asterix $*$ stands for the mathematical operation *conjugate transpose*, **U** and **V** represent the left and right singular vectors, respectively and $\mathbf{\Sigma}$ is a main diagonal matrix containing the singular values.

One can prove that if **X** is a positive-definite Hermitian Matrix, then the singular values and singular vectors out of the SVD correspond to the eigenvalues and the eigenvectors of EVD, respectively:

$$\begin{aligned}\mathbf{X}^*\mathbf{X} &= \mathbf{V}\mathbf{\Sigma}^*\mathbf{U}^*\,\mathbf{U}\mathbf{\Sigma}\mathbf{V}^* = \mathbf{V}(\mathbf{\Sigma}^*\mathbf{\Sigma})\mathbf{V}^*\\ \mathbf{X}\mathbf{X}^* &= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*\,\mathbf{V}\mathbf{\Sigma}^*\mathbf{U}^* = \mathbf{U}(\mathbf{\Sigma}\mathbf{\Sigma}^*)\mathbf{U}^*\end{aligned} \tag{2.15}$$

The squares of the non-zero singular values of **X** are equivalent to the non-zero eigenvalues of either $\mathbf{X}^*\mathbf{X}$ or $\mathbf{X}\mathbf{X}^*$. Furthermore, the columns of U (left singular vectors) are eigenvectors of $\mathbf{X}\mathbf{X}^*$ and the columns of V (right singular vectors) are eigenvectors of $\mathbf{X}^*\mathbf{X}$.

Since all values of **X** are not complex, the conjugate transpose of **X** is equivalent to its transposed matrix. The latter results allow us to compute the EVD out of the (empirical) covariance matrix $\mathbf{S} = \mathbf{X}^T\mathbf{X}$ to gain the principal components of **X**. Remember that **X** was centered already or else this approach won't work.

After applying EVD to **S**

$$\mathbf{S} = \mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{D}\mathbf{V}^T \tag{2.16}$$

one will receive the eigenvalues stored in a main diagonal matrix **D** and the eigenvectors in a matrix **V** whereas

$$\mathbf{D} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_p \end{pmatrix}, \qquad \mathbf{V} = \begin{pmatrix} \lambda_{11} & \lambda_{21} & \cdots & \lambda_{p1} \\ \lambda_{12} & \lambda_{22} & \cdots & \lambda_{p2} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{1p} & \lambda_{2p} & \cdots & \lambda_{pp} \end{pmatrix}. \tag{2.17}$$

Usually, one has to sort the eigenvectors $\vec{\lambda_i} = (\lambda_{i1}, \lambda_{i2}, \ldots, \lambda_{ip})$, $i = 1, \ldots, p$, according to their eigenvalues λ_i. The bigger the eigenvalue, the more information or variance is put into this dimension. One can also compute the (empirical) variances s_i of each single dimension as follows:

$$s_i = \frac{\lambda_i}{\sum\limits_{j=1}^{p} \lambda_j} \tag{2.18}$$

After sorting, the k-th eigenvector or column of $\mathbf{V}$ is related to the kth biggest eigenvalue, for $k = 1, \dots, p$. Now we can keep $h \leq p$ dimensions of $\mathbf{V}$ to construct our model $\mathbf{W_h}$. Finally, one computes $\mathbf{Y}_{n,h\leq p} = \mathbf{X}_{n,p}\mathbf{W}^T_{h\leq p,p}$.

Algorithm 1 Calculate principal components of $\mathbf{X}$

```
Require: data $\mathbf{X}$, mean values $\bar{\mathbf{X}} = \{\bar{X}_1, \dots, \bar{X}_p\}$, standard deviation $\bar{\mathbf{s}} = \{s_1, \dots, s_p\}$
Ensure: $\mathbf{S} = \mathbf{X}'^T\mathbf{X}' = \mathbf{V}\mathbf{D}\mathbf{V}^T$
1: $\mathbf{X}' \leftarrow (\mathbf{X} - \bar{\mathbf{X}})/\mathbf{s}$                    // normalize $\mathbf{X}$
2: $\mathbf{S} \leftarrow \mathbf{X}'^T\mathbf{X}'$                    // covariance matrix calculation
3: $[\mathbf{V}, \mathbf{D}] \leftarrow$ eigenvalueDecomposition($\mathbf{S}$)    // EVD of covariance matrix
4: sort $\mathbf{V}$ after $\mathbf{D}$
5: return $\mathbf{V}$
```

2.2.2 Partial Least Squares

Especially in Chemometrics but in other applied sciences as well, we can find PLS regression methods. As mentioned in [7] the main differences between established regression methods like e.g. multiple linear regression or ridge regression are

- the common situation that there are many variables but few samples
- the stability of predictors derived from PLS methods in terms of minimum number of variables.

PLS methods select those components that reduce the covariance matrix $\mathbf{X}^T\mathbf{Y}$ at most. Therefore the number of variables will be minimal. The basic idea of PLS is founded on the nonlinear iterative partial least squares (NIPALS) (see e.g. [4]) which describes PCA. However, the principal components are not computed at once. It is done iteratively as follows.

The fundamental concept of PLS are two outer relations for $\mathbf{X}$ and $\mathbf{Y}$, and one inner relation that combines the outer ones. The outer relations are derived from the NIPALS algorithm. In PCA, we only take the first h dimensions with the biggest variances. Here, we do the same for the independent variables and the dependent ones, respectively.

The outer relation for $\mathbf{X}$ is

$$\mathbf{X} = \mathbf{T}\mathbf{P}^T + \mathrm{Err}_X = \sum \mathbf{t_i}\mathbf{p_i}^T + \mathrm{Err}_X \tag{2.19}$$

and for $\mathbf{Y}$ in the same manner

$$\mathbf{Y} = \mathbf{U}\mathbf{Q}^T + \mathrm{Err}_Y = \sum \mathbf{u_i}\mathbf{q_i}^T + \mathrm{Err}_Y. \tag{2.20}$$

The aim is to approximate $\mathbf{Y}$ as good as possible, hence to minimize $\|\mathrm{Err}_Y\|$ and to get a meaningful relation between $\mathbf{X}$ and $\mathbf{Y}$. If one assumes a linear dependency between $\mathbf{X}$ and $\mathbf{Y}$, then we can build the following model

$$\mathbf{U} = \mathbf{T}\mathbf{B} \tag{2.21}$$

where the matrix $\mathbf{B}$ represents the linear regression coefficients. This model, however, is the simplest one that we can choose. Since the principal components for $\mathbf{X}$ and $\mathbf{Y}$ are computed separately, there won't be a very good relation between them.

There are miscellaneous approaches to put more information into the inner relation. In [10] e.g., we can find the *kernel algorithm for PLS* which is recommended if the number of objects n is large.

The algorithm that I have chosen is called *SIMPLS* and was introduced by de Jong [1]. A detailed implementation of it can be found in Algorithm 2. However, before I will explain SIMPLS more precisely, let me mention some profound ideas of it.

Matrix $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_p}\}$ is transformed to A latent variables, the so-called factor scores $\mathbf{T} = \{\mathbf{t_1}, \mathbf{t_2}, \dots, \mathbf{t}_A\}$, where $A \leq p$. This can be compared to PCA where the predictor variables are reduced to few latent variables.

Each single factor $\mathbf{t}_a$, $a = 1, 2, \dots, A$, is computed iteratively by NIPALS. These orthogonal factors are used to fit all n observations to all m dependent variables $\mathbf{Y} = \{\mathbf{y_1}, \mathbf{y_2}, \dots, \mathbf{y_m}\}$. Now, I will describe Algorithm 2 step by step since it might not be very easy to understand.

First of all, both matrices $\mathbf{X}$ and $\mathbf{Y}$ are transformed to the mean value of zero in line 1 and 2, respectively. This can be achieved by centering which is explained in Section 2.1.1.

For convenience, let matrix $\mathbf{M}_0 \equiv \mathbf{M} - \bar{\mathbf{M}}$ be the centered form of $\mathbf{M}$ whereas $\bar{\mathbf{M}}$ is the mean of $\mathbf{M}$. Furthermore, let the matrix $\mathbf{M_A}$ with the final index A be equivalent to the resulting one $\mathbf{M}$.

The covariance matrix $\mathbf{S_1} \leftarrow \mathbf{X}_0^T\mathbf{Y}_0$ is computed in line 3. The index a, here $a = 1$, represents the iteration step of the matrix.

After initialization, the following steps will be repeated by a loop that runs from $a = 1$ to A. The very first command is the computation of the most dominant singular vector or eigenvector $\mathbf{q}_a$ of $\mathbf{S}_a$ by applying SVD or EVD, respectively in line 5. Remember that SVD in our case is equivalent to EVD as I pointed out in Section 2.2.1.

Then, the so far not normalized weight vector $\mathbf{r}_a \leftarrow \mathbf{S}_a \cdot \mathbf{q}_a$, is computed in line 6. Hence, $\mathbf{r}_a$ is a singular vector as well. Its normalization is performed in line 10.

In line 7, the orthogonal latent variable of $\mathbf{X}$, the factor score $\mathbf{t}_a \leftarrow \mathbf{X}_0 \cdot \mathbf{r}_a$, is computed. Since all factor scores shall be normalized to mean of zero and standard deviation of one, we can find these steps in lines 8 and 9, respectively.

Algorithm 2 Calculate the PLS model for $\mathbf{Y} = \mathbf{X} \cdot \mathbf{B}$

```
Require: $A > 0, \mathbf{X}, \mathbf{Y}$
Ensure: Matrix $\mathbf{B}$ with $\mathbf{Y} = \mathbf{X} \cdot \mathbf{B}$
 1: $\mathbf{X}_0 \leftarrow \mathbf{X} - \bar{\mathbf{X}}$                         // center $\mathbf{X}$
 2: $\mathbf{Y}_0 \leftarrow \mathbf{Y} - \bar{\mathbf{Y}}$                         // center $\mathbf{Y}$
 3: $\mathbf{S} \leftarrow \mathbf{X}_0^T \cdot \mathbf{Y}_0$                       // covariance matrix calculation
 4: for $a = 1$ to $A$ do                                       // per dimension
 5:     $\mathbf{q} \leftarrow$ getDominantEigenvector($\mathbf{S}^T \cdot \mathbf{S}$)   // Y block factor weights
 6:     $\mathbf{r} \leftarrow \mathbf{S} \cdot \mathbf{q}$                         // X block factor weights
 7:     $\mathbf{t} \leftarrow \mathbf{X}_0 \cdot \mathbf{r}$                       // X block factor scores
 8:     $\mathbf{t} \leftarrow \mathbf{t} - \bar{\mathbf{t}}$                       // center scores
 9:     $\mathbf{t} \leftarrow \mathbf{t}/\|\mathbf{t}\|$                           // normalize scores
10:     $\mathbf{r} \leftarrow \mathbf{r}/\|\mathbf{t}\|$                           // adapt weights accordingly
11:     $\mathbf{p} \leftarrow \mathbf{X}_0^T \cdot \mathbf{t}$                     // X block factor loadings
12:     $\mathbf{q} \leftarrow \mathbf{Y}_0^T \cdot \mathbf{t}$                     // Y block factor loadings
13:     $\mathbf{u} \leftarrow \mathbf{Y}_0^T \cdot \mathbf{q}$                     // Y block factor scores
14:     $\mathbf{v} \leftarrow \mathbf{p}$                                          // initialize orthogonal loadings
15:     if $a > 1$ then
16:         $\mathbf{v} \leftarrow \mathbf{v} - \mathbf{V} \cdot (\mathbf{V}^T \cdot \mathbf{p})$   // make $\mathbf{v}\perp$ previous loadings
17:         $\mathbf{u} \leftarrow \mathbf{u} - \mathbf{T} \cdot (\mathbf{T}^T \cdot \mathbf{u})$   // make $\mathbf{u}\perp$ previous $\mathbf{t}$ values
18:     end if
19:     $\mathbf{v} \leftarrow \mathbf{v}/\|\mathbf{v}\|$                           // normalize orthogonal loadings
20:     $\mathbf{S} \leftarrow \mathbf{S} - \mathbf{v} \cdot (\mathbf{v}^T \cdot \mathbf{S})$   // deflate $\mathbf{S}$ with respect to current loadings
21:     Store $\mathbf{r}, \mathbf{t}, \mathbf{p}, \mathbf{q}, \mathbf{u}$ and $\mathbf{v}$ into $\mathbf{R}, \mathbf{T}, \mathbf{P}, \mathbf{Q}, \mathbf{U}$ and $\mathbf{V}$, respectively.
22: end for
23: $\mathbf{B} \leftarrow \mathbf{R} \cdot \mathbf{Q}^T$                           // regression coefficients
24: $\mathbf{varX} \leftarrow \mathrm{diag}(\mathbf{P}^T \cdot \mathbf{P})/(n-1)$   // variance explained for $\mathbf{X}_0$ variables
25: $\mathbf{varY} \leftarrow \mathrm{diag}(\mathbf{Q}^T \cdot \mathbf{Q})/(n-1)$   // variance explained for $\mathbf{Y}_0$ variables
26: return $\mathbf{B}$
```

The loading vector $\mathbf{p}_a \leftarrow \mathbf{X}_0^T \cdot \mathbf{t}_a$ of the predictor matrix is computed in line 11. It represents how strong the $\mathbf{X}$ variables are related to the first PLS factor $\mathbf{t}_a$. Remember that we want to decompose $\mathbf{X}$ into $\mathbf{T} \cdot \mathbf{P}^T$.

In the same manner as the latter computation, the loading vector $\mathbf{q}_a \leftarrow \mathbf{Y}_0^T \cdot \mathbf{t}_a$ of the dependent variables is computed in line 12. Please recall our goal to decompose $\mathbf{Y}$ into $\mathbf{U} \cdot \mathbf{Q}^T$. Hence the computation of the a-th factor scores of the response variables Y is carried out by $\mathbf{u}_a \leftarrow \mathbf{Y}_0^T \mathbf{q}_a$ in line 13.

In order to go on with the next step $a + 1$, it is necessary to subtract the latent variable from $\mathbf{S}_a$. This procedure is called deflating. Therefore, the projection onto $\mathbf{P}_a$ will be performed as a sequence of orthogonal projections. Hence, it is necessary to have an orthonormal basis of $\mathbf{P}_a$, named $\mathbf{V}_a \equiv \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_a\}$.

Initially, $\mathbf{V}_1$ will be $\mathbf{v}_1 \propto \mathbf{p}_1$ as pointed out in line 14. If $a > 1$, then $\mathbf{V}_a$ will be obtained by orthogonalization of $\mathbf{P}_a$ in line 16. The same orthogonalization step is carried out for the $\mathbf{Y}$ factor scores $\mathbf{u}_a$ with respect to $\mathbf{X}$ factor scores $\mathbf{t}_a$. This ensures that we maximize the covariance of both matrices $\mathbf{X}$ and $\mathbf{Y}$.

Before deflating $\mathbf{S}_a$ (see line 20) in order to remove the information that was found in the current step, one has to normalize the already orthogonal loadings which is performed in line 19. Finally, all computed vectors $\mathbf{r}_a, \mathbf{t}_a, \mathbf{p}_a, \mathbf{q}_a, \mathbf{u}_a$ and $\mathbf{v}_a$ are added as a-th column to their matrices $\mathbf{R}_a, \mathbf{T}_a, \mathbf{P}_a, \mathbf{Q}_a, \mathbf{U}_a$ and $\mathbf{V}_a$ in line 21.

Finally when the loop is done, the regression coefficients $\mathbf{B} \leftarrow \mathbf{R}_A \cdot \mathbf{Q}_A^T = \mathbf{R} \cdot \mathbf{Q}^T$ which are computed in line 23 are returned in line 26. If we are interested in the variance of $\mathbf{X}$ and $\mathbf{Y}$, SIMPLS can also compute these vectors. The variance computation is carried out in line 24 and 25, respectively.

Compared to other PLS regression methods, SIMPLS is faster than the standard *NIPALS* in computational speed because the construction of deflated matrices is just applied to $\mathbf{S}$. SIMPLS will lead to the same results as NIPALS if the problem is univariate. The produced model will be slightly different from NIPALS, however, if $\mathbf{Y}$ is multivariate. Further differences can be found in [1].

2.3 Batch Analysis

Especially in Chemometrics, we can find the problem of batch analysis. A batch is a single experiment that will be carried out again with slightly different or equal parameters. If the similarities between the b batches are large to a certain degree, statistics like minimum, maxiumum and averaging may be used to approximate the true values.

If one wants to compute e.g. PCA (see Section 2.2.1) for a data set with batches, one can perform PCA for each single batch as well and combine them by statistics. These results will be presented with the normal PCA that was carried out for the whole data. For each batch, its own principle components will be presented. However, the curves produced by the batch analysis will be the same for each single batch.

Usually the duration and the data resolution of each batches are nearly the same. However, there will be a shortest batch. Its duration $\Delta t^{\text{shortest}}$ is taken as a measure for the maximal length of the statistics' time series. Then, the first $\Delta t^{\text{shortest}}$ time points from each batch are taken to compute the statistics.

Averaging over all batches might reveal the so-called *golden batch* which is supposed to be found by

$$\forall t \in \left[0, \Delta t^{\text{shortest}}\right] : \quad f_{\text{golden}}(t) = \tfrac{1}{b} \sum_{i=1}^{b} f\left(t_0^{i\text{-th batch}} + t\right) \tag{2.22}$$

whereas $t_0^{i\text{-th batch}}$ indicates the first time stamp of the i-th batch. The computation of a missing value f at time point t can be carried out by interpolation between its neighbor points.

Moreover it is also possible to have upper and lower thresholds for outlier detection by changing (2.22) slightly. The minimum and maximum over all batches provides a min-max-tunnel in order to restrict unseen values to a certain range of possibilies.

$$\forall t \in \left[0, \Delta t^{\text{shortest}}\right] : \quad f_{\text{min}}(t) = \min_{i=1}^{b} f\left(t_0^{i\text{-th batch}} + t\right) \tag{2.23}$$

$$\forall t \in \left[0, \Delta t^{\text{shortest}}\right] : \quad f_{\text{max}}(t) = \max_{i=1}^{b} f\left(t_0^{i\text{-th batch}} + t\right) \tag{2.24}$$

An example of the tunnel and the golden batch is shown in Figure 2.1. Since (2.22), (2.23) and (2.24) are computed once, the functions $f_{\text{golden}}, f_{\text{min}}$ and f_{max} will be the same for each batch.

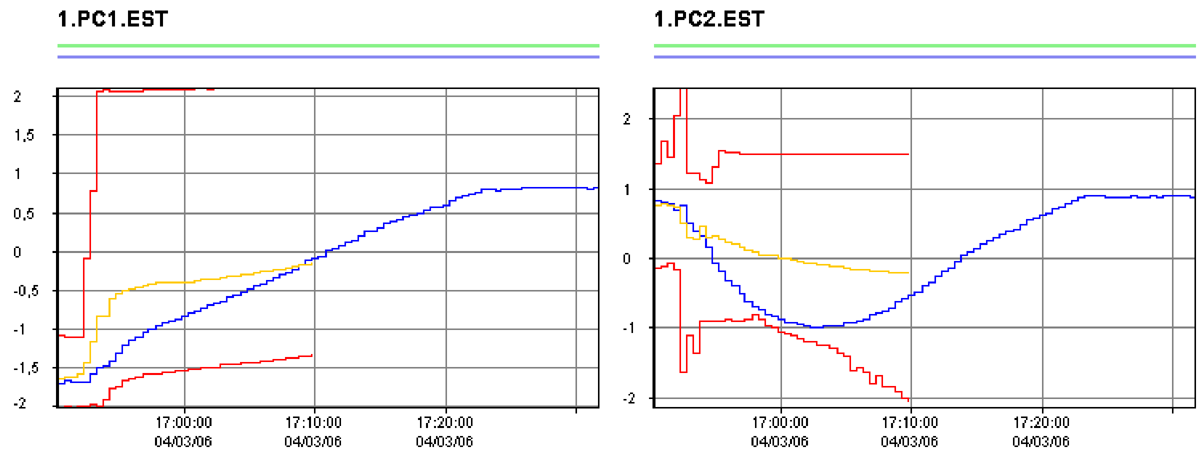


Figure 2.1: Tunnel visualization of the first two principal components (PC) of one paracetamol data's batch versus time. The first PC is shown on the left side and the second PC on the right, resp. The blue curve indicates the PC computed with all data. The minimum (lower red curve), maximum (upper red curve) and the golden batch (orange curve) were computed by taking the minimum, maximum and average over all batches, resp. Both figures are produced by CAP with a Spectrum Model (see Section 3.3.1).

Note that the tunnel and the golden batch will usually end before the batch. This is due to the fact that the batch statistics are just computed until the end of the shortest batch since there would be no data available after $\Delta t^{\text{shortest}}$.

Chapter 3

CAP Software Architecture

In the present chapter I will briefly outline the software architecture CAP (System for Condition Assessment and Prognosis). The presented insights are definitely not complete and sometimes not profound enough to address all problems. Hence, the shown UML diagrams do not contain all methods or attributes. However, the basic concepts are state out clearly.

Since CAP was not developed to solve PAT problems in first instance, I will present the actual task of CAP in Section 3.1 which is the detection of machine faults that are predicted for the future. The PAT client I was basically working on will be discussed in Section 3.2. The next Section 3.3 will show insights into the different machine learning models and algorithms that exist in CAP to analyze data. Since CAP has a bunch of graphical user interfaces for arbitrary tasks, I will only refer to some of my own GUIs in Section 3.4. The final task whilst my internship was the development of *Java Native Interfaces* for CAP in order to connect to extern machine learning tools and measurement devices, respectively. Both interfaces are presented in Section 3.5.

3.1 Fault Detection with CAP

Originally, CAP was developed to be a software solution for condition based monitoring and predictive maintenance that learns complex system behavior based on historical sensor data. A statistical model is created during the training phase which is then able to detect significant deviations from the normal sensor pattern. The deviations are then analyzed in order to recognize and interpret faults and predict remaining operation time until failure.

The previous version of CAP which is called PowerMonitor was developed to predict machine faults for huge electricity generator turbines. However, CAP shall be flexible enough to maintain not only turbines but nearly every machine that produces a time series of sensor data.

Basically, CAP includes three main components as they can be seen in Figure 3.1:

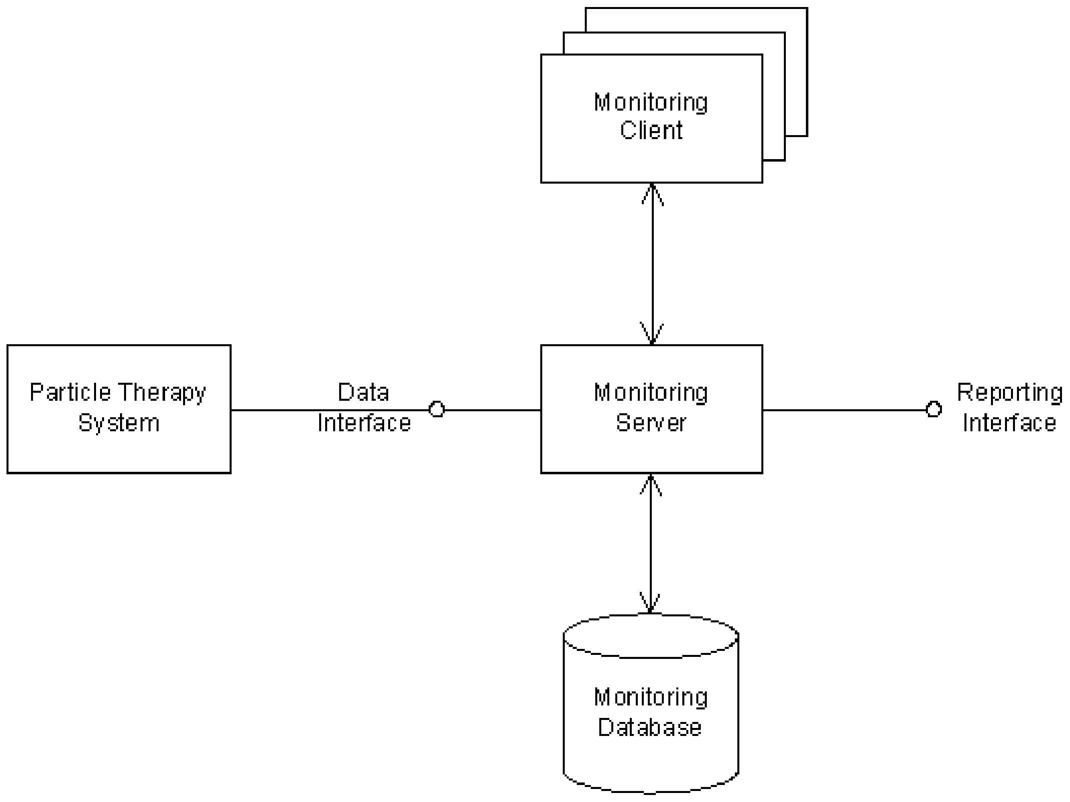


Figure 3.1: Components of CAP for a certain client. It includes a monitoring server that analyzes incoming data from an arbitrary system, predicts faults and creates alarm events or tickets. The monitoring database contains all configuration and working data. The monitoring client provides a GUI for server administrator tasks and a visualization of data and fault information.

a monitoring server, a monitoring database and a monitoring client. The monitoring server provides the analysis of process data. It predicts faults and creates alarm events or tickets in order to replace the machine before it will produce a failure. The monitoring server is able to continuously monitor process data in an unattended fashion.

The monitoring database contains all data such as models, limits, rules, trend information that is used internally by the monitoring server. It also stores all information necessary to reproduce historical monitoring results for faults that have been reported through the reporting interface.

The monitoring client provides a graphical user interface for all administrative tasks related to the monitoring server, as well as for visualization of data and fault information. The monitoring client provides the user the ability to start and stop continuous monitoring and to manually trigger the condition assessment process on the monitoring server.

In addition to the regular monitoring client application, a web-based monitoring client shall be provided in future. The monitoring server, monitoring database and monitoring client shall be able to run on separate computers and communicate through the network.

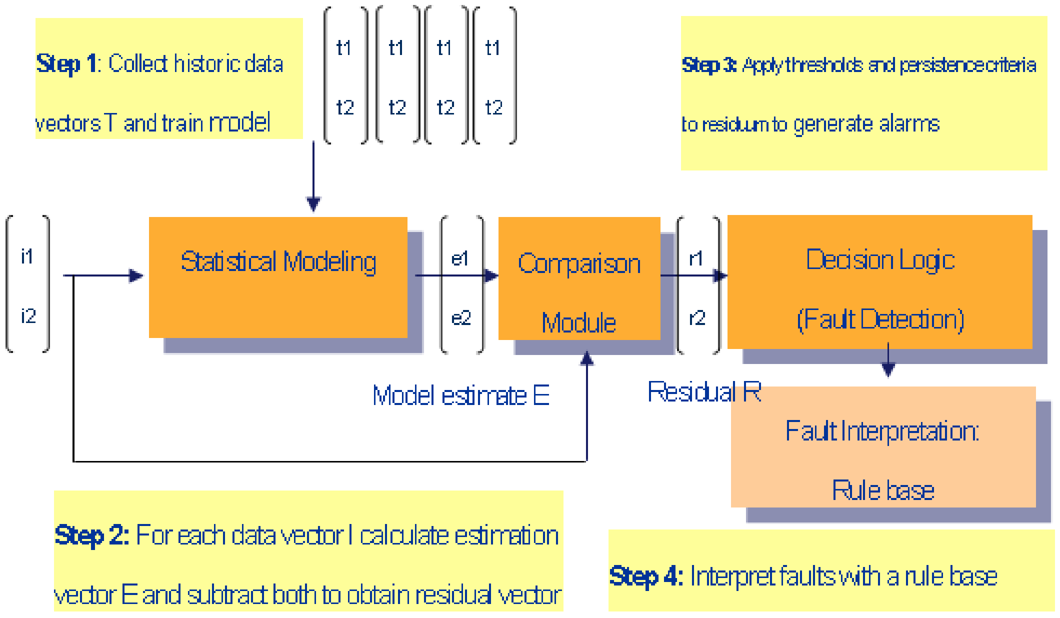

Step 1: Collect historic data vectors T and train model
t1
t2
Step 3: Apply thresholds and persistence criteria to residuum to generate alarms
i1
i2
Statistical Modeling
e1
e2
Comparison Module
r1
r2
Decision Logic (Fault Detection)
Model estimate E
Residual R
Fault Interpretation: Rule base
Step 2: For each data vector I calculate estimation vector E and subtract both to obtain residual vector
Step 4: Interpret faults with a rule base

Figure 3.2: Main functions of CAP.

CAP includes several basic functionalities as shown in Figure 3.2 such as the import of condition Data and process data. CAP loads sensor recordings within specified period of time from an external historical data archive. The multivariate sensor data shall be used to train statistical models and for testing and periodic monitoring. In the basic version data shall be loaded from files as 2D arrays of equidistant, synchronized sensor values. In the extended CAP version, data access shall be more flexible following the OPC standard.

Data preprocessing and feature extraction is provided in CAP as well. The preprocessing module shall provide configurable mathematical functions to define features (also computed or virtual sensors) from raw sensor data. Features are used to incorporate knowledge about the physical system. Good features help to reduce data volume and improve accuracy of the subsequent fault detection and interpretation.

Further preprocessing steps include data interpolation and normalization. The system is extensible to accommodate additional preprocessing methods. This point plays an important role for the PAT preprocessing steps as presented in Section 2.1.

Another function in CAP is the creation of modes and data filters. Modes are defined with a rule editor in order to filter the processed data and to create specialized models that capture a specific operational state. This functionality might be useful as well for the reduction of outliers in data.

The main task of CAP is the training of a statistical model with historical data. The trained statistical model takes an array of actual sensor measurements as input in order to compute an array of sensor estimates as output. The estimates can be interpreted as the sensor vector that is closest to the measurement vector but within the trained distribution obtained from normal, fault free operation.

The difference between sensor measurement and sensor estimate is called residual. Under normal operation the residual is close to zero. A residual which is significantly different form zero indicates a fault condition or at least a condition that has not been observed during training.

The user shall be given the choice to specify those sensors that measure the process drivers as far as this knowledge is available. Based on this input the system shall automatically distinguish whether deviations from normal are due to a new operation condition or due to a real fault.

After a model is trained, stored and set active for monitoring, the system shall periodically check for new data and apply all activated models to search for faults. For each sensor an alarm shall be produced at a certain timestamp if the residue is outside its normal range. The operator shall also be able to specify a persistency window size.

In such a case, the system shall look at all time stamps within this window, which ends at the current time stamp. If the residue is outside its normal range at more than 50 percent of the time stamps within this window, an alarm shall be produced for the current time stamp. Otherwise, no alarm is to be produced at this time stamp.

3.2 The Client for PAT

In the latter section I discussed about the actual task for CAP. Machine monitoring is not the only ability of CAP, however. PAT problems such as described in Section 4.2 can also be solved with our framework.

Fundamentally, one just has to exchange the functionality and the look of CAP in order to produce a complete new software out of the same basis. Whereas it was important to predict machine outages for CAP, the client for PAT named *SCR ProcessMonitor* (see Figure 3.3) shall be able to analyze spectral and process data recorded whilst a chemical reaction.

The common point of both problems is the regression step which is applied. If we want to predict future values for the behavior of machines or the temperature of a chemical process, we will use the historic data to compute a model that is able to approximate the desired predictions. The PAT client offers three different statistical models which have different machine learning algorithms as well. I refer to Section 3.3 for a detailed description of them.

The analysis of data containing spectral and process values is different from the standard modeling process for machine sensor data. First of all, we deal with many more dimensions since the spectrum is nearly recorded continuously in frequency. This causes certain treatments for visualization, preprocession and modeling. In Section 4.3 I will give an explanation of the present spectral and process data that were used for

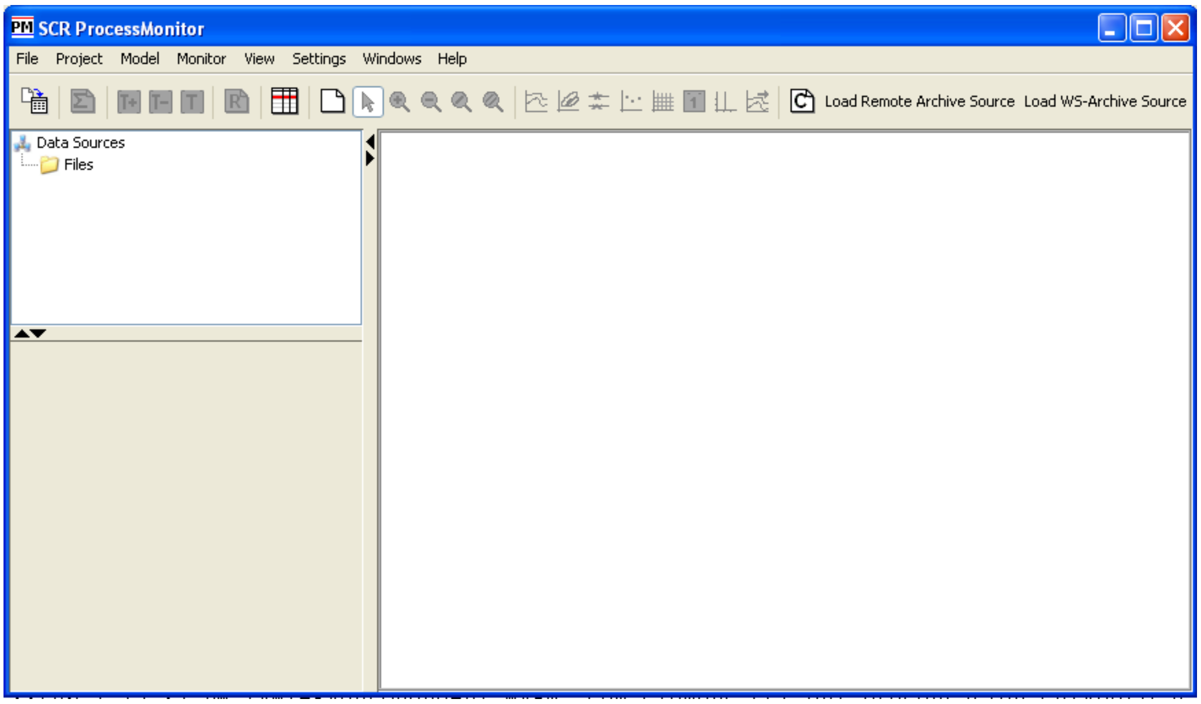

SCR ProcessMonitor
File Project Model Monitor View Settings Windows Help
Load Remote Archive Source Load WS-Archive Source
Data Sources
Files

Figure 3.3: The GUI for the PAT client named *SCR ProcessMonitor*.

PAT.

To conclude this section, I will describe the single working steps that have to be done to analyze data with the PAT client. The first step is the loading of a new data source. This could be an arbitrary text file that contains the sensor information in a 2D array. After loading is done, a new data source was created.

Now, the user can open a new project that will be related to the loaded data source. In Section 3.3 I will explain more in detail the concept of a project. The third step is to add a new model to the selected project. One can choose between different types of models depending on the task that has to be solved.

The next step is to add a collection of data source's sensors to the model. This can be seen as sensor or variable filter. The fifth step is the filter in time since training time for the model has to be selected. After that, the model can be trained.

3.3 Models and Algorithms

In this section I will introduce the different models and algorithms that exist in CAP. Therefore I will briefly outline the concept of a relation and a project. Both terms are needed to understand the KDD process in CAP.

Whenever a data source such as a file or database is loaded into CAP, a relation is created out of it. A relation consists of attributes and tuples. It is the main interface of the FReND package (Framework for Relational Numerical Data) and represents numerical data in a relational structure similar to a table in a relational database.

If one wants to create one or more models related to a certain data source, one can create a so-called project that refers to the selected data source. Hence one project takes care of one machine or job and contains several models. Each model is a state estimator and produces derived attributes.

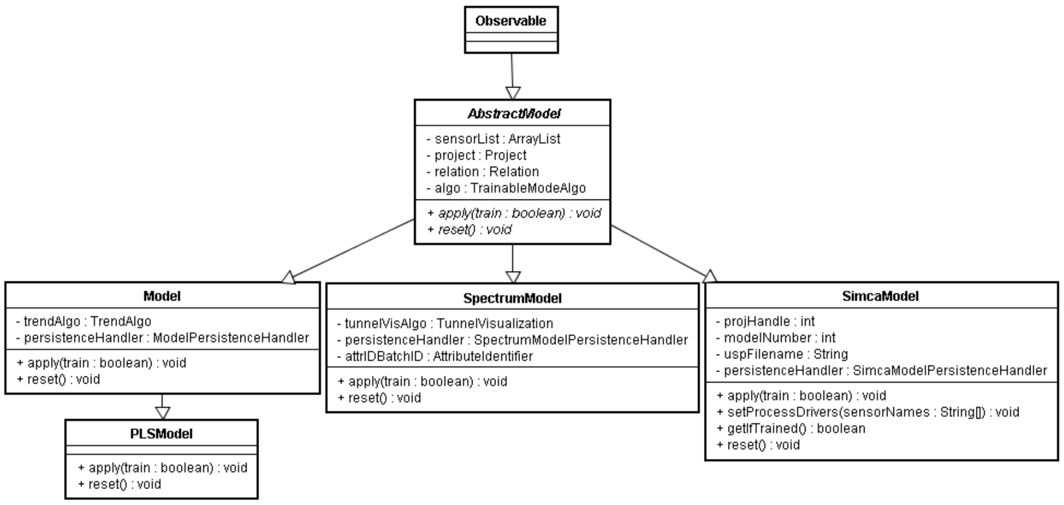

Observable
AbstractModel
- sensorList : ArrayList
- project : Project
- relation : Relation
- algo : TrainableModeAlgo
+ apply(train : boolean) : void
+ reset() : void
Model
- trendAlgo : TrendAlgo
- persistenceHandler : ModelPersistenceHandler
+ apply(train : boolean) : void
+ reset() : void
SpectrumModel
- tunnelVisAlgo : TunnelVisualization
- persistenceHandler : SpectrumModelPersistenceHandler
- attrIDBatchID : AttributeIdentifier
+ apply(train : boolean) : void
+ reset() : void
SimcaModel
- projHandle : int
- modelNumber : int
- uspFilename : String
- persistenceHandler : SimcaModelPersistenceHandler
+ apply(train : boolean) : void
+ setProcessDrivers(sensorNames : String[]) : void
+ getIfTrained() : boolean
+ reset() : void
PLSModel
+ apply(train : boolean) : void
+ reset() : void

Figure 3.4: UML diagram of the model architecture of CAP. `AbstractModel` offers the frame for all inherited models. Each `AbstractModel` has a list of sensors to get the data of the relation, a project and an algorithm. A project serves as collection of different models related to a certain data source. Every `AbstractModel` has its own training algorithm to analyze data. The method `apply()` is the actual training step, whereas `reset()` is called whenever the model has to be reset.

A model in general is a `AbstractModel` that represents the frame for all inherited implementations of this abstract class as they are shown in Figure 3.4. It contains a list of sensors that were selected by the user out of the set of all sensors given by the relation. Moreover, the model contains the project it belongs to which makes it easier to reference the model. The relation is stored in the model as well to access attributes and tuples on the one hand. On the other hand, the algorithm's computed attributes from the model itself are stored in the relation, too. The training of the model is carried out in `apply()`. Its commands for resetting are done in `reset()`.

The inheritors `SpectrumModel`, `PLSModel` and `SimcaModel` are explained more in detail in Sections 3.3.1, 3.3.2 and 3.3.3, respectively. Since the inheritor `Model` is only used for state estimation and the prediction of machine outages, but it is not relevant for PAT, `Model` is not considered in this report.[1]

Since all of these three special outcomes of `AbstractModel` do have their own training

[1]Note that `PLSModel` inherits methods and attributes from `Model` directly.

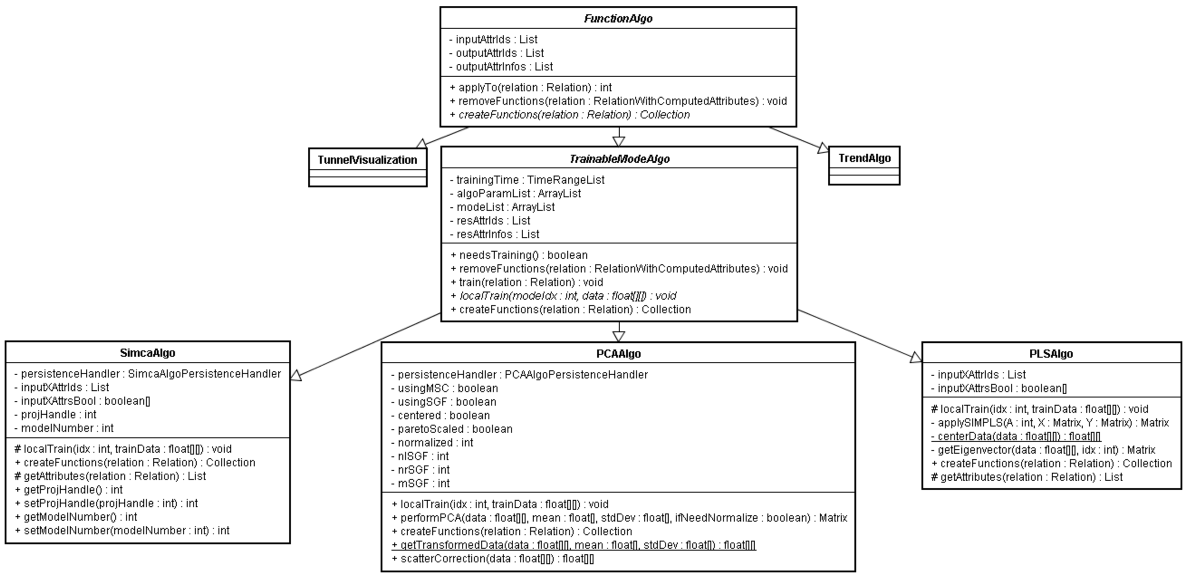

FunctionAlgo
- inputAttrIds : List
- outputAttrIds : List
- outputAttrInfos : List
+ applyTo(relation : Relation) : int
+ removeFunctions(relation : RelationWithComputedAttributes) : void
+ createFunctions(relation : Relation) : Collection
TunnelVisualization
TrainableModeAlgo
- trainingTime : TimeRangeList
- algoParamList : ArrayList
- modeList : ArrayList
- resAttrIds : List
- resAttrInfos : List
+ needsTraining() : boolean
+ removeFunctions(relation : RelationWithComputedAttributes) : void
+ train(relation : Relation) : void
+ localTrain(modeIdx : int, data : float[][]) : void
+ createFunctions(relation : Relation) : Collection
TrendAlgo
SimcaAlgo
- persistenceHandler : SimcaAlgoPersistenceHandler
- inputXAttrIds : List
- inputXAttrsBool : boolean[]
- projHandle : int
- modelNumber : int
# localTrain(idx : int, trainData : float[][]) : void
+ createFunctions(relation : Relation) : Collection
# getAttributes(relation : Relation) : List
+ getProjHandle() : int
+ setProjHandle(projHandle : int) : int
+ getModelNumber() : int
+ setModelNumber(modelNumber : int) : int
PCAAlgo
- persistenceHandler : PCAAlgoPersistenceHandler
- usingMSC : boolean
- usingSGF : boolean
- centered : boolean
- paretoScaled : boolean
- normalized : int
- nlSGF : int
- nrSGF : int
- mSGF : int
+ localTrain(idx : int, trainData : float[][]) : void
+ performPCA(data : float[][], mean : float[], stdDev : float[], ifNeedNormalize : boolean) : Matrix
+ createFunctions(relation : Relation) : Collection
+ getTransformedData(data : float[][], mean : float[], stdDev : float[]) : float[][]
+ scatterCorrection(data : float[][]) : float[][]
PLSAlgo
- inputXAttrIds : List
- inputXAttrsBool : boolean[]
# localTrain(idx : int, trainData : float[][]) : void
- applySIMPLS(A : int, X : Matrix, Y : Matrix) : Matrix
- centerData(data : float[][]) : float[][]
- getEigenvector(data : float[][], idx : int) : Matrix
+ createFunctions(relation : Relation) : Collection
# getAttributes(relation : Relation) : List

Figure 3.5: UML diagram of CAP's algorithm architecture. Due to the model architecture, each model has its own algorithm. `SpectrumModel` includes `PCAAlgo`, `PLSModel` works with `PLSAlgo` and `SimcaModel` with `SimcaAlgo`. `TunnelVisualization` is used in `SpectrumModel` as additional algorithm. `TrendAlgo` is related to the task of machine outages and not discussed here.

algorithm, it is worth to have a look at the algorithm architecture, too. Generally, every algorithm is a `FunctionAlgo`. It computes one or more functions of attribute values. When this algorithm is applied, it adds computed attributes to the given relation. The computation itself may be delayed until the values of these computed attributes are actually needed. This safes computation time since only a small fraction of values are usually needed, e.g. whilst plotting them online.

A `TrainableModeAlgo` furthermore enables to process user's selected training time (so-called mode) in order to compute the output functions. Since a user can select several modes, the output functions have to be created for each mode. All data mining methods in CAP have to be inherited from this class. The single inheritors for each algorithm are discussed in the next three sections.

Before I will explain each single model in detail in the next three sections, I will explain how a model works and how it is saved. Figure 3.6 reveals the connections between the single classes.

Whenever the user adds a new model to an existing project, several new instances are created. Every model will contain its own algorithm in order to compute the output functions for the relation. Furthermore each model will include its own persistence handler that will take care when the model has to be saved. Since each model class is different, its persistence handlers differs, too.

When a model is trained, its algorithm will include as many algorithm parameters as

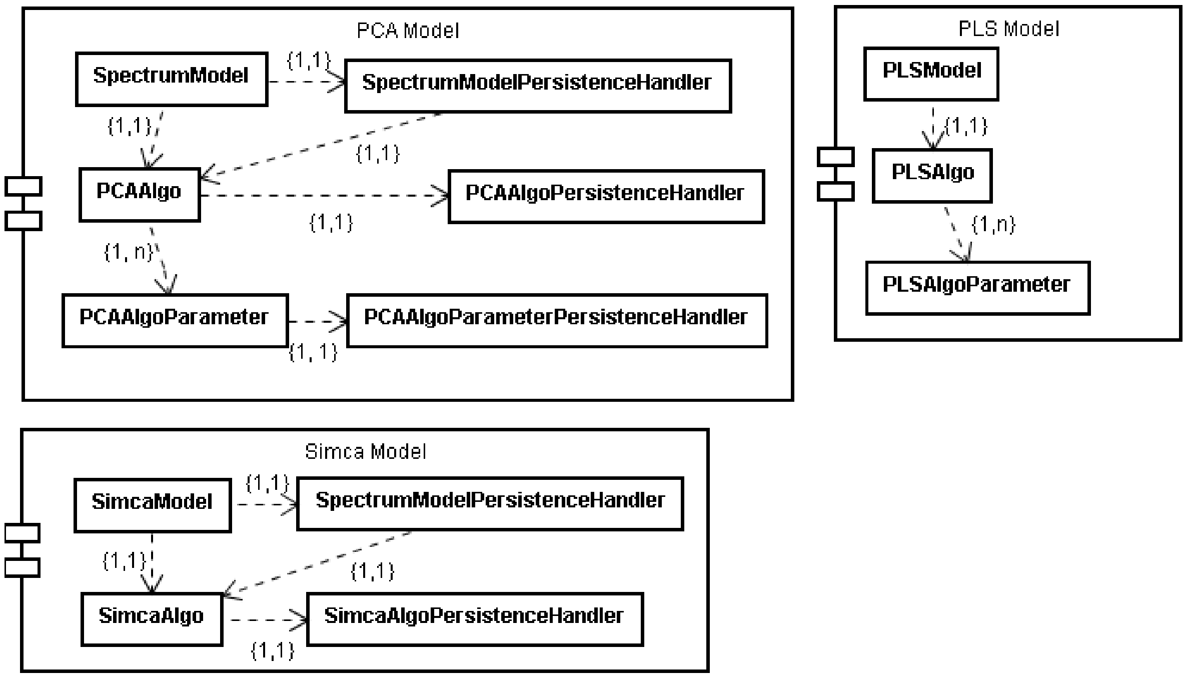


Figure 3.6: UML diagram of three model components. The used classes and their links between them are shown. When the user trains a PCA Model, a new instance of a `SpectrumModel` with its own `PCAAlgo` will be created. For each user's mode, the algorithm has to store one `PCAAlgoParameter` which contains the model information. Whenever the user wants to store a model, its persistence handlers (on the right side) are called. Simca Model does not need any parameter, whereas the PLS Model cannot be saved.

there are user modes. Each parameter will store information needed to build the output functions for the relation, e.g. transformation matrix for PCA (confirm Section 2.2.1).

If the user wants to save a project, all its models are saved separately. The saving is propagated down from the project to each single algorithm parameter. This is done in an XML tree structure, whereas the root represents the project and its children are the project's models plus additional project information. When the propagation comes down to the algorithm level, each parameter is saved as a children of its algorithm. The reading of this XML file is performed when the user loads the project again.

The actual tree extension for saving the project is done by separate persistence handlers. Hence there exist a project persistence handler, a model persistence handler and a algorithm parameter persistence handler to save and load a complete project. Note that a persistence handler is not necessary at all for any model, algorithm or project, respectively. If there is just one persistence handler missing, as it is the case for the `PLSModel`, the complete project cannot be saved and loaded.

Table 3.1: List of the DEM file header and its values. This file format which is suitable for batch processes demands the attribute `BatchID` as 2nd column.

1st column	2nd column	3rd column	...	(n+2)-th column
TimeStamp *yyyy/dd/mm hh:mm:ss*	BatchID *batch number*	*1st attribute* *value*	... ...	*n-th attribute* *value*

3.3.1 Spectrum Model

A Spectrum Model is created when the user adds a PCA Model to the project. The class `SpectrumModel` represents the PCA Model on the implementation level. Each instance contains one `PCAAlgo` to train the model and one `SpectrumModelPersistenceHandler` to load and save the model.

In `PCAAlgo` we can find the implementation of principal component analysis as it was presented in Section 2.2.1. Moreover several preprocessing methods are available for this algorithm. Every method in Section 2.1 is implemented and can be selected by the user. The data will be transformed and processed.

Moreover it is possible to toggle an additional visualization which looks like a tunnel surrounding the principal components versus time. The tunnel consists of a minimal and a maximal border. Additionally, the so-called *golden batch* is toggled as well. It can be considered as the mean value of all batches.

For definitions of tunnel and golden batch, I refer to Section 2.3 where one can also find an example produced by the Spectrum Model shown in Figure 2.1.

3.3.2 PLS Model

The PLS Model is implemented in the class `PLSModel` as shown in Figure 3.4. It inherits from the class `Model` which basically covers the approximation of functions. Using the latter model, it is possible for the user to choose between different methods for function approximation as follows:

- SVM – Support Vector Machine
- HNN – Hidden (Layer) Neural Network
- KLS – Kernel Least Squares
- GMM – Gaussian Mixture Model

The PLS Model, however, only allows Partial Least Squares as learning algorithm. Hence, it differs fundamentally from the usual model since it does not learn to approximate functions. Moreover it learns the dependency

$$f(\mathbf{X}) = \mathbf{Y} \tag{3.1}$$

whereas f is a linear function in our case, and the matrices $\mathbf{X}$ and $\mathbf{Y}$ represent the input (usually independent) and output (dependent) variables, respectively.

Therefore it is necessary for each regression model like `PLSModel` to additionally store information about every variable; whether it is a input or output variable. This distinction is needed for the construction of $\mathbf{X}$ and $\mathbf{Y}$. The implementation of PLS was already discussed in Section 2.2.2. The SIMPLS algorithm can be found in the class `PLSAlgo`

In the following section I will present another regression model for PLS. However, this model just calls already implemented functions out of a DLL, whereas `PLSModel` is transparent. Hence, the ability for interpretation of the latter is much easier to reveal.

3.3.3 Simca Model

Whenever the user wants to employ the power of the well established SIMCA-P toolbox, he or she has to create a new Simca Model. So far the Simca Model is only able to perform PLS which produces similar results compared to the model presented in the last section.

A Java Native Interface (see Section 3.5) is used to call functions out of the SIMCA DLL. Siemens Corporate Research has got a bunch of SIMCA-P files in order to adopt CAP to SIMCA-P. So far the customer just demanded the loading of a given SIMCA-P model into CAP.

Hence, creating a Simca Model in Java does not mean that a new SIMCA-P model is created. Moreover, the user can load a SIMCA-P model out of a USP (Umetrics SIMCA Project) file. The loaded model is then stored into a Simca Model. Simca Model's algorithm is a `SimcaAlgo` which does not need to be trained since it only loads the PLS information that is needed for the model. Some of its functions are listed in Figure 3.5.

Every USP file is equivalent to a project in CAP. First of all, it (maybe) contains the database that is used for learning.[2] Second, every learned model that belongs to this project can be found in this file.

To reference a project in Simca-P, there exists a unique key called `projHandle` which identifies each project. Thus, each instance of `SpectrumModel` has to store this handle in order to read out a Simca-P project's model. Moreover, every USP's model is indexed by a so-called `modelNumber` which is of course unique as well. Hence, this integer is also stored in an `SpectrumModel`'s instance. A brief overview of the model's data structures is shown in Figure 3.4.

[2]In Umetrics Simca-P, there are two different kinds of project files. The standard USP file contains the raw data which is usually the case. The smaller RUSP (reduced USP) file does only include the learned models.

3.4 Graphical User Interfaces

This section will give a brief overview of some graphical user interfaces (GUIs) which I developed in order to provide a usable link between the program's user and its logic behind it. Since CAP was supposed to be a graphical platform to facilitate the usage of data mining algorithms, there are many more GUIs that are not discussed in this report.

CAP was programed in Java (see http://java.sun.com for further details). Java itself provides lots of potent libraries for development of GUI. Most of the GUIs, however, are based on CAP's own GUI classes and structures. This is due to the fact that some visualization concepts are too special to be found or realized by Java packages.

First of all I want to talk about one example of CAP's own GUI structure in Section 3.4.1. Then I will outline some of CAP's plotting techniques for univariate sensors in Section 3.4.2 and for spectral values Section 3.4.3, respectively. At the end of this section I will shortly explain some concepts of the Simca GUI in Section 3.4.4.

3.4.1 The Architecture of GUI Actions

In this section I want to present one GUI example in more detail. Since the action's hierarchy is quite easy to understand and most of the classes were developed by myself, I will use these concepts to explain one of CAP's GUIs more precisely.

Figure 3.7 gives a partly overview of the class hierarchy of actions. Every non abstract class (indicated by non italic font of the class name) was developed by me whilst my stay at SCR. Basically all actions are inheritances of `AbstractAction` to be consistent to the Java hierarchy. There exists a direct descent of this class named `GUIAction` to distinguish between actions for GUI purpose and actions which are not used for GUIs.

Abstract classes like `ModelAction` or `ProjectAction`, that are direct descents of `GUIAction`, are used to differentiate stronger between every action. Of course there exist some non abstract descents, e.g. like `NewSimcaPlotAction`, which cannot be pressed into the predefined scheme of classes. An exemplary implementation of a `ProjectAction` is the class `CreateSimcaModelAction` which will effect the project since a new Simca Model will be added to it whenever this action is called.

If model features should be changed by calling functions, a `ModelAction` has to be designed. For an example, let us have a closer look at the `SpectrumModel` that uses `PCAAlgo` as learning algorithm. The latter object contains boolean features, e.g. like `usingMSC`, `usingSGF` and `centered`, respectively. All these preprocessing methods were explained in detail in Section 2.1. So, if the user wants to toggle these features on or off, there has to be one `PCAAlgoToggleAction` for each feature.[3] Thus one can find the classes `ApplyMSCAction`, `ApplySGFAction` and `ApplyCenteringAction`, respectively.

[3]Note that the class `PCAAlgoToggleAction` is an abstract descent from `ModelToggleAction` which is implemented if a feature of a model is boolean. The latter one's super class has to be `ModelAction` per definition.

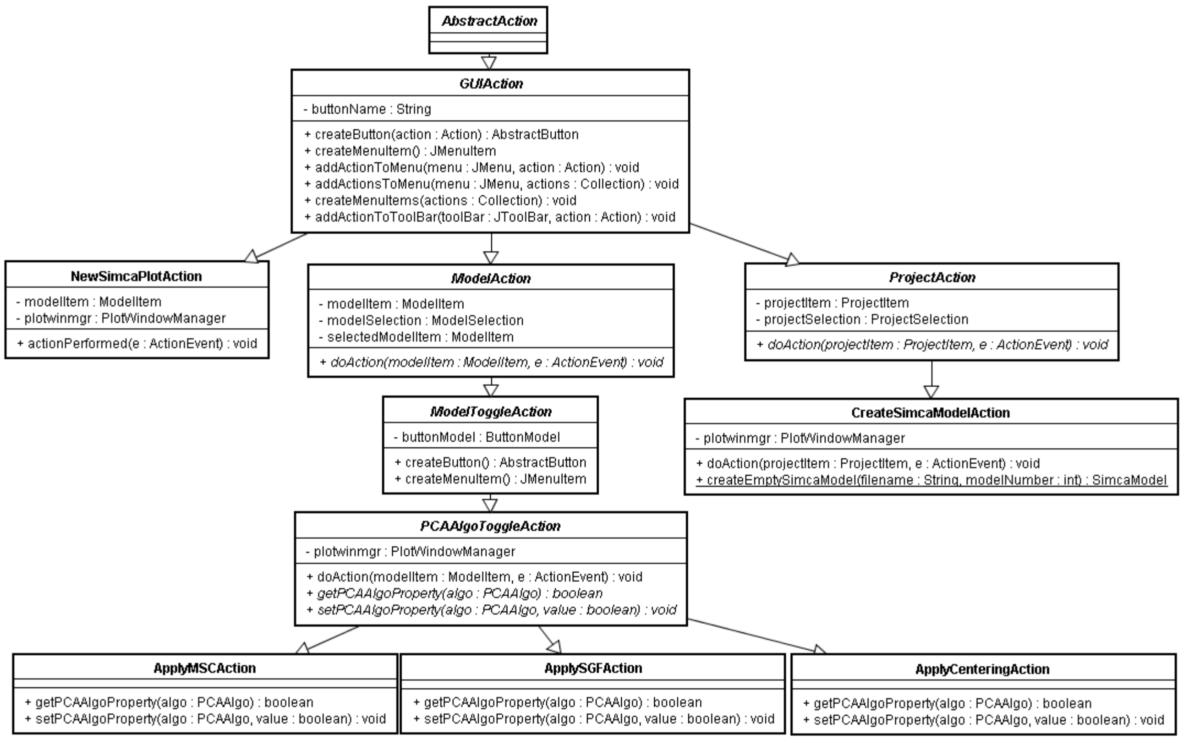

AbstractAction
GUIAction
- buttonName : String
+ createButton(action : Action) : AbstractButton
+ createMenuItem() : JMenuItem
+ addActionToMenu(menu : JMenu, action : Action) : void
+ addActionsToMenu(menu : JMenu, actions : Collection) : void
+ createMenuItems(actions : Collection) : void
+ addActionToToolBar(toolBar : JToolBar, action : Action) : void
NewSimcaPlotAction
- modelItem : ModelItem
- plotwinmgr : PlotWindowManager
+ actionPerformed(e : ActionEvent) : void
ModelAction
- modelItem : ModelItem
- modelSelection : ModelSelection
- selectedModelItem : ModelItem
+ doAction(modelItem : ModelItem, e : ActionEvent) : void
ProjectAction
- projectItem : ProjectItem
- projectSelection : ProjectSelection
+ doAction(projectItem : ProjectItem, e : ActionEvent) : void
ModelToggleAction
- buttonModel : ButtonModel
+ createButton() : AbstractButton
+ createMenuItem() : JMenuItem
CreateSimcaModelAction
- plotwinmgr : PlotWindowManager
+ doAction(projectItem : ProjectItem, e : ActionEvent) : void
+ createEmptySimcaModel(filename : String, modelNumber : int) : SimcaModel
PCAAlgoToggleAction
- plotwinmgr : PlotWindowManager
+ doAction(modelItem : ModelItem, e : ActionEvent) : void
+ getPCAAlgoProperty(algo : PCAAlgo) : boolean
+ setPCAAlgoProperty(algo : PCAAlgo, value : boolean) : void
ApplyMSCAction
+ getPCAAlgoProperty(algo : PCAAlgo) : boolean
+ setPCAAlgoProperty(algo : PCAAlgo, value : boolean) : void
ApplySGFAction
+ getPCAAlgoProperty(algo : PCAAlgo) : boolean
+ setPCAAlgoProperty(algo : PCAAlgo, value : boolean) : void
ApplyCenteringAction
+ getPCAAlgoProperty(algo : PCAAlgo) : boolean
+ setPCAAlgoProperty(algo : PCAAlgo, value : boolean) : void

Figure 3.7: UML diagram of Action classes in CAP. Every class but `AbstractAction` was developed by SCR. A `ModelAction` or `ProjectAction` is executed whenever a model or project is created, changed or deleted by the GUI. Boolean model features, e.g. MSC for PCA, can be toggled on or off. If a Simca Model is created, `CreateSimcaModelAction` is called.

3.4.2 Plotting Univariate Sensors

In this section I want to reveal how one-dimensional sensor values are plotted in *CAP*. Note that the presented formalism is not based on my work at SCR. As I already outlined in Section 1.3, the flexible plotting system *PowerPlot* was already developed before *CAP*. In order to use its plotting routines, CAP implements a bunch of *PowerPlot*'s classes.

In time series analysis, it is very important to visualize the data by appropriate plotting routines. The usual sensor plot in CAP for state estimation is a line chart with deadband interpolation.[4] I want to underline again that state estimation is usually not performed if the *PATClient* of *CAP* is used. The line-chart, however, is utilized for PLS when a non spectral variable shall be plotted.[5] Whereas a deadband is usually drawn between two time stamps, it is also possible to fit a line between them. This is called linear interpolation.

[4]Note that a state estimation is only performed if the user applies an State Estimation Model. This model is realized by the class `Model`.

[5]Since spectral variables are the only multivariate sensors in PAT applications, non spectral variables have to be univariate.

One example of such a deadband line-chart is shown in Figure 3.8. Here one batch of the Paracetamol dataset is shown for the variable `pH-value`. At the top there will be the name of the sensor which is read out from the dataset. The time ranges after applying all filters is shown as green bar under the name. Without applying any filter, the complete time range will be taken for training. If there are several disjoint time ranges, some time stamps might not be taken. This will be the case if they will not be in the filtered time ranges. This might be used to reduce outliers or similar problems.

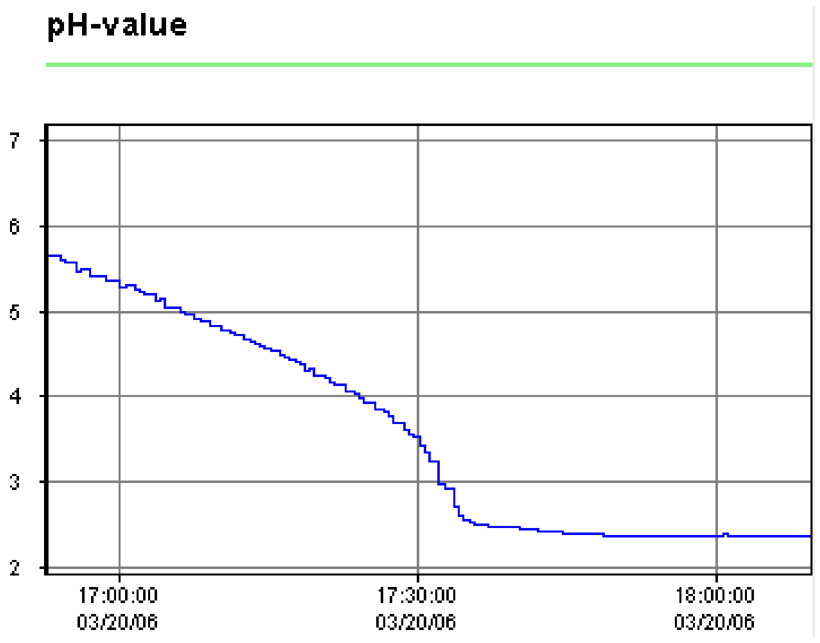


Figure 3.8: Univariate sensor via time plot of the variable `pH-value` which belongs to the Paracetamol dataset. This line-chart is drawn by deadband interpolation. The green bar above the plot represents the time ranges after filtering which will be taken for learning (here whole range selected).

3.4.3 Plotting Spectral Values

In the previous section I explained how univariate sensors are plotted. The set of sensors for PAT applications is usually huge and the biggest part of the data is spectral. If a spectrum is recorded via time, there will be hundreds of frequency variables since the spectrum is nearly continuous. In our case the spectrum is only taken in the near infrared (NIR).

Thus the multivariate variable *NIR data* consists of hundreds of different frequencies. Each carries one intensity value per time stamp. To distinguish between spectral data and other data, there has to be a name convention. Non spectral sensor names will be a usual string which cannot be parsed into a number. However, a frequency of spectral data will be referenced by its wavelength which has to be a unique positive number. Hence it is quite easy to separate spectral data from non spectral ones.

The separation is needed in order to have the standard line-charts (see Section 3.4.2) for non spectral values and one specialized plot for the spectrum. This plot is realized by the class `DrawableSpectrum` as shown in Figure 3.9. The difference between a normal line-chart and this class are the following attributes and methods:

- `grayScale` is true if a gray scale picture shall be shown instead of a colored one (standard is false)
- `numLines` represents the number of spectral lines that will be shown (equivalent to the picture height)
- `grayScaleJetColors` stores all colors needed to plot a gray scale picture if `grayScale` is true
- `coloredJetColors` stores all colors for a false-color picture if `grayScale` is false
- `createJetColors()` is computed once when CAP starts
- `drawSpectrum()` maps every data point to a color and plots the whole spectrum (this method is used in `draw()`)

Since there are only 64 colors (no matter if `grayScale` is true or false), every intensity is mapped to the numbers from 0 to 63. These numbers are used as array index in order to access `grayScaleJetColors` and `coloredJetColors`, respectively. An example false-color picture is shown in Figure 3.10. The ordinate represents the wavelength. Note that due to computation time, the plotting of the spectral lines is subsampled with Algorithm 3. Moreover the subsampling will show a clear false-color picture if there are too many spectral dimensions, e.g. more than the height of the monitor in pixels.

Algorithm 3 Compute plotting step width for false-color picture

```
Require: number of spectral dimensions $m$, number of maximal plotted lines $m_{\max}$
Ensure: step width between two plotted spectral lines $s$
1: $s \leftarrow m/m_{\max}$                 // plot every $s$-th line
2: if $s = 0$ then                           // $m < m_{\max}$
3:     $s \leftarrow 1$                      // plot all spectral lines
4: end if
5: return $s$
```

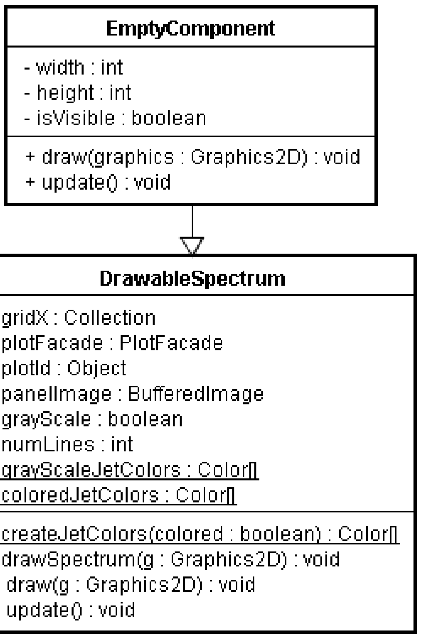

EmptyComponent
- width : int
- height : int
- isVisible : boolean
+ draw(graphics : Graphics2D) : void
+ update() : void
DrawableSpectrum
- gridX : Collection
- plotFacade : PlotFacade
- plotId : Object
- panelImage : BufferedImage
- grayScale : boolean
- numLines : int
- grayScaleJetColors : Color[]
- coloredJetColors : Color[]
- createJetColors(colored : boolean) : Color[]
- drawSpectrum(g : Graphics2D) : void
+ draw(g : Graphics2D) : void
+ update() : void

Figure 3.9: UML diagram of the class `DrawableSpectrum`. The classes `SpectrumModel`, `PLSModel` and `SimcaModel` have to use this class to plot the spectral values. `DrawableSpectrum` is a sub class of `EmptyComponent`.

3.4.4 The SIMCA GUI

In this section I will shortly outline the GUI for the Simca Model. If the user wants to create an new Simca Model, he or she has to load an USP file. For more insights into the Spectrum Model, I refer to Section 3.3.3.

A dialog will pop up and guide the user through the loading process. The UML diagram of this dialog named `CreateSimcaModelDialog` is shown in Figure 3.11. First of all, a USP file has to be loaded. After successful loading, the user has to select a model out of the list of the USP's models. Then the information of the selected model will be transfered into a Java instance of `SimcaModel`.

3.5 Java Native Interfaces

In this section I want to present the hardware interface I designed at Siemens Corporate Research with the help of the Java Native Interface. For a detailed description of JNI with helpful programming examples, I refer to the book of Liang [9].

Before I will talk about the certain implementations for the Simca DLL and the OPUS DLL, respectively, I want to introduce the basic concept of JNI. The Java Native Interface is useful if one is interested in the following aspects like the following:

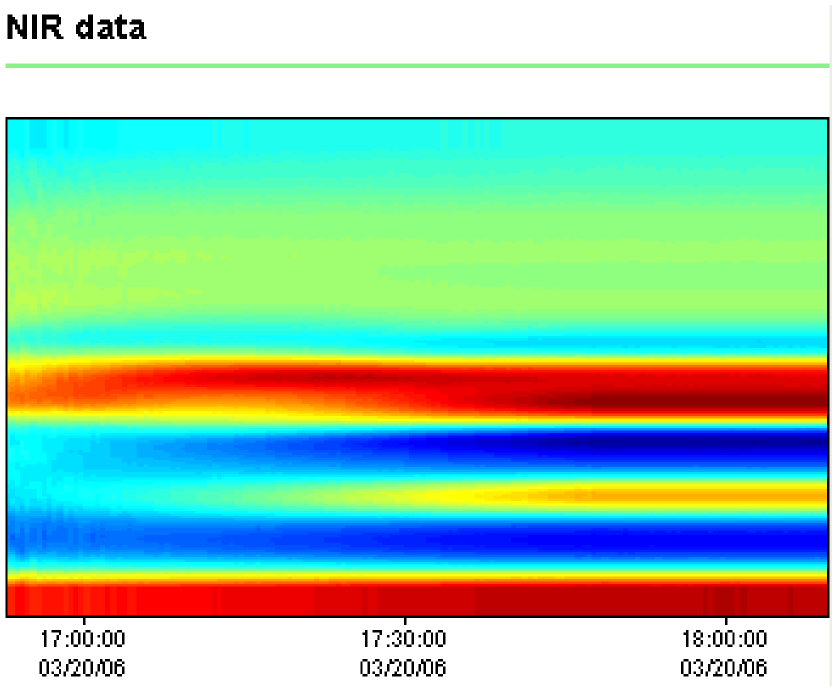


Figure 3.10: False-color picture for spectral data of one Paracetamol batch. The axis of ordinates describes the wavelength. The latter is subsampled to reduce computation time.

- integrating a Java application with legacy code written in languages such as C or C++
- incorporating a Java virtual machine implementation into an existing application written in languages such as C or C++
- implementing a Java virtual machine
- understanding the technical issues in language interoperability, in particular how to handle features such as garbage collection and multi-threading

Since OPUS and Simca are libraries in C and shall be called in Java, the JNI was mandatory for this task. I was using the JNI to write native methods that allow CAP to call functions implemented in these native libraries. CAP calls native methods in the same way that it calls methods implemented in the Java programming language. Behind the scenes, however, native methods are implemented in another language, here C, and reside in native libraries.

As a design convention, the JNI implementation between the Java application and the C DLLs will consist of one Java class with native functions and a wrapper DLL in C that converts Java data structures into C ones in order to call the legacy code of the C DLLs. In the following sections I will first talk about the native libraries OPUS and Simca. Then I will give one example for each JNI implementation.

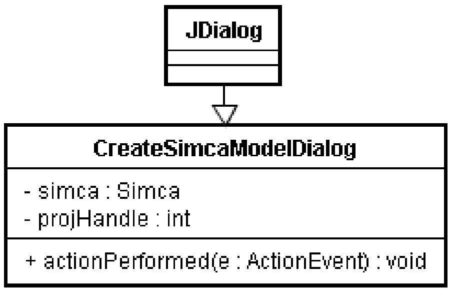


Figure 3.11: UML diagram of the class `CreateSimcaModelDialog` which is a sub class of `javax.swing.JDialog`. If a Simca Model shall be created, this dialog will appear to first load a USP file, then store its information into a new Simca Model.

3.5.1 The Simca-P Toolbox

The Simca-P toolbox consists of a couple of DLLs which were optimized to solve PAT problems. Umetrics developed its own powerful tool that is able to realize much more than CAP. Whereas PCA with preprocessing and PLS is the only thing implemented in CAP for PAT, there are many more features, filtering methods and algorithms in Simca-P. Siemens Corporate Research was asked to integrate this library into their own framework.

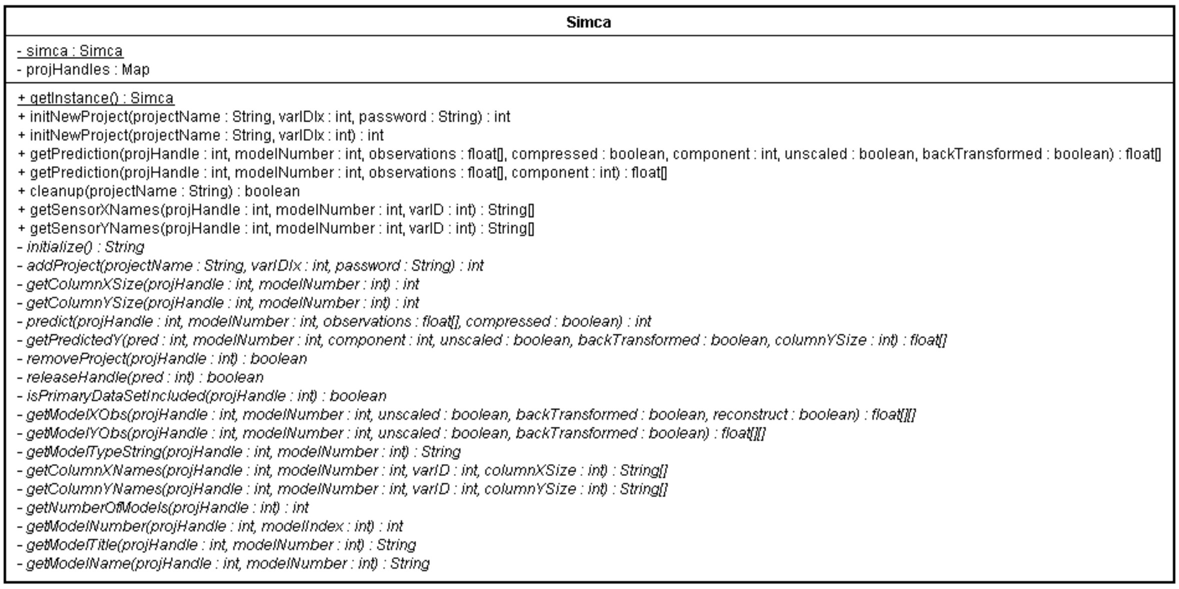
It would burst the frame of this report if I talked about the details of the whole toolbox. Thus I would like to mention some features that are implemented in Simca-P. Of course, PCA and PLS are included as well. However, there exists even cross correlation of variables, the more and more popular orthogonal signal correction (OSC), wavelet transformation and dozens of useful plotting methods to visualize data.

3.5.2 A JNI Implementation for Simca

In this section I will present the native methods I designed to call some Simca-P functions from its DLL. Note that due to the huge amount of Simca-P function and the need just to load and apply a USP model, the implementation is rather short than exhaustive. The list of native functions that where needed to store and apply a Simca-P model can be found in Figure 3.12.

In order to call the relevant C functions from Simca-P in Java, there has to be a wrapper class in C that converts the each Java variable into the appropriate C variable. After converting, the native C function from the original Simca DLL can be called. Its result is reconverted and returned into a Java variable after the used memory is freed.

As an example, I want to present one native function in Java and its counterpart of the `simcawrapper.dll`. The function `getColumnXSize()` shall return the number of columns from the input matrix $\mathbf{X}$ of a certain model. The native code in Java is pretty easy to read since it just consists of one line. This single code line plus its JavaDoc



```
Simca
- simca : Simca
- projHandles : Map
+ getInstance() : Simca
+ initNewProject(projectName : String, varIDIx : int, password : String) : int
+ initNewProject(projectName : String, varIDIx : int) : int
+ getPrediction(projHandle : int, modelNumber : int, observations : float[], compressed : boolean, component : int, unscaled : boolean, backTransformed : boolean) : float[]
+ getPrediction(projHandle : int, modelNumber : int, observations : float[], component : int) : float[]
+ cleanup(projectName : String) : boolean
+ getSensorXNames(projHandle : int, modelNumber : int, varID : int) : String[]
+ getSensorYNames(projHandle : int, modelNumber : int, varID : int) : String[]
- initialize() : String
- addProject(projectName : String, varIDIx : int, password : String) : int
- getColumnXSize(projHandle : int, modelNumber : int) : int
- getColumnYSize(projHandle : int, modelNumber : int) : int
- predict(projHandle : int, modelNumber : int, observations : float[], compressed : boolean) : int
- getPredictedY(pred : int, modelNumber : int, component : int, unscaled : boolean, backTransformed : boolean, columnYSize : int) : float[]
- removeProject(projHandle : int) : boolean
- releaseHandle(pred : int) : boolean
- isPrimaryDataSetIncluded(projHandle : int) : boolean
- getModelXObs(projHandle : int, modelNumber : int, unscaled : boolean, backTransformed : boolean, reconstruct : boolean) : float[][]
- getModelYObs(projHandle : int, modelNumber : int, unscaled : boolean, backTransformed : boolean) : float[][]
- getModelTypeString(projHandle : int, modelNumber : int) : String
- getColumnXNames(projHandle : int, modelNumber : int, varID : int, columnXSize : int) : String[]
- getColumnYNames(projHandle : int, modelNumber : int, varID : int, columnYSize : int) : String[]
- getNumberOfModels(projHandle : int) : int
- getModelNumber(projHandle : int, modelIndex : int) : int
- getModelTitle(projHandle : int, modelNumber : int) : String
- getModelName(projHandle : int, modelNumber : int) : String
```

Figure 3.12: UML diagram of the JNI class `Simca`. This JNI loads the wrapper library `simcawrapper.dll` which calls Simca-P functions from its DLL. There has to be one native method for every function that is called from Simca-P.

comment can be found as follows in `com.siemens.scr.simca.Simca`:

```
/**
 * Retrieves the number of X columns in a specific model.
 *
 * @param projHandle
 *            the project handle to get the column from
 * @param modelNumber
 *            the model number to use
 * @return the number of X columns in the model
 */
private native int getColumnXSize(int projHandle, int modelNumber);
```

On the other hand, the code in C implements the latter native function by converting its parameters into C, calling the appropriate Simca DLL function and finally returning the result. The code for it looks like the following:

```
JNIEXPORT jint JNICALL Java_com_siemens_scr_simca_Simca_getColumnXSize
  (JNIEnv *env, jobject obj, jint projHandle, jint modelNumber)
{
  /* set the project handle to the pointer */
  SQX_ProjectHandle *pProjHandle = (SQX_ProjectHandle *) projHandle;
  int iModelNumber = (int) modelNumber;
  int iColumnXSize = 0;

  (*pSQX_GetColumnXSize)(*pProjHandle, iModelNumber, &iColumnXSize);

  return iColumnXSize;
}
```

Note that the data type `SQX_ProjectHandle` is declared in Simca, however, basically it stores one integer to index the loaded projects. The legacy function `SQX_GetColumnXSize()` is called by its pointer `*pSQX_GetColumnXSize` in order to be independent from the location of the Simca DLLs.

Once understood, it is quite easy to design a JNI and to call C functions in Java, respectively. Thus, future tasks, such as learning a new USP model, can be easily realized by extending the Java class `Simca` and `simcawrapper.dll`.

3.5.3 The Spectroscopic Software OPUS

Whereas Simca-P was designed to solve PAT problems by machine learning techniques, OPUS' purpose is to read out online data from a spectrometer. Hence, the data acquisition can be handled with OPUS. Since it was developed in C like Simca, my task was to create a JNI for OPUS that works similar to Simca. Details of the JNI implementation can be found in the next section.

OPUS is very easy to apply. Basically, there exist just 5 different functions to acquire spectral data as they are listed below:

- `InitInstrument(LPCTSTR pConfigFile, LPTSTR pResult, UINT uiBufferLen)`
- `BackgroundScan(LPCTSTR pXPM, LPTSTR pResult, UNIT uiBufferLen)`
- `SampleScan(LPCTSTR pXPM, LPTSTR pResult, UNIT uiBufferLen)`
- `GetAcquiredData(LPTSTR pResult, UNIT uiBufferLen)`
- `DirectCommandCall(LPCTSTR pCommand, LPTSTR pResult, UNIT uiBufferLen)`

Each function takes a parameter `pResult` which is a pointer to a buffer that will receive the result of the function and a parameter `uiBufferLen` that holds the length of the buffer that receives the result. Note that its unsigned integer value has to be big enough to store the result into the buffer. Otherwise information might be lost whilst spectral data acquisition. Usually it is set to a multiplier of 1024 to have enough buffer space. Every function will directly return the length of the result. The result usually is `"OK"`, an error message or the measured data written into `pResult`.

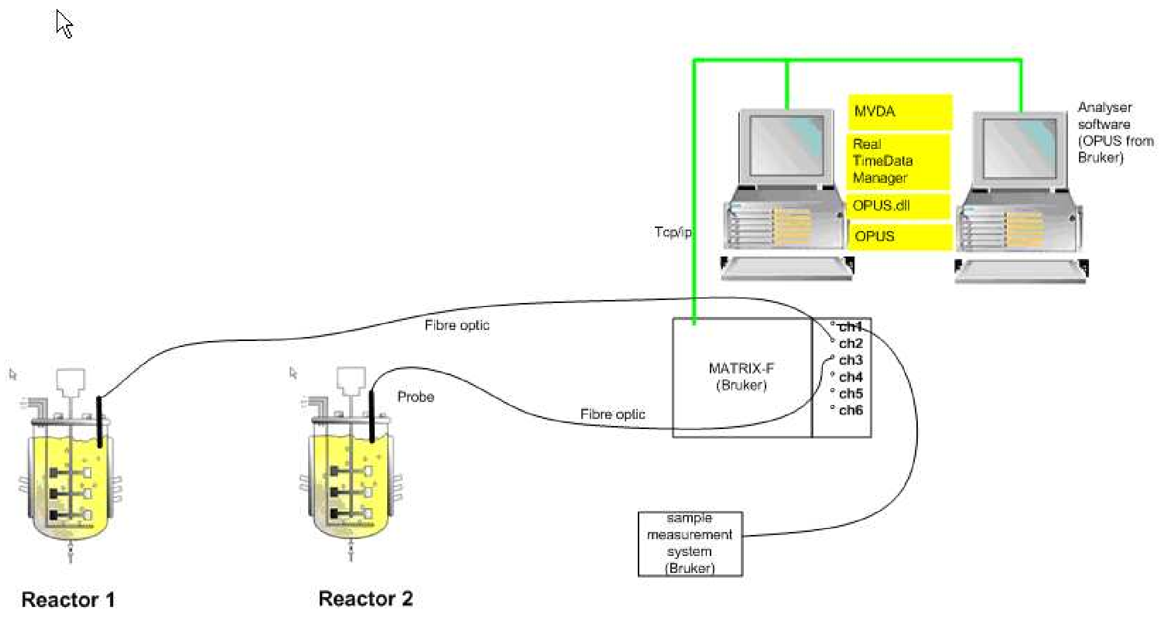


Figure 3.13: Experimental setup for data acquisition with OPUS. Spectrometric probes in bioreactors send data via fiber optics by the software MATRIX-F. The analyzer on a PC receives data by TCP/IP.

The function `InitInstrument()` connects to a measurement instrument, e.g. a spectrometer. The instrument itself is defined by an NTI file found in the same directory as the application. The name is passed in the variable `pConfigFile`. One has to call the function once when the application is initializing.

`BackgroundScan()` executes a background measurement. Certain parameters are defined by an OPUS experiment file whose name is passed in the variable `pXPM`. The function `SampleScan()` executes a sample measurement. Again, its parameters will be passed in the same variable `pXPM`.

The function `GetAcquiredData()` receives the data collected by the last `SampleScan()` command. Usually if no error occurred, the wavelengths and their intensities can be read out from the buffer `pResult`. The last function `DirectCommandCall()` allows the execution of any Opus client/server command. Since we will not use these commands, I will only mention this function without any further insights.

The hardware setup is shown in Figure 3.13. Different bioreactors are connected by fiber optics to the software interface MATRIX-F that acts as a server. The latter system sends received spectral data to personal computers with analyzer software (so-called clients) via TCP/IP. If a client requests a certain command, e.g. a sample scan, MATRIX-F will send the result in a character buffer back to the client.

I would like to underline that whilst my internship at SCR there was only the possibility to connect to a simulator that is not able to send continuous online data. However, SCR ordered an experimental setup in order to develop the OPUS interface for CAP.

3.5.4 The JNI for OPUS

As I already explained for Simca-P in Section 3.5.2, a JNI is needed whenever one wants to call C function in Java and vice versa. Hence I was supposed to develop a native Java class called `OPUS.java` and a wrapper DLL as interface between Java and Opus DLL. Its file name is `opuswrapper.dll`.

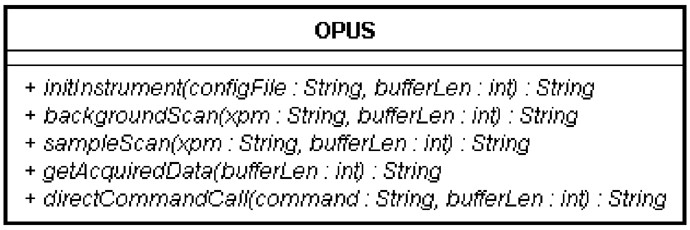

OPUS
+ initInstrument(configFile : String, bufferLen : int) : String
+ backgroundScan(xpm : String, bufferLen : int) : String
+ sampleScan(xpm : String, bufferLen : int) : String
+ getAcquiredData(bufferLen : int) : String
+ directCommandCall(command : String, bufferLen : int) : String

Figure 3.14: UML diagram of the native Java class `OPUS`. Every function that is available from the OPUS DLL is listed in this interface. The implementation can be found in `opuswrapper.dll`.

The native Java functions can be seen in Figure 3.14. Their implementations are located in the wrapper DLL for OPUS. As an example I want to present the wrapper function for initializing a spectrometer which is located in `opuswrapper.dll` as well:

```
JNIEXPORT jstring JNICALL Java_com_siemens_scr_opus_OPUS_initInstrument
  (JNIEnv *env, jobject obj, jstring configFile, jint bufferLen)
{
  /* define necessary variables */
  UINT uiBufferLen = (UINT) bufferLen;
  LPTSTR pResult = (LPTSTR) malloc(sizeof(char) * uiBufferLen);

  /* convert every jstring into a LPCTSTR */
  LPCTSTR pConfigFile = env->GetStringUTFChars(configFile, NULL);
  if (pConfigFile == NULL) {
    return NULL; /* OutOfMemoryError already thrown */
  }

  /* call the appropriate C function */
  InitInstrument(pConfigFile, pResult, uiBufferLen);

  /* release used memory */
  env->ReleaseStringUTFChars(configFile, pConfigFile);
  /* return a new jstring out of the LPTSTR pResult */
  return env->NewStringUTF(pResult);
}
```

Since the parameter lists of all functions are similar and their returned values are always identically (`pResult`), the other functions pretty much look the same as the one I have presented before. Finally, I want to emphasize that another intern of our group, Peng Ren, will continue my work on OPUS, such that the basics I have designed can be used, e.g. to create a GUI for PAT.

Chapter 4

Application at SCR

In this chapter I will discuss two real world applications that I experienced at SCR. Since both cases deal with NIR spectrometry, I will first shortly introduce the basic techniques that are needed to collect data sets. Then I will describe the objectives for both applications. Furthermore I will show some results of the algorithms I implemented.

4.1 Fundamentals of NIR spectroscopy

IR spectroscopy in general studies the composition of organic compounds. Furthermore one tries to find out a compound's structure and its composition based on the percentage transmittance of IR radiation through a sample. Different frequencies are absorbed by different stretches and bends in the molecular bonds occurring inside the sample.

Infrared radiation has wavelengths between approximately $750nm$ and $1mm$. Near infrared spectroscopy is a special field of infrared radiation spectroscopy. It deals with the NIR region of the electromagnetic spectrum ($750nm$ – $2600nm$).

NIR spectroscopy catches overtones and harmonic vibrations. These oscillations are caused by chemical bonds which have specific frequencies at which they vibrate corresponding to energy levels. The resonant frequencies are determined by several parameters:

- the shape of the molecular potential energy surfaces
- the masses of the atoms
- the associated vibrionic coupling

Thus, the frequency of the vibrations can be associated with a particular bond type. Simple molecules which consist of two atoms have only one bond that may stretch. More complex molecules may have many bonds. Hence vibrations can be conjugated which leads to infrared absorptions at characteristic frequencies that may be related to chemical groups.

Usually it is difficult to assign specific features to specific chemical components. Multivariate calibration techniques (e.g., principal components analysis or partial least squares) are often used to extract the chemical information. Especially in Chemometrics we can find a wide range of applications that try to analyze the collected spectrum.

4.1.1 Spectroscopic Data Acquisition

To apply MVDA methods, one has to have a device for the chemical process that should be analyzed. This device is usually called bioreactor. It can have multiple supply inputs (e.g., for the raw materials) and several outputs (the product itself, waste gas, etc). Furthermore we can attach any type of sensor that is needed to collect data. A mass spectrometer maybe analyses the waste gas. A NIR spectrometer itself is focused on the interior of the bioreactor where the chemical reaction goes on.

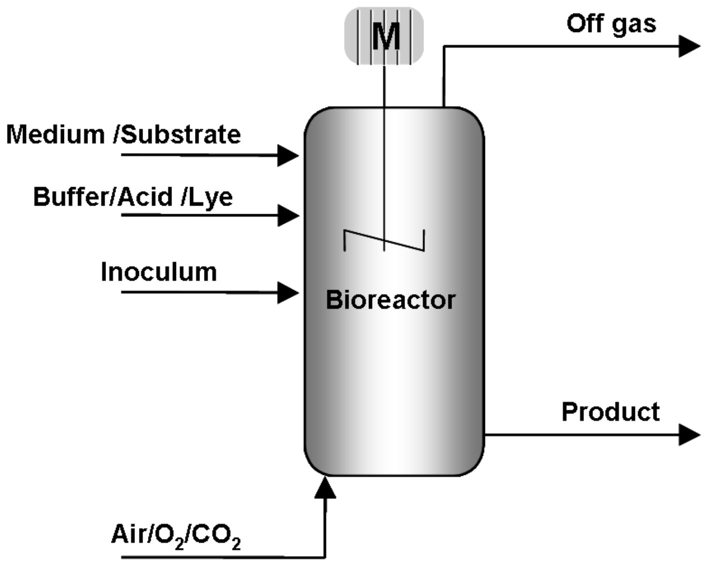


Figure 4.1: Schematic representation of a bioreactor for mass and NIR spectrometry.

In Figure 4.1 one can see the schematic representation of such a bioreactor. Several inputs that are needed to produce the desired chemical matter are stirred inside a device with the help of an engine. It heavily depends on the process which kind of components are used. Figure 4.2 shows a real world bioreactor with lots of tubes and measurement devices.

Figure 4.3 shows standard spectrometry devices that are necessary to acquire data. These electronical gadgets are absolutely essential for MVDA methods. After we know how data acquisition works, we can concentrate on the basis of PAT.

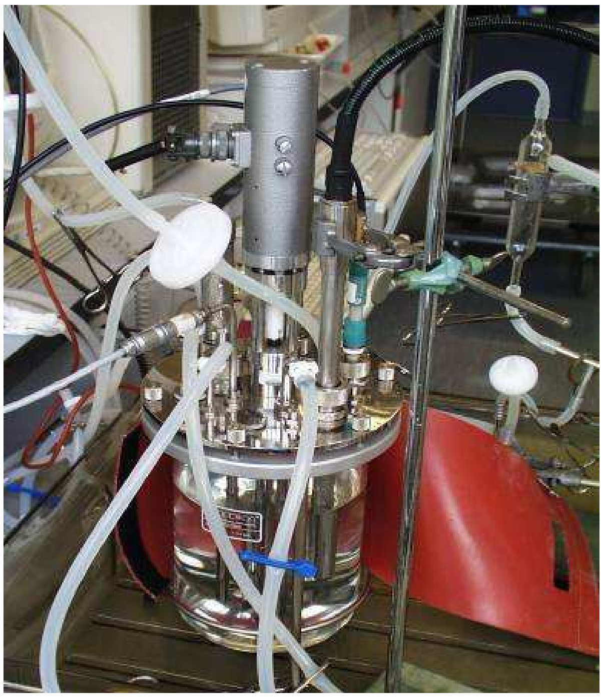


Figure 4.2: One typical bioreactor that is used for spectrometry.

4.2 Process Analytical Technology

In [6] the U.S. Food and Drug Administration (www.fda.gov) published guidelines for PAT. Here, I outline some important issues briefly. The goal of PAT is to understand and control the manufacturing process. PAT is system for designing, analyzing, and controlling manufacturing through timely measurements (i.e., during processing) of critical quality and performance attributes of raw and in-process materials and processes with the goal of ensuring final product quality.

A desired goal of the PAT framework is to design and develop processes that can consistently ensure a predefined quality at the end of the manufacturing process. Gains in quality, safety and/or efficiency will vary depending on the product.

Figure 4.4 shows the PAT arrangements that are used in our case. The bioreactor is fed with all materials that are needed to produced the desired output. Physical process parameters e.g., temperature, pressure or pH-value, are controlled directly.

The parameters that are required by PAT are treated differently. First they are acquired via spectroscopy, e.g., then all values are monitored online. An advanced process control finally decides how to adjust the physical parameters to drive the process inside the bioreactor.

Some of the PAT parameters can be found in the process output as well. Therefore according data are transformed into so-called quality data that are representative for the process.

(a) mass spectrometer (b) NIR spectrometer (c) NIR probe

Figure 4.3: Spectrometry devices. Figure (a) shows the mass spectrometer that analyzes the composition of the waste gas coming out of the reactor. Figure (b) displays the NIR spectrometer that is connected to the interior of the reactor itself via a probe that is shown in Figure (c).

The system CAP which is developed by SCR should monitor and control the process online. In order to do this, one has to have a PC next to the breadboard construction. The computer will monitor the given process information and will decide how to drive the physical process parameters. This is done by applying a learned model. Thus, an implementation of PAT like PLS, PCA or filtering was needed for CAP.

4.3 Two Real-World Examples

I am going to discuss two real-world applications that I experienced during my time in Princeton. In 4.3.1, the given data sets were used for PLS regression. The more complex application is discussed in 4.3.2.

In [22] we can find the six main steps to calibrate a multivariate system:

1. Specification of the analytes with concentration ranges, selection of spectroscopic method and wavelength range
2. Selection of a representative calibration set (training set)
3. Measurement of spectra ($\mathbf{X}$) - the analyte concentrations are measured with the reference method ($\mathbf{Y}$)
4. Evaluation of raw data (outliers) and preprocessing (filtering and compression)

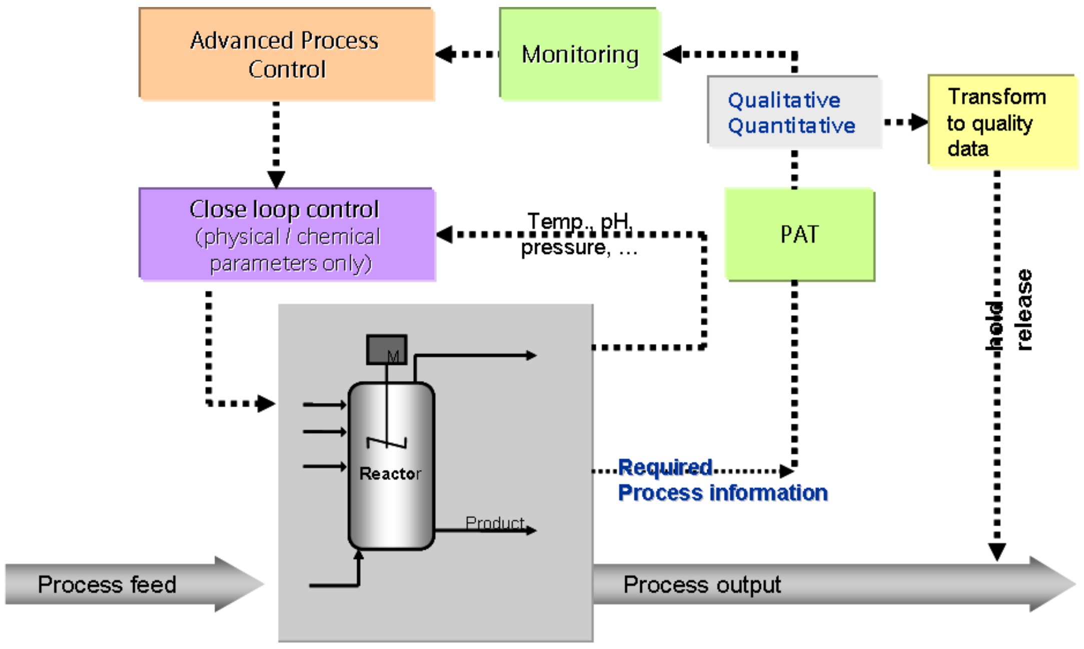


Figure 4.4: Infrastructure of PAT.

5. Calculation of the calibration model (review of fit, interpretation)

6. Validation/prediction of analyte concentrations in new samples (prediction set)

I want to introduce these steps since they reveal the current status of an application. At SCR, we never dealt with the first three steps. Sometimes even the model out of step 5 was given.

4.3.1 Binary Powder Blending

Suppose that you want to mix two different powders. One has a finer granulation ($< 200\mu m$) than the other ($> 300\mu m$). One shall develop a methodology for the quantitative analysis of binary powder mixtures for in-line process monitoring with the aim of end-point detection.

A calibrated system shall decide when to stop mixing after both powders reached a certain degree of blending. Figure 4.5 shows the setup of the binary powder blending application. A vertical cone mixer was used to blend the powders. The mixer contains an orbiting screw that creates a convection mixing mechanism. The NIR spectra were recorded using fiber optics.

Out of the given spectrum, a PLS model shall predict the degree of mixture which is a value between 0 and 100%. What we have got from the customer was the matrix **X** containing the spectral values and a ready-made model for prediction of **Y**. My task was to load the model into CAP to apply its prediction method.

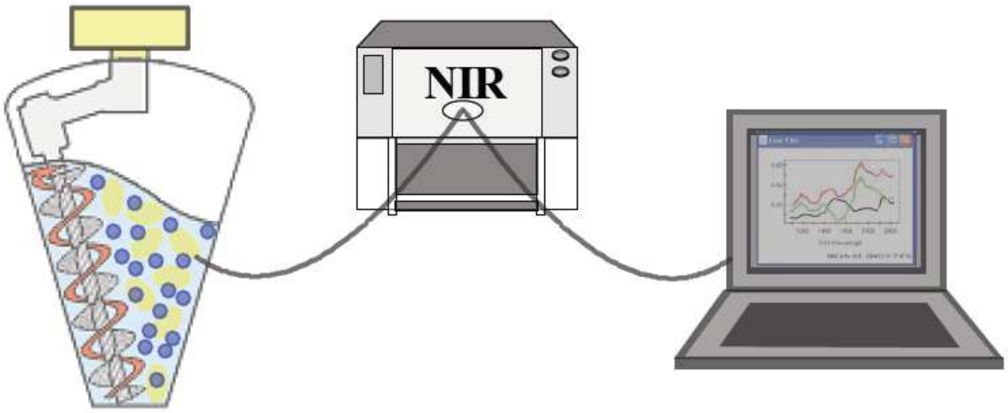


Figure 4.5: Breadboard construction of the binary powder mixture.

Since there is only one dimension for **Y**, this problem is univariate and hence quite easy to tackle. Furthermore the calibration steps 1 (selection of wavelength range) to 5 (calculation of calibration model) were already done. Naturally, these circumstances make it very easy.

As the license for the given model expired after I left SCR, there is no way to show any results. This is not a disadvantage at all, since it is not very challenging to load and apply an already calibrated model. It is much more interesting to use an own approach.

Thus I implemented my PLS model for CAP to verify if the given one produces a correct result. The method I have used is explained in detail in 2.2.2. The pseudo code for this algorithm can be found in Algorithm 2.

Remember that my model needs both matrices **X** and **Y** to learn a PLS model. Moreover the customer just gave us the spectral matrix and a learned model. I used this given model to reproduce the process matrix **Y** by predicting its values. This is definitely needed to train any regression model $\mathbf{Y} = \mathbf{XB}$.

Suppose that `R--10` is the name of the dimension that has to be predicted. Figure 4.6 shows training data for my PLS model in the left window shows before training. The right window in Figure 4.6 displays the learned curve after training[1].

I can only assume that the given model uses a similar implementation. As I outlined in 2.2.2, some PLS algorithms will produce same results if the given problem is univariate.

4.3.2 Synthesis of Paracetamol

In [19] one can find an overall description of Paracetamol. Paracetamol or acetaminophen is probably the most common analgesic and antipyretic drug. It is used for the relief of fever, headaches and other minor aches and pains. Paracetamol in combination of lower dosages of stronger analgesics is helpful in treating more severe pain. It is a

[1]Note that the original data curve (blue) is nearly totally covered by the trained curve (red)

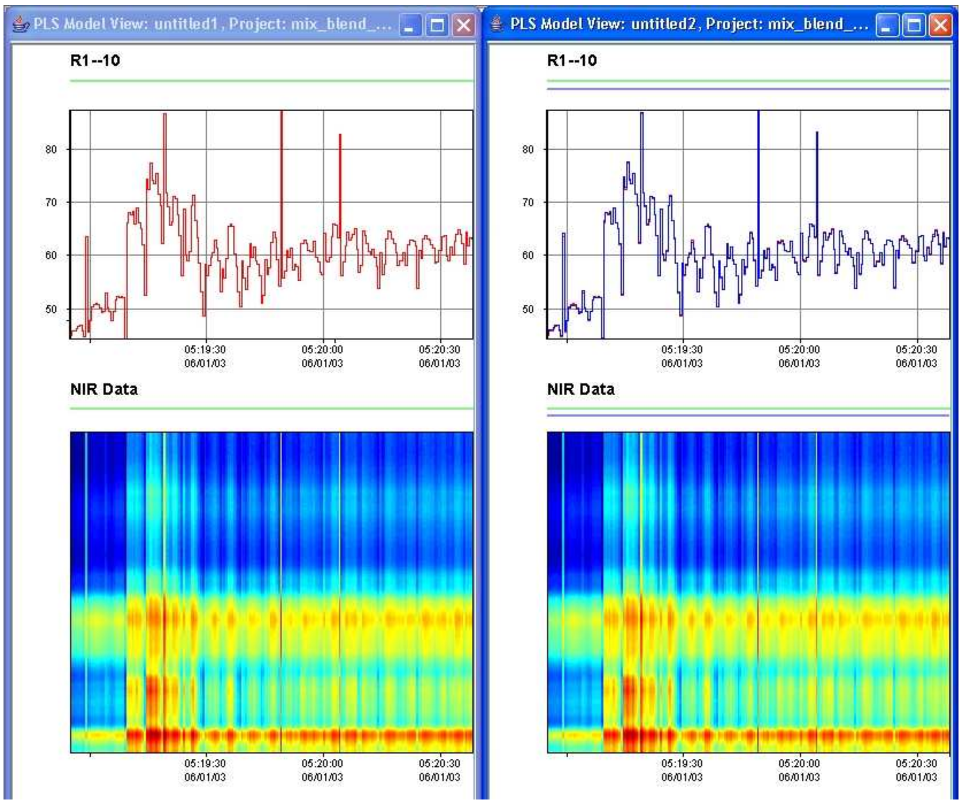

PLS Model View: untitled1, Project: mix_blend_...
R1--10
80
70
60
50
05:19:30 06/01/03
05:20:00 06/01/03
05:20:30 06/01/03
NIR Data
PLS Model View: untitled2, Project: mix_blend_...

Figure 4.6: Visual comparison of two PLS models. The left window shows the result of a given Simca-P model from a customer, whereas the right window shows the result with my PLS implementation.

major ingredient in numerous cold and flu medications and many prescription analgesics. Moreover it is remarkably safe in recommended doses.

Thus the mass production of Paracetamol is a necessity. Unfortunately until today it is not clear yet how to produce this drug very effective and efficient. The chemical reaction for the synthesis for Paracetamol is shown in Figure 4.7. This chemical process has many parameters which can be tuned. Hence it is too complex to be analyzed by human experts. MVDA with its methods is for sure one way to tackle this problem.

One customer of SCR ran several tests with different initialized parameters. All in all they performed 19 runs, so-called *batches*. The parameters that were changed are as follows:

- reactor temperature with $\{333.15\mathrm{K}, 348.15\mathrm{K}, 363.15\mathrm{K}\}$
- dosing time of acetic anhydride with $\{1\mathrm{min}, 30.5\mathrm{min}, 60\mathrm{min}\}$
- start-up concentration of p-amino phenol with $\{40\frac{\mathrm{g}}{\mathrm{l}}, 70\frac{\mathrm{g}}{\mathrm{l}}, 100\frac{\mathrm{g}}{\mathrm{l}}\}$ solvent
- molar ratio acetic anhydride/p-amino phenol with $\{1, 1.25, 1.5\}$

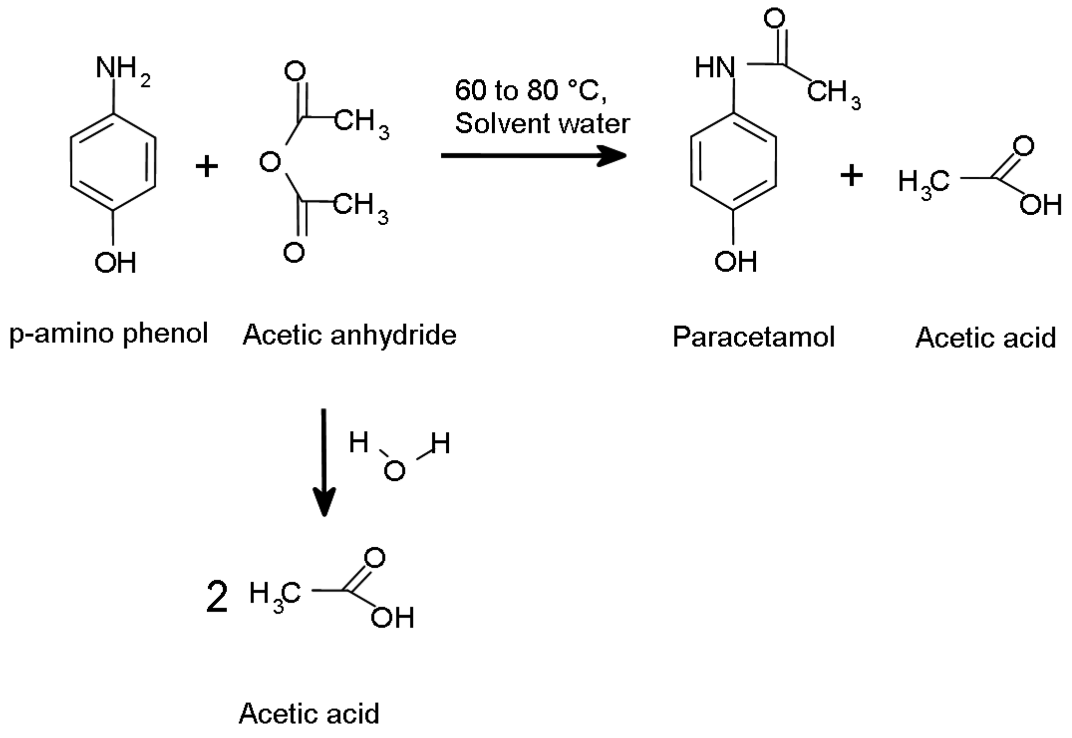


Figure 4.7: Synthesis of Paracetamol

Since the chemical reaction is know, it is not very hard to start calibrating this multivariate system. Remember that the first step is to specify the analytes with concentration ranges. Moreover it is very important to select the spectroscopic method and the wavelength range. We know which bonds maybe stretch and bend because it is predetermined in our chemical equation. Hence, the frequencies that may occur can be looked up in tables like for instance in [12].

Fortunately, this important knowledge was already given by the customer. Figure 4.8 shows the main region of interest lying inside the complete spectrum. The filtered wavenumber range $\tilde{\nu}$ amounts from $6413.85\frac{1}{\text{cm}}$ to $5623.2\frac{1}{\text{cm}}$. With equation (4.1) we can compute the equivalent wavelength range λ which is 1559nm to 1778nm.

$$\tilde{\nu} = 1/\lambda \tag{4.1}$$

Steps 2 (selection of a representative calibration set) and 3 (measurement of spectra) were also done by the customer. The best way to preprocess and filter data which is the next step is not clear. Thus I implemented several standard filters like Savitzky-Golay 2.1.5 or MSC 2.1.4.

Since adjacent data sets are smooth in time, one can easily reduce data by taking only every k-th time stamp. The implementation for subsampling takes 500 equally distributed time stamps out of all. Some tests showed that subsampling in wavenumbers is harder to apply. I assume that it will heavily depend on the given data sets if

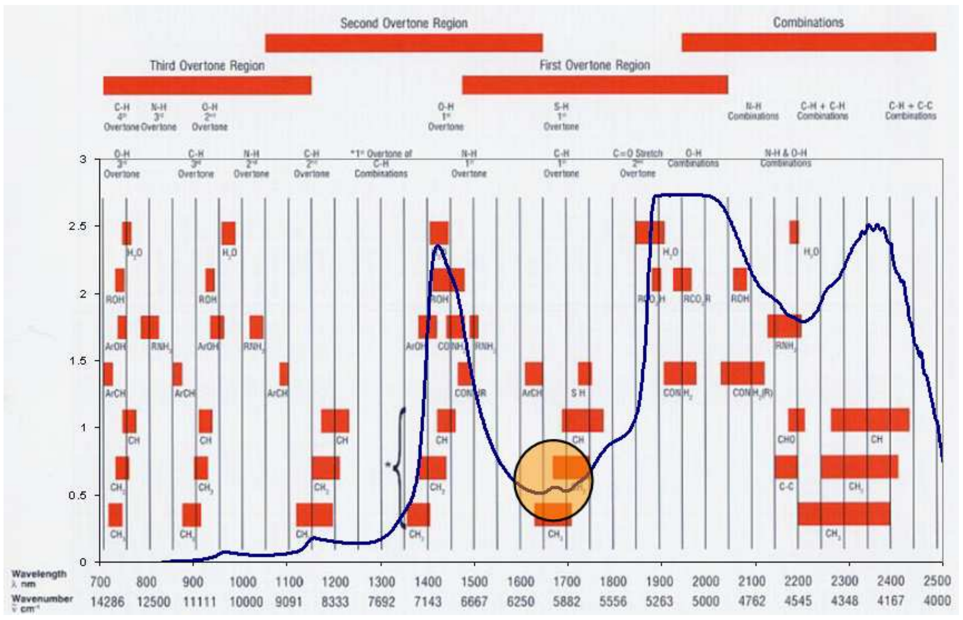


Figure 4.8: NIR spectrum of Paracetamol synthesis. The circle is showing the region of interest where one assumes the main information in the spectrum.

subsampling can be applied or not.

Furthermore centering 2.1.1 has to be done. PCA 2.2.1 expects at least centered data. If all dimensions have the same units, there will be no need for normalization. If this is not the case, one can apply normalization 2.1.2.

As I pointed out in Section 2.1.3, Pareto-Scaling works quite good empirically. Either one applies Pareto-Scaling, normalization or just centering. It is up to the user what should be done.

The customer sent a result for PCA with two principal components as it is shown in Figure 4.9. Our task was to reproduce exactly the same. Figure 4.10 shows the result. Unfortunately, this is the only way to compare the applied approaches since all used techniques were treated as confidential. More or less it was a test for SCR.

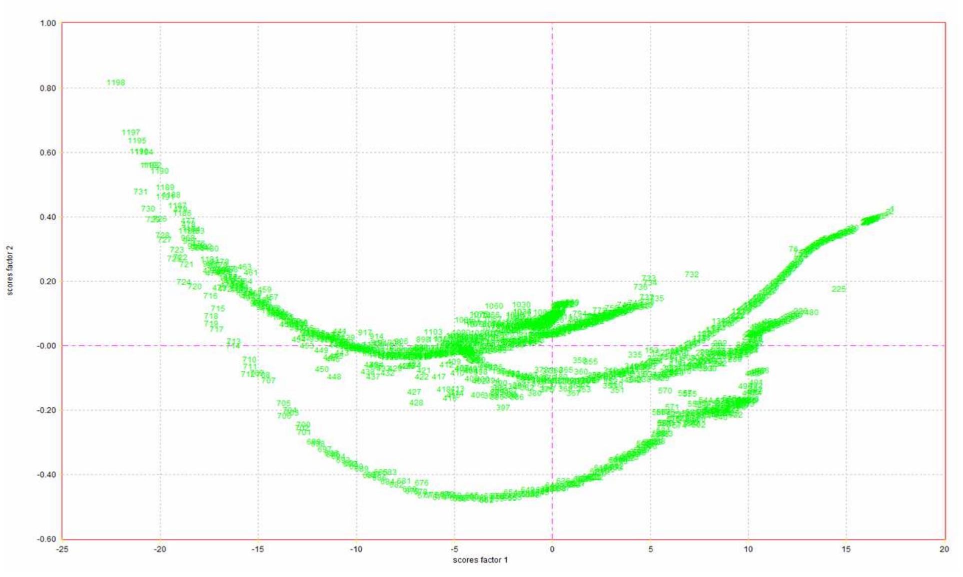
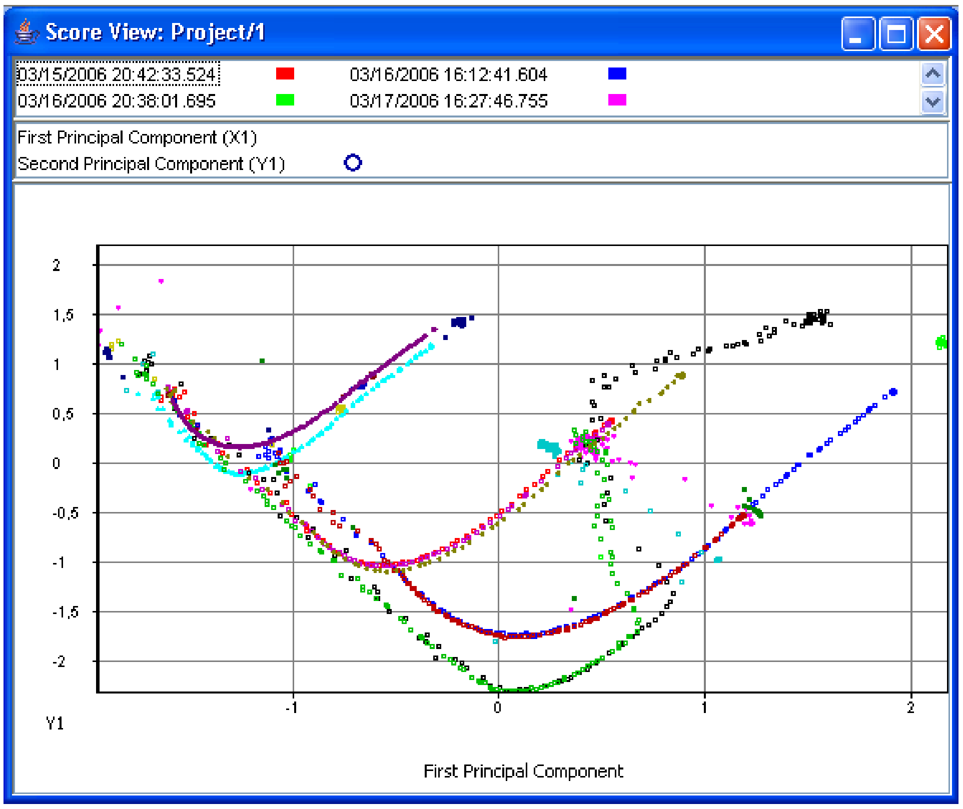


Figure 4.9: First and second principal components of several batch processes.



03/15/2006 20:42:33.524	🟥	03/16/2006 16:12:41.604	🟦
03/16/2006 20:38:01.695	🟩	03/17/2006 16:27:46.755	🟪

First Principal Component (X1)
Second Principal Component (Y1) ○

Figure 4.10: First and second principal components of 17 out of 19 batches with CAP. Only centering is applied as preprocessing method.

Chapter 5

Conclusions and Future Work

This final chapter summarizes the whole work. Furthermore it gives some ideas for improvements of the discussed algorithms and approaches on how this work could be continued.

5.1 Discussion

MVDA of NIR data are already established in Chemometrics. It is a well known and heuristically proven technique that leads to promising results. When chemical processes are complicated or complex, however, it is not enough to run some tests and let the computer find a model. The most important tasks that have to be solved is to describe the process with knowledge such that the computer generated model makes sense.

Since we maybe do not know the best set of initialized parameter for a chemical reaction, it is even very hard to tell which batch should be taken into the training set for the model. However, if the batch experiments are somewhat the same, the batches should be similar as well. Then batch analysis as pointed out in Section 2.3 can massively reduce outliers.

Another problem is the visualization of given data. Usually there are many variables that have to be reduced and transformed to those dimensions which carry most of the information. We introduced the basic concept of PCA to reduce the number of dimension to a certain minimum. Somehow we compressed data by throwing away lots of data that represent little information.

If we have a regression problem in Chemometrics where one has to predict $\mathbf{y}$ out of a certain $\mathbf{x}$, we can use PLS as powerful method to predict a small number of variables out of a set of highly correlated observables. Of course it is also possible to first reduce the spectra with PCA and second run PLS with the principal components of the original spectra.

5.2 Some Ideas for Improvements

At SCR, I was able to implement basic data mining techniques that were demanded by the customer. It is understood that this is not the optimum. There are so many open issues that have to be handled for sure.

First of all, what can we do if one sensor drops or fails briefly. Usually data can contain lots of missing values. They have to be replaced by some clever heuristics. In [13] we can find several methods to complete the missing measurements.

I briefly discussed MSC as a method for signal correction. Of course there are further approaches to tackle noise in spectral data. A new technique called orthogonal signal correction (OSC) was introduced by [21]. Since then, applications showed that OSC gives substantial improvements. In [3] one can find a real-world application that uses this technique as well.

PCA is only one possibility to compress data. Another way which shows to be very effective is wavelet compression. PLS in combination with wavelet transformation can be found in different papers. An introduction to this field can be found, e.g. in [18].

Remember in PCA that I fixed the number of principal components to two. This was motivated heuristically and was basically customer driven. Thus it is not hard to realize that this is a restriction since this strong assumption does not hold for all applications. [2] describe an approach to determine the number of principal components by cross-validation.

Moreover in PLS, I fixed the number of iteration steps as well. There are better and more intuitive ways to stop PLS. One possibility e.g. is to stop when a certain threshold for the approximation is passed under such that the residuals are small enough.

As I already outlined in Section 3.5.2, the JNI for Simca is only possible to load existing models so far. It is straight forward to implement other or even all Simca-P functions. Extending the Java native class `Simca.java` with some function heads and implementing their new function bodies in `simcawrapper.dll` is everything that has to be done.

For OPUS (see Section 3.5.3), there is still a GUI for CAP missing. Hence there exists no way to use the power of OPUS in CAP so far. This seems to be an easy task to me, too.

5.3 Conclusions

In the context of an system for time series analysis that was basically developed for breakdown and failure prediction, some elementary MDVA tools have been implemented. These tools are capable of applying a KDD process that includes noise reduction, error correction, data mining and prediction.

The treatment of noise is done with MSC which was especially designed for scatter correction of NIR spectra. Outliers are smoothed here by Savitzky-Golay Filtering that shows good results in Chemometrics. Dimension reduction is applied with PCA and PLS, respectively. Moreover PLS is used for multivariate regression analysis.

The implementation is written in a special framework for CAP in Java. It can be accessed by a GUI that is able to load and save so-called projects. A project may contain several models whereas these models can be PCA or PLS models. The given spectrum is visualized by a false-color plot. The principal components are shown as time series and as score plot. A learned PLS model is capable to show the original data and the learned curve for the dependent variables.

Based on this work, SCR is currently developing a system to plug CAP directly with the bioreactor to control the input.

Bibliography

[1] S. de Jong. SIMPLS: An alternative approach to partial least squares regression. *Chemometrics and Intelligent Laboratory Systems*, 18:251–263, 1993.

[2] H.T. Eastment and W.J. Krzanowski. Cross-validatory choice of the number of components from a principal component analysis. *Technometrics*, 24:73–75, February 1982.

[3] Lennart Eriksson, Johan Trygg, Erik Johansson, Rasmus Bro, and Svante Wold. Orthogonal signal correction, wavelet analysis, and multivariate calibration of complicated process fluorescence data. *Analytica Chimica Acta*, 420:181–195, 2000.

[4] Paul Geladi and Bruce R. Kowalski. Partial least-squares regression: a tutorial. *Analytica Chimica Acta*, 185:1–17, 1986.

[5] Inge S. Helland, Tormod Naes, and Tomas Isaksson. Related versions of the multiplicative scatter correction method for preprocessing spectroscopic data. *Chemometrics and Intelligent Laboratory Systems*, 29:233–241, 1995.

[6] Dirk C. Hinz. Process analytical technologies in the pharmaceutical industry: the fda's pat initiative. *Analytical and Bioanalytical Chemistry*, 384:1036–1042, 2006.

[7] Agnar Hoeskuldsson. Pls regression methods. *Journal of Chemometrics*, 2:211–228, 1988.

[8] Hector C. Keun, Timothy M.D. Ebbels, Henrik Antti, Mary E. Bollard, Olaf Beckonert, Elaine Holmes, John C. Lindon, and Jeremy K. Nicholson. Improved analysis of multivariate data by variable stability scaling: application to nmr-based metabolic profiling. *Analytica Chimica Acta*, 490:265–276, 2003.

[9] Sheng Liang. *Java Native Interface: Programmer's Guide and Reference.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[10] Fredrik Lindgren, Paul Geladi, and Svante Wold. The kernel algorithm for pls. *Journal of Chemometrics*, 7:45–59, 1993.

[11] H. Martens, S. Jensen, and P. Geladi. Multivariate linearity transformation for near-infrared reflectance spectrometry. In O.H.J. Christie, editor, *Proceedings of the Nordic Symposium in Applied Statistics*, pages 205–233, Stavanger, Norway, jun 1983. Stokkand Forlag Publ.

[12] Koji Nakanishi. *Infrared Absorption Spectroscopy: Practical.* Holden-Day, Inc. and Nankodo Company Limited, San Francisco, CA, USA and Tokyo, Japan, 1962.

[13] Philip R. C. Nelson, Paul A. Taylor, and John F. MacGregor. Missing data methods in pca and pls: Score calculations with incomplete observations. *Chemometrics and Intelligent Laboratory Systems*, 35:45–65, November 1996.

[14] K. Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, 2:559–572, 1901.

[15] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical recipes in C: the art of scientific computing.* Cambridge University Press, New York, NY, USA, 1988.

[16] A. Savitsky and M.J.E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Anal Chem*, 36:1627–1639, 1964.

[17] Rolf Sundberg. Multivariate calibration – direct and indirect regression methodology. *Scandinavian Journal of Statistics*, 26:161–207, 1999.

[18] Johan Trygg and Svante Wold. Pls regression on wavelet compressed nir spectra. *Chemometrics and Intelligent Laboratory Systems*, 42:209–220, August 1998.

[19] Wikipedia. Paracetamol — wikipedia, the free encyclopedia, 2006.

[20] Svante Wold. Chemometrics; what do we mean with it, and what do we want from it? *Chemometrics and Intelligent Laboratory Systems*, 30:109–115, 1995.

[21] Svante Wold, Henrik Antti, Fredrik Lindgren, and Jerker Öhman. Orthogonal signal correction of near-infrared spectra. *Chemometrics and Intelligent Laboratory Systems*, 44, December 1998.

[22] Svante Wold and Mats Josefson. Multivariate calibration of analytical data. pages 9710–9736, November 2000.

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Magdeburg, 22. Januar 2007

Christian Moewes