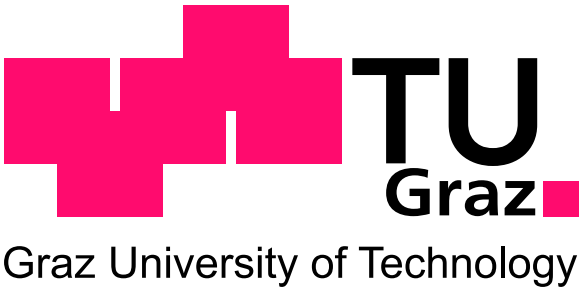
Bruno Cadonna

Implementation of an Analysis Module for Clinical Patient Data

Master Thesis



Institute for Genomics and Bioinformatics

Graz University of Technology

Petersgasse 14, 8010 Graz, Austria

Head: Univ.-Prof. Dipl.-Ing. Dr.techn. Zlatko Trajanoski

Supervisor: Dipl.-Ing. Peter Beck

Evaluator: Univ.-Prof. Dipl.-Ing. Dr.techn. Zlatko Trajanoski

Graz, March 2006

Abstract

Quality management in medical care cannot be just outcome-oriented, because outcomes are not always measurable or reversible. An overall quality management system is required by companies and organisations offering medical care services. These overall quality management systems to improve the quality of medical care are supported by the establishment of monitoring and control systems using information technology.

The objective of this work was the development of a reusable module for dynamically modifiable analyses of clinical patient data which facilitates handling analyses on a higher level of abstraction as plain SQL. Results of analyses should be visualised as customizable charts and tables.

Java technology was used to develop the data analysis module. The module generates SQL statements out of an XML document representing the analysis. The analysis result are visualised as charts using the charting library JFreeChart.

The developed data analysis module named PatAn allows handling analyses on a higher level of abstraction as with plain SQL. Recurring analysis patterns were identified and integrated in the analysis structure to achieve modularity, reusability and faster development.

Keywords: clinical data analysis, Java, XML, SQL, quality management

Zusammenfassung

Qualitätsmanagement in der medizinischen Versorgung kann nicht lediglich ergebnisorientiert sein, da Ergebnisse nicht immer messbar und reversibel sind. Ein umfassendes Qualitätsmanagementsystem ist nötig für Unternehmen und Organisationen, die medizinische Leistungen anbieten. Diese umfassenden Qualitätsmanagementsysteme zur Verbesserung der Qualität der medizinischen Versorgung werden durch die Einführung von Überwachungs- und Kontrollsystemen unter Verwendung der Informationstechnologie unterstützt.

Ziel dieser Arbeit war die Entwicklung eines wiederverwendbaren Moduls für dynamisch veränderbare Auswertungen von klinischen Patientendaten, das die Handhabung von Auswertungen auf einer höheren Abstraktionsebene als reines SQL ermöglicht. Die Ergebnisse der Auswertungen sollten als adaptierbare Diagramme und Tabellen visualisiert werden.

Die Java Technologie wurde verwendet, um das Datenauswertemodul zu entwickeln. Das Modul generiert aus einer Auswertung in XML Format SQL Anweisungen. Die Ergebnisse der Auswertung werden mit Hilfe der Charting-Bibliothek JFreeChart als Diagramme ausgegeben.

Das entwickelte Datenauswertemodul mit dem Namen PatAn erlaubt die Handhabung von Auswertungen auf einer höheren Abstraktionebene als reines SQL. Wiederkehrende Auswertungsmuster wurden identifiziert und in die Auswertungsstruktur integriert, um Modularität, Wiederverwendbarkeit und eine schnellere Entwicklung zu ermöglichen.

Schlagwörter: klinische Datenauswertung, Java, XML, SQL, Qualitätsmanagement

Contents

List of Figures

List of Tables

List of Abbreviations

ANSI	American National Standards Institute
API	Application Programming Interfaces
BARS	Benchmarking And Reporting Service
BIS	Basic Information Sheet
CSS	Cascading Style Sheet
CSV	Comma Separated Values
DBMS	DataBase Management System
DTD	Data Type Defintion
EAV	Entity-Attribute-Value
EBM	Evidence Based Medicine
EJB	Enterprise Java Beans
FQSD	Forum for Quality Systems in Diabetes Care (Forum QualitätsSicherung in der Diabetologie)
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IDF	International Diabetes Federation
ISO	International Organization for Standardization
J2EE	Java 2 Enterprise Edition
JPEG	Joint Photographic Experts Group - actually the name of the group which created the standard underlying the JPEG format
JSP	JavaServerPages
JVM	Java Virtual Machine

MSG	Institut für Medizinische Systemtechnik und Gesundheitsmanagement, the german name of the Institute of Medical Technologies and Health Management of the JOANNEUM RESEARCH Forschungsgesellschaft m.b.H.
OO	Object-Oriented
PatAn	Patient data Analyzer
PDCA	Plan Do Check Act
PDF	Portable Document Format
PDSA	Plan Do Study Act
PNG	Portable Network Graphics
SGML	Standarized Generalized Markup Language
SOP	Standard Operating Procedure
SQL	former SEQUEL Structured English QUEry Language
SVG	Scalable Vector Graphics
TDD	Test-Driven Development
TQM	Total Quality Management
URI	Unified Resource Identifier
W3C	World Wide Web Consortium
WHO	World Health Organisation
X11	X Windows System
XML	eXtensible Markup Language

Acknowledgements

This thesis and my entire studies would not have been possible without the fortune of having parents who have been disposed to finance me for years.

Without the professional support of my supervisor Peter Beck and of all the other people at the Institute of Medical Technologies and Health Management of the JOANNEUM RESEARCH Forschungsgesellschaft m.b.H. headed by Thomas R. Pieber, writing this thesis would have been much harder.

Zlatko Trajanoski – head of the Institute of Genomics and Bioinformatics at Graz University of Technology – evaluated this thesis.

Thanks a lot!

1. Introduction

1.1. Quality Management in Medical Care

The ISO 8042 standard defines *quality* as:

> "The totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs."

where *entity* may not only be a product, but it may be a service as well.
For the *quality of medical care* Donabedian stated the following definition:

> "Quality of care is the extent to which actual care is in conformity with preset criteria for good care."[11]

Quality management is important for providing high quality products and services to customers. In product manufacturing quality management is popular and applied with success, whereas in the service domain it has not been completely accepted and implemented yet. This applies also to the medical care domain, which should offer patients the possibility to take advantage of high quality medical care services.

Quality management in medical care services is also demanded in Austrian law:

> "§1. (1) Zur flächendeckenden Sicherung und Verbesserung der Qualität im österreichischen Gesundheitswesen ist systematische Qualitätsarbeit zu implementieren und zu intensivieren. Dazu ist ein gesamtösterreichisches Qualitätssystem basierend auf den Prinzipien Patientinnen- und Patientenorientierung, Transparenz, Effektivität und Effizienz nachhaltig zu entwickeln, umzusetzen und regelmäßig zu evaluieren. Dabei ist insbesondere die Qualität bei der Erbringung von Gesundheitsleistungen unter Berücksichtigung der Patientinnen- und Patientensicherheit zu gewährleisten."[3]

Quality management changed in the last decades. New concepts of quality management are no longer just outcome-oriented – this means restricted to check the quality of process outcomes – but they cover all aspects of a process and the people involved in it [17]. Such a overall concept would be helpful for quality management of medical care services where the outcome is not always measurable or reversible.

Total Quality Management (TQM) is a quality management concept firstly developed by W. Edwards Deming which is not restricted to check outcomes, but covers all aspects of a company or organisation. TQM is defined by the Deming Prize committee as follows:

> "TQM is a set of systematic activities carried out by the entire organization to effectively and efficiently achieve company objectives so as to provide products and services with a level of quality that satisfies customers, at the appropriate time and price. [...]"[22]

And in the more detailed explanation in [22] there is written:

> "[...] The company maintains and improves its processes and operations, and uses appropriate statistical techniques and other tools. Based on facts, the company manages its business by rotating the management cycle of PDCA (plan, do, check and act). The company also rebuilds its management system by utilizing appropriate scientific methods and information technology. [...]"

This means a company or organisation which wants to offer high quality medical care services following TQM concepts needs to implement the PDCA[1] cycle by applying appropriate methods and tools for its settings and utilizing information technology for that purpose.

1.2. FQSD

1.2.1. Saint Vincent Declaration

The Saint Vincent Declaration is a document containing recommendations to improve the quality of care for the therapy of diabetes mellitus. It originated from a meeting of representatives of government health departments, patients organisations from all European countries and diabetes experts organised by the World Health Organisation (WHO) and the International Diabetes Federation (IDF) in 1989 in Saint Vincent, Italy. All participants unanimously agreed upon the stated recommendations in the declaration.

The Saint Vincent Declaration consists of two general goals:

> "General goals for people - children and adults - with diabetes:
>
> - Sustained improvement in health experience and a life approaching normal expectation in quality and quantity.
> - Prevention and cure of diabetes and of its complications by intensifying research effort."[16]

[1] also refered to as PDSA with "study"instead of "check"or Deming cycle

and several five-year targets to reach the goals. The targets refering to the reduction of the major diabetes related complications and to the establishment of monitoring and control systems are the most important in the context of this thesis. These two targets taken from [16] are quoted in the following:

- "Implement effective measures for the prevention of costly complications:
  - Reduce new blindness due to diabetes by one third or more.
  - Reduce numbers of people entering end-stage diabetic renal failure by at least one third.
  - Reduce by one half the rate of limb amputations for diabetic gangrene.
  - Cut morbidity and mortality from coronary heart disease in the diabetic by vigorous programmes of risk factor reduction.
  - Achieve pregnancy outcome in the diabetic woman that approximates that of the non-diabetic woman.
- Establish monitoring and control systems using state of the art information technology for quality assurance of diabetes health care provision and for laboratory and technical procedures in diabetes diagnosis, treatment and self-management."

The complete list of the five-year targets in the Saint Vincent Declaration can be found in appendix A.

1.2.2. Aim of FQSD

The aim of FQSD is developing and applying tools to implement a quality management system that assures high quality of care for the therapy of diabetes mellitus in order to reach the goals of the Saint Vincent Declaration.

The project *Forum Qualitätssicherung in der Diabetologie (FQSD)* is a cross-national initiative adressing quality management in the therapy of diabetes mellitus. It was founded in 1994 in Germany (FQSD-D) and 1996 in Austria (FQSD-Ö) as reaction to the poor achievements regarding the five-year targets of the Saint Vincent Declaration. The members of FQSD are diabetes consultants, dieticians, training personnel and physicians from all kinds of diabetes care centers like university hospitals, general hospitals, rehabilitation centers, medical practices specialised in internal medicine and general practices.

According to FQSD the PDCA cycle in figure 1 has to be implemented by a diabetes care center to reach the goals of the Saint Vincent Declaration.

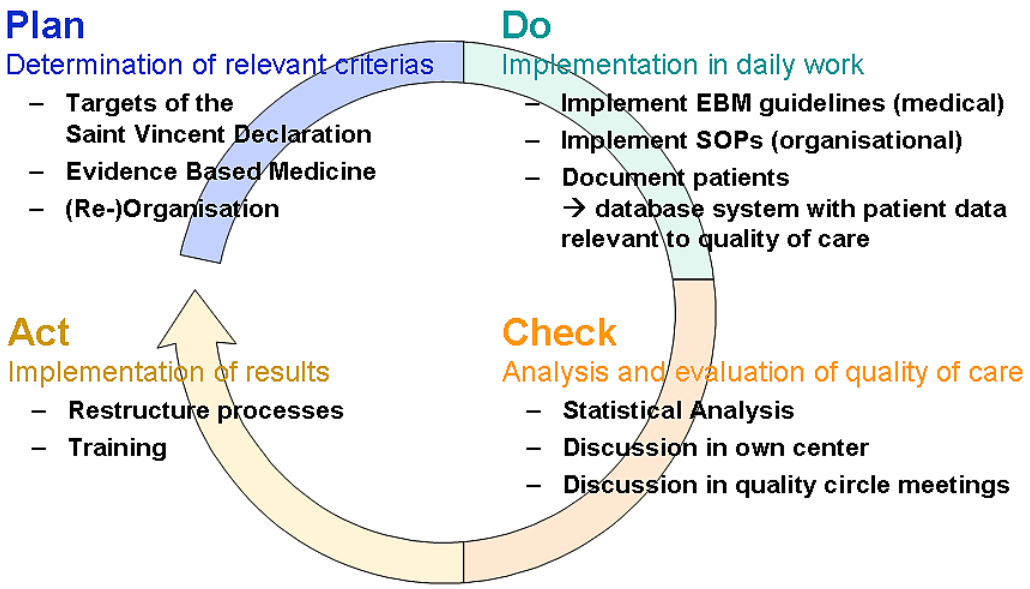


Figure 1: Interpretation of the PDCA cycle in medical care involving Evidence Based Medicine (EBM) according to FQSD

In the *Plan*-phase the relevant criteria for the quality management of medical care are determined refering to the Saint Vincent Declaration and involving Evidence Based Medicine (EBM). Moreover the further proceedings to assure and increase the quality are organised by defining EBM guidelines and Standard Operating Procedure (SOP). In the *Do*-phase EBM guidelines and SOPs are implemented in daily work and patients documentations are collected. In the *Check*-phase the collected documentations are analysed applying statistical methods and used as basis for discussions within the own center and in quality circle meetings with other centers. In the *Act*-phase the findings from the previous phases are implemented by optimising processes and training participants. Finally the cycle starts again.

FQSD offers following methods and tools to realise the PDCA cycle:

- a database system to collect patient data of the own center according to the Basic Information Sheet (BIS) adopted from WHO's DiabCare initiative[2];
- *open benchmarking* (see section 1.3) to analyse and compare the own collected data with other known centers;

[2]WHO's DiabCare initiative was created to develop methods and tools to achieve the targets of the Saint Vincent Declaration.

- quality circle meetings to discuss in a small group results from collected data and to develop strategies to improve quality of care;
- workshops to discuss results from collected data and new scientific outcomes as well as gain further knowledge about quality management in medical care.

The information technology systems used by FQSD to achieve its aims are developed and maintained by the Institute of Medical Technologies and Health Management of the JOANNEUM RESEARCH Forschungsgesellschaft m.b.H. (MSG) and provided under the trademark Healthgate.

1.3. Healthgate

Healthgate hosted at `www.healthgate.at` offers web applications to monitor and control quality of care. The web applications are not restricted to support the quality managment of diabetes mellitus, they also support the quality management of other chronic disease as hepatitis C and cardiovascular disease. These web applications assist medical facilities in collecting and analysing their patient data, organising quality circle meetings and searching medical knowledge obtained through EBM methods. Medical facilities – subsequently refered to as *centers* in this thesis – may be hospitals, rehabilitation centers or medical practices.

1.3.1. Healthgate BARS

BARS is the acronym for *Benchmarking And Reporting Service*. It is a web application used by health care professionals to collect clinical patient data according to BIS, perform data analyses and *open benchmarking*. Using benchmarking health care professionals can compare the performance of their center with other centers. Open benchmarking means that all centers are known, so that the center with the best or worst performance can be identified. A schematic illustration of Healthgate BARS is shown in figure 2.

A center collects patient data either online using its Healthgate BARS account or on paper sending it by ground mail to its regional office. Benchmarking reports containing analysed data from selected centers for comparison are available either in paper format by ground mail four times a year or online at any time.

The development of the FQSD information system for open benchmarking on clinical patient data for management of diabetes mellitus was initialised in 1997/98 by

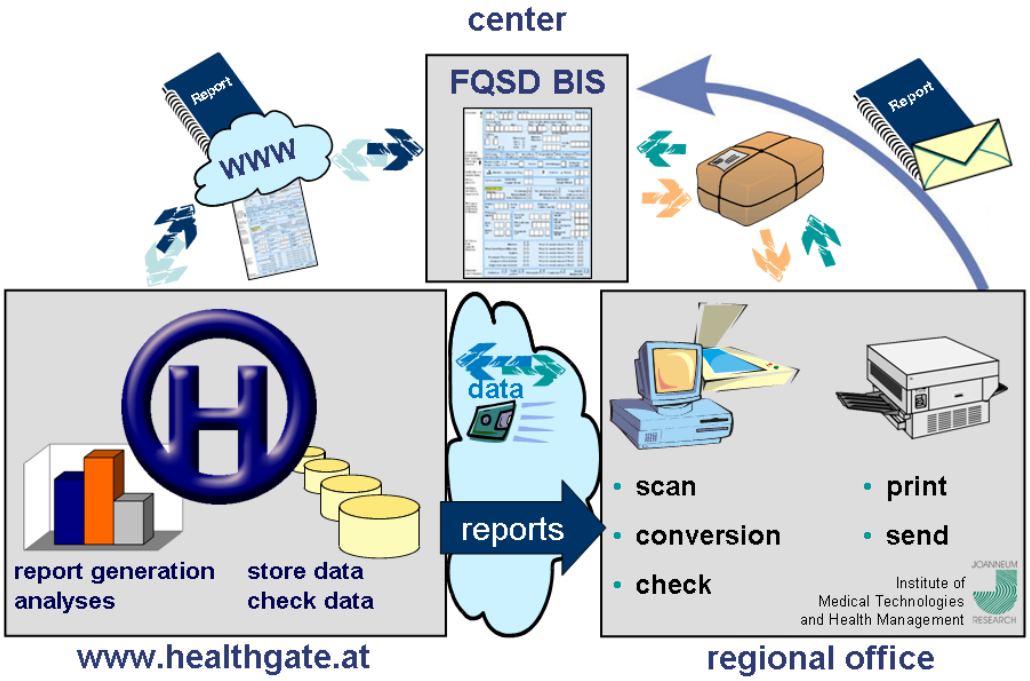


Figure 2: Schematic illustration of data collection and analysis with Healthgate BARS

FQSD in collaboration with the Institute of Biomedical Engineering Graz University of Technology and financed by the federal province of Styria and Novo Nordisk Pharma. At the beginning the FQSD information system consisted of a decentralised Microsoft Access database where each participating center had its own copy of the database on its computer. Each center could formulate its own queries to analyse data and compare the results with other centers. Predefined queries and their graphical representation were also already implemented. Through synchronisation with a central server it was possible to make recently collected patient data available to all other centers and update its local copy of the database. This synchronisation turned out to be a problem due to deficiencies in the Microsoft Access synchronisation implementation and the steadily increasing number of participating centers and the consequently increasing amount of data sent back and forth which needed more and more system resources. Moreover the Microsoft Access database management system changed between releases so that incompatibilities arose.

In 1999 the development of a new FQSD information system using a client-server architecture started [7]. After a year of research, designing and testing and a further year of implementation, at the beginning of 2001 a new version of the FQSD information system was released, the first version of Healthgate BARS. This system –

which is still in use – is a web application developed using Java 2 Enterprise Edition (J2EE) technologies. The clinical patient data is stored in a central database avoiding the synchronisation of all participating centers. The system provides a web interface accessed by a web browser without the need to install a local client. In Healthgate BARS there are several predefined analyses with user-modifiable parameters which can be executed on clinical patient data comparing centers to each other.

To develop analyses for Healhtgate BARS an editor for the creation and execution of queries using Extensible Markup Language (XML) was developed [13]. The editor tried to offer the same flexibility in query definition as the Microsoft Access database. Moreover with structured query coded in XML it is possible to better manage access rights of participating centers to clinical patient data. The analyses now available in Healthgate BARS were developed with this editor. Since predefined queries turned out to be sufficient for the participating centers, the formulation of own queries was no longer demanded and the editor was therefore not made available to the centers.

1.4. Requirements

In the actual version of Healthgate BARS the data is analysed using queries formulated as XML documents. The structure of these XML documents mirrors the functionality of SQL one-to-one. One XML element corresponds to an SQL element. This approach facilitates structuring analyses and eases its validation and modification through already existing libraries for XML processing. Moreover with XML it is possible to integrate additional information in the analyses such as data access rights. However, the analyses are still strongly dependent on the database structure. Additionally, the execution of the same analysis on different subsets of data requires complex modifications in several places and not in a central place, which makes the analyses less reusable.

To overcome these limitations, a reusable module for user-modifiable analysis of clinical patient data in multiple applications should be developed. This data analysis module should handle analyses on a higher level of abstraction. This would increase modularity and reusability of analyses and the data analysis module itself saving query and software development time.

1.4.1. Abstraction

Handling analyses on a higher level of abstraction should be possible if recurring patterns can be identified in existing analyses and expressed as generic structure. The

abstract analyses could be developed and maintained faster, because the recurring patterns would be pre-implemented in configurable components. Through the abstraction it should be easier to modify the analysis dynamically at runtime. Thus, it could be possible to analyse different subsets of data with the same analysis and to easily select whether an analysis should be executed over time or not.

1.4.2. Visualisation

An analysis of clinical patient data consists of the execution of a query and the visualisation of its results. The results of an analysis should be visualised as charts and tables. Charts give a coarse overview of the results. Big differences are easily identified. However, charts can be distorted. Tables show the numeric values of results in a structured form. Big differences are not so easily identified, but tables show the exact numeric values, hence they are much harder to distort as charts.

The visualisation of the results should be costumizable. The user should be able to change the appearance in order to use the visualisation in presentations and publications. Where possible the user should be able to toggle between absolute and relative representation of the results without the need of querying the database again. Moreover, the total amount of records included in the analysis should be available for display at any time.

The visualisation should be internationalised in order to be able to use the data analysis module for different languages.

Since not each chart type can represent all types of results, it should be possible to define which chart types can be used to visualise the results of an analysis. For example average value and standard deviation cannot be visualised through a pie chart. Therefore a result containing average value and standard deviation should not be visualised through a pie chart.

1.4.3. Modularity

Modularity means that the data analysis module can be configured in multiple applications with different data and different technical requirements.

The data structure for which the data analysis module may be integrated can differ among systems. For example a system to analyse data of diabetes care centers may have a different data structure than a system to analyse data of patients in an intensive care unit. Hence the used database structure has to be configurable to ensure modularity.

The module should be independent from the architecture of the surrounding application in order to be usable with different application architectures. For example a J2EE web application differs from a Java stand-alone application in several aspects. The most important aspect for data analysis is how the application manages the database connection.

Almost all database management systems (DBMS) use their own SQL dialect. Hence the DBMS used has to be configurable within the data analysis module to ensure modularity.

2. Methods

2.1. SQL

SQL (or SEQUEL) is a standard for manipulating and querying relational databases. SQL was developed during the 1970s at IBM. The name SQL is derived from SEQUEL, which means Structured English QUery Language. The name had to be changed from SEQUEL to SQL due to a trademark dispute. Since 1986 SQL has been an ANSI and since 1987 an ISO standard. The actual version of the standard is SQL:2003.

The part of SQL for querying relational databases is set-oriented. This means that a query bundles data to a set and the data of interest can be selected from this set. Additionally SQL provides functions to process the selected data. Examples for such functions are the computation of the average value and of the standard deviation from a group of data within the set.

Let us suppose we have a table in a relational database named `famous` with information about famous people, such as:

name	**city**	**id**
Jim Morrison	Los Angeles	1
Elvis Presley	Memphis	2
Andy Warhol	New York	3
Lou Reed	New York	4
Janis Joplin	San Francisco	5

If we want to know who of the famous people lives in New York, we would write the following SELECT statement:

```
SELECT name
FROM famous
WHERE city = 'New York'
```

The FROM clause defines the set of data, which in our case is the table containing the information about the famous people. An element of the set is a table row. The WHERE clause restricts the set to the elements which have got 'New York' as value in the column containing the cities. Finally the SELECT clause selects only the names from the remaining elements. The result set of our SELECT statement would be:

name
Andy Warhol
Lou Reed

On the one hand SQL is very popular among relational database management systems (DBMS), but on the other almost each relational DBMS implements its own slightly different version of the SQL standard – also known as SQL dialects – which leads to a lack of interoperability of SQL statements. In the majority of the cases SQL statements written on one DBMS cannot be executed on another DBMS.

2.2. XML

Extensible Markup Language (XML) is a W3C Recommendation for structuring data objects [9]. XML was developed by the XML Working Group, formed under the auspices of the World Wide Web Consortium (W3C) in 1996. It is a subset of the Standard Generalized Markup Language (SGML).

A data object, which is structured according to XML, is called an XML document. An XML document consists of elements. Each element can contain zero or more elements or character data. Mixed contents consisting of elements and character data are allowed as well. Additionally, each element can have zero or more attributes, whose values are character data. The element which is not contained in any other element is called root element. An XML document can have only one root element. Thus the XML document and – at the same time – the represented data object get a hierarchical structure, the so-called XML schema or simply schema. Because of the schema each element can be easily and unambiguously retrieved.

An example of an XML document is the purchase order taken from [12]:

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
   <shipTo country="US">
      <name>Alice Smith</name>
      <street>123 Maple Street</street>
      <city>Mill Valley</city>
      <state>CA</state>
      <zip>90952</zip>
   </shipTo>
   <billTo country="US">
```

```
      <name>Robert Smith</name>
      <street>8 Oak Avenue</street>
      <city>Old Town</city>
      <state>PA</state>
      <zip>95819</zip>
   </billTo>
   <comment>Hurry, my lawn is going wild!</comment>
   <items>
      <item partNum="872-AA">
         <productName>Lawnmower</productName>
         <quantity>1</quantity>
         <USPrice>148.95</USPrice>
         <comment>Confirm this is electric</comment>
      </item>
      <item partNum="926-AA">
         <productName>Baby Monitor</productName>
         <quantity>1</quantity>
         <USPrice>39.98</USPrice>
         <shipDate>1999-05-21</shipDate>
      </item>
   </items>
</purchaseOrder>
```

The `<purchaseOrder>`-element is the root element of the purchase order. The `<purchaseOrder>`-element has the attribute `orderDate` for the date of the order and contains the elements `<shipTo>` for the address to which the items should be shipped, `<billTo>` for the address to which the bill should be sent, `<comment>` for a comment regarding the order and `<items>` for the purchased items.

The schema of an XML document can be defined according to one's needs and an XML document can be validated according to the defined schema. Some XML schema languages – which are languages to define schemas – are not just capable to determine the hierarchy of the elements but also to specify the type of the character data contained in the elements and in the attributes, the so-called data typing. Thus the validation checks the correctness of the structure and the type conformance of all the character data.

2.2.1. W3C XML Schema

W3C XML Schema is one of several XML schema languages [12]. It was developed by the XML Schema Working Group of the W3C. Before XML Schema was approved as a W3C Recommendation in 2001, schemas were described by Data Type Defintions (DTD).

XML Schema provides functionality above and beyond what is provided by DTD. DTD almost completely lacks support for data typing. There is no way to define an element, which should contain only predefined character data like for example the numbers between 1 and 12 to represent the months. With XML Schema this would be possible. XML Schema provides powerful data typing, which allows a more precise definition of the character data to be used in an XML document than it would be possible using a DTD. A schema defined using XML Schema is by itself an XML document, hence one can process the XML document and the corresponding schema with the same tools. One of the most important advantages compared to DTD is the support of XML Namespaces.

An example of an XML Schema is the schema definition of the purchase order from the previous section taken from [12] with added comments:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:annotation>
    <xsd:documentation xml:lang="en">
     Purchase order schema for Example.com.
     Copyright 2000 Example.com. All rights reserved.
    </xsd:documentation>
  </xsd:annotation>

  <!-- purchase order element -->
  <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

  <!-- comment element -->
  <xsd:element name="comment" type="xsd:string"/>

  <!-- purchase order type -->
  <xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
```

```
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items"  type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>

<!-- adress type for US adresses -->
<xsd:complexType name="USAddress">
  <xsd:sequence>
    <xsd:element name="name"   type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city"   type="xsd:string"/>
    <xsd:element name="state"  type="xsd:string"/>
    <xsd:element name="zip"    type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>

<!-- items to order -->
<xsd:complexType name="Items">
  <xsd:sequence>
    <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="productName" type="xsd:string"/>
          <xsd:element name="quantity">
            <xsd:simpleType>
              <xsd:restriction base="xsd:positiveInteger">
                <xsd:maxExclusive value="100"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:element>
          <xsd:element name="USPrice"  type="xsd:decimal"/>
          <xsd:element ref="comment"   minOccurs="0"/>
```

```
            <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
          </xsd:sequence>
          <xsd:attribute name="partNum" type="SKU" use="required"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <!-- Stock Keeping Unit, a code for identifying products -->
  <xsd:simpleType name="SKU">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-[A-Z]{2}"/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

The element `<xsd:element>` with the attributes `name="purchaseOrder"` and `type="PurchaseOrderType"` defines an element with the name `purchaseOrder` of type `purchaseOrderType`. It defines the `<purchaseOrder>`-element of the XML document shown above. The type `purchaseOrderType` is defined further below in the schema.

2.2.2. XML Namespaces

XML namespaces allow to use elements and attributes (markup vocabulary) which are defined in different XML schemas in one single XML document. Thus XML schemas become more modular due to the ability to reuse existing markup vocabularies rather than reinvent them. An XML namespace is a collection of names, identified by an URI reference. XML namespaces are defined by a W3C Recommendation [8].

An example for the use of XML namespaces is the modified XML document for the purchase order from the previous sections:

```
<?xml version="1.0"?>
<po:purchaseOrder xmlns:po="http://www.foo1.org/purchase-order"
                  xmlns:price="http://www.foo2.org/price"
                  orderDate="1999-10-20">
```

```
   <po:shipTo country="US">
      <po:name>Alice Smith</po:name>
      <po:street>123 Maple Street</po:street>
      <po:city>Mill Valley</po:city>
      <po:state>CA</po:state>
      <po:zip>90952</po:zip>
   </po:shipTo>
   <po:comment>Hurry, my lawn is going wild!</po:comment>
   <po:items>
      <po:item partNum="872-AA">
         <po:productName>Lawnmower</po:productName>
         <po:quantity>1</po:quantity>
         <price:price>
           <price:value>148.95</price:value>
           <price:currency>USD</price:currency>
         </price:price>
         <po:comment>Confirm this is electric</po:comment>
      </po:item>
      <po:item partNum="926-AA">
         <po:productName>Baby Monitor</po:productName>
         <po:quantity>1</po:quantity>
         <price:price>
           <price:value>39.98</price:value>
           <price:currency>USD</price:currency>
         </price:price>
         <po:shipDate>1999-05-21</po:shipDate>
      </po:item>
   </po:items>
</po:purchaseOrder>
```

In the above example there are two XML namespaces. One XML namespace is identified by the URI

`http://www.foo1.org/purchase-order`

with the prefix `po` and the other is identified by the URI

`http://www.foo2.org/price`

with the prefix `price`. In other words, the XML schema of the modified purchase order XML document is part of the former XML namespace and imports the markup vocabulary of the latter XML namespace.

2.3. Java

The Java platform is a computing environment, which can run applications developed using the object-oriented (OO) and platform independent Java programming language. The Java platform was developed by Sun Microsystems and was first released in 1995. Different versions and editions are available for free download at [21]. The Java technology became famous because it brought the motion to the internet through Java applets. Altough over the years the Java applets were superseded more and more by other technologies (for example Macromedia Flash), on the server-side Java gained popularity with JavaServerPages, Servlets, Enterprise Java Beans and other related technologies, especially in the web application business.

2.3.1. The Java Platform

The Java platform consists basically of the Java virtual machine (JVM) and the provided Java class libraries. The JVM is a virtual processor, which interprets and excutes Java bytecode programs. The Java bytecode is produced by the Java compiler from Java source files. Since Java bytecode needs solely the JVM to execute, a pure Java application can be executed on any operating system platform which can run a JVM. At the time of writing the JVM was ported to the most popular operating system platforms as Microsoft Windows, Solaris and Linux. Because of the intermediate step of interpreting Java bytecode by the JVM instead of translating Java source code directly to machine code the Java platform has a lower performance as other programming languages, which is – facing the permanently increasing computing power, the continuous enhancements in JVM technology and the big advantage of the *write once run everywhere* concept – a minor issue. Besides the JVM the Java platform comes along with the provided Java class libraries, which can slightly differ between the editions. The provided Java class libraries represent a collection of application programming interfaces (APIs). They comprise core components like strings, numbers and basic input/output elements as well as more advanced components as for executing SQL statements on databases. To additionally augment the power of Java there are a lot of projects dealing with Java on the internet, many of which are open source projects.

2.3.2. Object Orientation in the Java Programming Language

The OO approach of the Java programming language allows to develop applications in a modular way. The modularity supports the reuse of existing and functioning source code. Object orientation in the Java programming language has the following concepts:

- An object bundles data and methods to access and process the data.
- A class specifies the implementation of objects of the same type.
- An interface defines the access to objects, thus objects of different classes which implement the same interface can be used interchangeably.
- A class can inherit data and methods from another class, thus a hierarchy of abstractions becomes possible. Through inheritance it is possible to extend an existing object for one's needs without redeveloping the required functionality from scratch.

All these features of the Java programming language increase modularity and consequently the reusability of source code saving resources in the development process.

2.4. JUnit

JUnit is an open source testing framework developed by Erich Gamma and Kent Beck to write and perform automated unit tests in Java [2]. Unit tests grouped in test cases check the correctness of small components of an application.

Unit testing is the building block to the software engineering method of test-driven development (TDD). TDD is summarized in two simple rules [6]:

- Write a failing automated unit test before you write any application source code.
- Refactor the source code to remove duplications.

Therefore a TDD process consists of three iterative steps:

- Write a failing unit test.
- Write application source code, which passes the unit test.
- Refactor the source code in order to remove duplications.

Writing a unit test before writing any source code of the component under test leads to a better understanding what the component should do. At the same time the unit test represents an instrument for feedback, which improves the quality assurance of the development process. To make unit testing easy, an application must consist of loosely coupled components, thus the application becomes more modular. The feedback and the increased modularity achieved by unit tests enhance the reusability and simplifies the further development of the application.

In JUnit each test case is a Java class which refers to a component under test. A component under test can be a group of Java classes, but mostly it is a single Java class. A unit test within a test case executes in three steps:

1. a setup method (`setUp()`);
2. the test method (`runTest()`);
3. a deinitialisation method (`tearDown()`);

Through initialisation (e.g. create or initialise an object) and deinitialisation (e.g. closing connections) it is possible to ensure the same preconditions for each unit test and independency among unit tests for a test case.

Within the unit test it is possible to define assertions to check, whether the returned values of the component under test are right. The result of a unit test can be of three types:

- The unit test can run correctly, this means all assertions are fulfilled.
- The unit test can fail, this means at least one assertion was not fulfilled.
- The unit test can throw an error, which means that a unanticipated error occurred.

The different result types ease the search for errors in the component under test.

JUnit includes a graphical user interface to show the test results called JUnit Test Runner. Additionally many integrated development environments (IDE) for Java provide support for JUnit.

2.5. XMLBeans

XMLBeans is an open source XML-Java binding tool first developed by BEA Systems and in September 2003 submitted to the Apache Incubator and the Apache XML

Project. Since June 2004 XMLBeans is a top level project at the Apache Software Foundation [5].

XMLBeans gives an object view of an XML document. In computer science context the process of deserialising an object is called unmarshalling and the opposite – serialising an object – is called marshalling. XMLBeans supports unmarshalling (getting the object view from an XML document) and marshalling (getting the XML document from an object view). The classes and interfaces needed for unmarshalling are generated by an Ant task [4] using the XML Schema, which defines the structure of the XML document. The generated classes and interfaces allow to access the objects representing the elements of an XML document through JavaBean-style accessors like `getFoo()` and `setFoo()`. For example the `<purchaseOrder>` element in 2.2.2 would be represented as an XMLBeans object of class `PurchaseOrder` with methods `getShipTo()` and `setShipTo()`. In this way one can get information from and set information in an XML document.

In XMLBeans it is possible to navigate an XML document and access the information on a low level using a cursor. A cursor can move inside an XML document between tokens. Tokens can have different levels of granularity, a token can be a whole element with children, an attribute or another component of an XML document.

XMLBeans provides full XML Schema support[3] and it provides a XML Schema object model which contains the schema meta information. Schema meta information is for example an enumeration of possible values for an attribute which could be useful to display in an application.

XMLBeans has performance benefits through incremental unmarshalling. An XML document is parsed by XMLBeans, but it is not unmarshalled at once. The original structure of the XML document is stored and only on demand an XMLBeans object is instantiated. This means, that if an element of an XML document will never be accessed, the corresponding XMLBeans object will never be instantiated.

2.6. JFreeChart

JFreeChart is an open source charting library for Java maintained by David Gilbert which can be used in applications, applets, servlets and JavaServerPages(JSP) [1]. JFreeChart generates charts as for example pie charts, bar charts and line charts from a dataset.

In JFreeChart there are several kinds of datasets. The interface `CategoryDataset`

[3] ... or nearly full support [18]

for example declares a dataset which can be used in charts showing values belonging to categories such as bar charts and line charts. An example for such a dataset could be the amounts of people (values) in a city belonging to different religious groups (categories). Furthermore there are other datasets for time series, statistical data and more.

In JFreeChart the charts are versatile in their representation through the ability of customising the visual components and the different data formats in which the chart can be exported. The customisable visual components of a chart are the colors of the chart, but also the title, subtitles and labels to add additonal information. The charts can be exported to PNG, JPEG, PDF and SVG. JFreeChart uses the Java 2D API [19] for drawing.

JFreeChart runs in Java Headless mode. Java Headless mode allows executing Java code that relies on AWT in a server environment running Unix/Linux without X11. It is the preferred solution for Java 2 release 1.4 and later [20]. The charting library from KLGroup[4] used in Healthgate BARS at the moment does not run in Java Headless mode.

[4]now called Quest

3. Results

The developed data analysis module named PatAn [10] is schematically shown in figure 3. PatAn is short for PATient data ANalyzer.

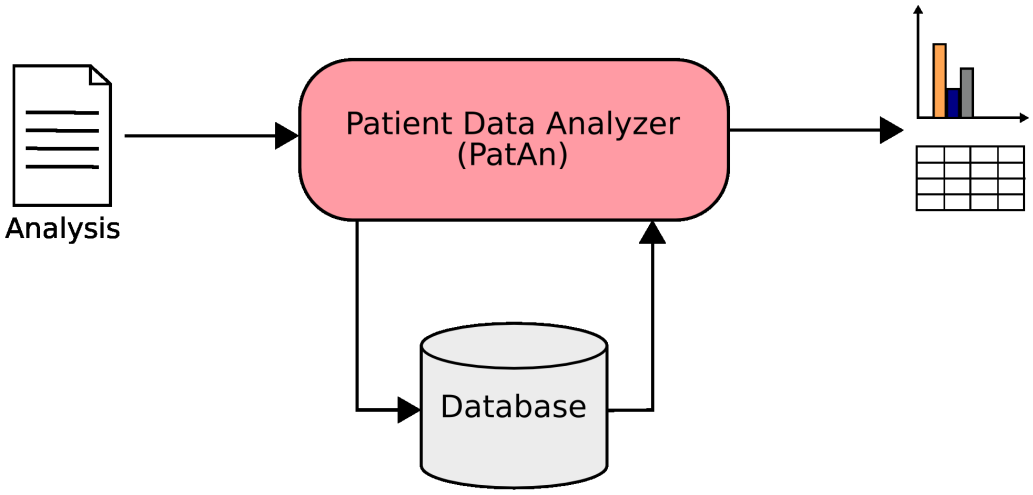


Figure 3: Schematic illustration of the Patient Data Analyzer (PatAn)

PatAn takes an analysis as input, then queries a database using an SQL statement and finally PatAn visualises the results as a chart and as a table.

Three XML schemas were developed for PatAn. PatAnML structures the analyses, PatAnQueryBaseML structures the one-time configuration of the SQL generation and PatAnChartBaseML the one-time configuration of the chart generation.

PatAn was written in the Java programming language. The Java programming language was chosen because it allows to use object-oriented design concepts. Moreover it assures the best integration of PatAn in the Healthgate applications of the Institute of Medical Technologies and Health Management of the JOANNEUM RESEARCH Forschungsgesellschaft m.b.H. (MSG) for which it was developed.

3.1. Analysis

In PatAn an analysis for a clinical patient data set consists of two main questions:

- What do we want to know?
- How do we present what we want to know?

Consequently, an analysis can be divided into two parts:

- a query part, which states what we want to know;
- a visualisation part, which states how the results of the query should be presented.

An analysis is passed to PatAn as an Extensible Markup Language (XML) document. The analysis XML document structure named PatAnML was defined using XML Schema. XML was chosen because it allows to structure and process data objects with minimal effort thanks to already existing applications and libraries. The structure – also known as XML schema – of the XML document reflects the division in query and visualisation sections by the two XML elements `<query>` and `<visualisation>` as child elements of the root element `<analysis>`. According to that an analysis XML document looks simplified like this:

```
<analysis>
  <query>
    ... definition of the query ...
  </query>
  <visualisation>
    ... definition of the visualisation ...
  </visualisation>
</analysis>
```

3.1.1. Query

A query defines which information should be extracted from a set of clinical patient data. Hence, during development of a query two questions arise:

- Which information should be extracted?
- What is the source data set for the query?

These two questions suggest the two main parts of a query:

- the specification of the information to be extracted;
- the definition of the source data set.

Analysis Types

After analysing already existing analyses in Healthgate BARS, it was possible to identify recurring patterns in the extracted information and in the corresponding SQL queries. For example the query on average alcohol consumption per patient and week follows the same pattern as the query on gender distribution in a set of patients. The pattern is the calculation of amounts of patients in different categories. In the former example categories are defined by different amounts of alcohol consumed per week and in the latter the categories are male, female and unknown (see figure 4). The identified recurring patterns led to the definition of different types of analyses. Throughout this thesis such types are called query or analysis types.

Currently three analysis types can be used in the XML element `<query>`:

- **categories** (`<categories>`): data categorised and counted;
- **average value** (`<averageValue>`): the average value and optionally the standard deviation of data is calculated;
- **counts** (`<counts>`): the elements in the analysed set are counted.

The analysis types can be seen as templates for developing queries.

Parametrized Set

A query is executed on a set of clinical patient data, where such a set consists of elements and elements contain attributes (see figure 5). Manipulating the set changes the extracted information in the result while the analysis type stays the same. For example the information about the gender distribution will probably be different for a set of patients older than 65 years and a set of patients younger than 65 years both suffering from diabetes mellitus (see figure 6). To define a set of clinical patient data to analyse, an XML element `<set>` was introduced in PatAnML to define a parametrized set.

A parametrized set is a set of data created by joining added sets to a basic and restricting it with parameters, called set parameters. A basic set defines the elements of a clinical patient data set (e.g. patients). It is defined once and can be used in multiple queries. Added sets contain additional element data in the clinical patient data set which is needed to extract the queried information (e.g. examination data). Joining added sets adds attributes to the elements of the basic set. Set parameters are like sieves which filter a set of clinical patient data (e.g. patient age, type of

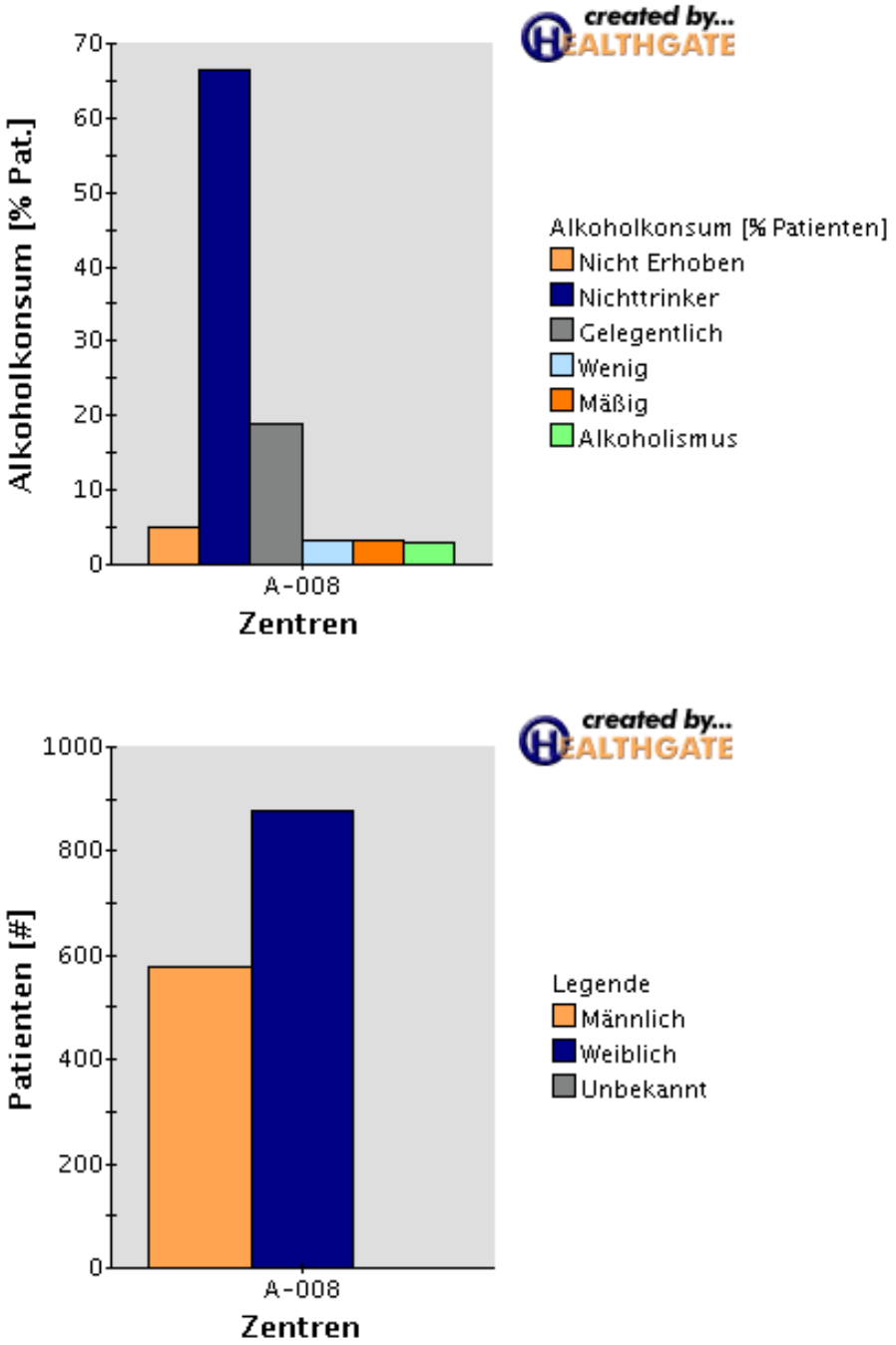


Figure 4: The chart on the top shows categories of alcohol consumption per week and the percentage of patients belonging to each category. The chart at the bottom shows the categories male, female and unknown and the absolute number of patients belonging to each category. Both charts refer to the same period of time and the same center. (Alkoholkonsum = alcohol consumption, Zentren = centers, Patienten = patients, Nicht Erhoben = not ascertained, Nichttrinker = non-drinker, Gelegentlich = occasionally, Wenig = little, Mässig = moderate, Alkoholismus = alcoholism, Legende = legend, Männlich = male, Weiblich = female, Unbekannt = unknown). The charts are taken from Healthgate BARS.

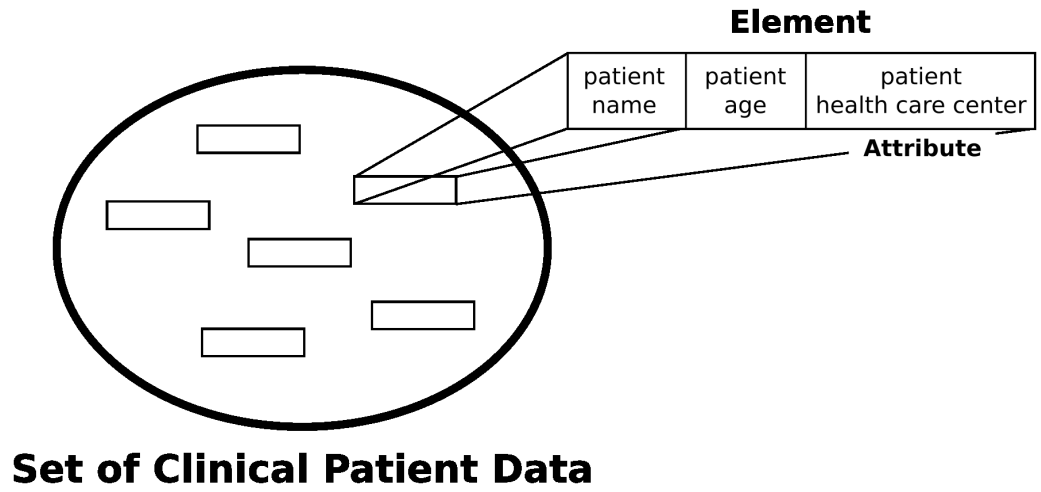


Figure 5: Example to illustrate the components of a clinical patient data set

disease and health care center). The concept of the parametrized set is illustrated in figure 7.

Each query must have a basic set. Added sets will be automatically joined to the basic set at runtime. Which added sets are joined depends on the query. For example if the basic set consists of elements containing only one attribute with unique patient identifiers and the query should extract the distribution of the gender, the set with the information about the gender will be joined to the basic set. If additionally the query should extract the distribution of the gender of patients of a specific center, the set containing this information will be joined to the basic set as well. The sets with the information about gender and centers represent added sets. The former is needed to extract the queried information, the latter is needed to restrict the set by a set parameter.

Set Parameters

Set parameters are used to restrict the set of clinical patient data to analyse by excluding elements which do not match dynamically modifiable conditions. There are different kinds of set parameters:

- simple set parameters,
- computed set parameters,
- complex set parameters,
- special set parameters.

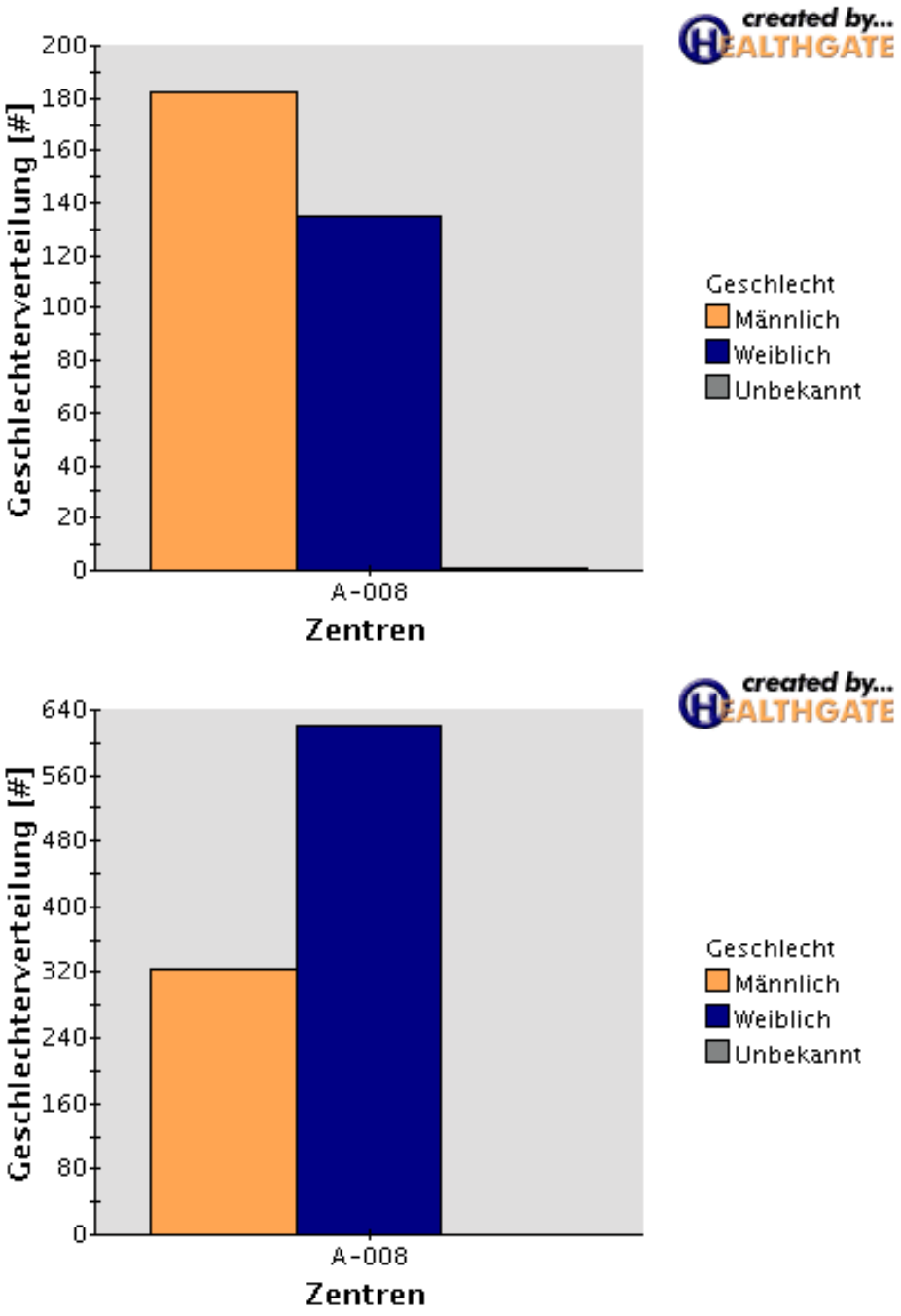


Figure 6: The chart on the top shows the gender distribution for a set of patients older than 65 years with diabetes mellitus. The chart at the bottom shows the gender distribution for a set of patients younger than 65 years with diabetes mellitus. Both charts refer to the same period of time and the same center. (Legende = legend, Männlich = male, Weiblich = female, Unbekannt = unknown). The charts are taken from Healthgate BARS.

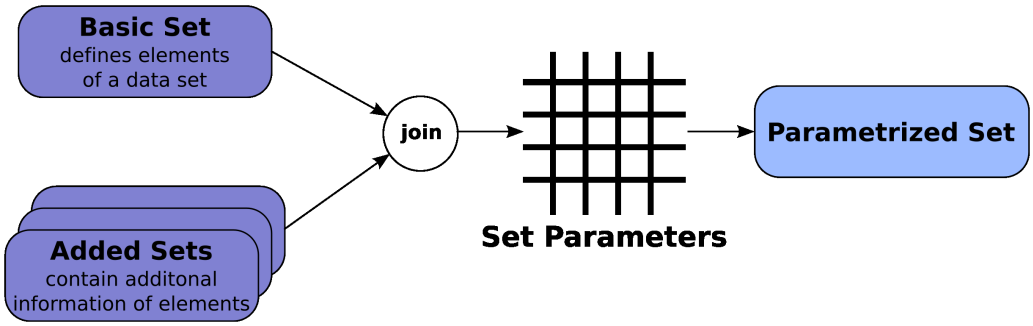


Figure 7: The creation of a parametrized set

A simple set parameter is used by PatAn to check if attributes contained in elements either equal an assigned value or are within an assigned range. All elements which do not match this logical expression are excluded from the set to analyse. For example if a set to analyse contains patients who belong to centers then a simple set parameter could restrict the set to patients of one center. Also the negation of the logical expression is possible. Additionally, multiple values and ranges can be assigned to the simple set parameter. In that case the resulting multiple logical expressions are connected through disjunctions. The assigned values and ranges are dynamically modifiable.

A computed set parameter is used by PatAn to check if a computation either equals an assigned value or is within an assigned range. Computations consist of attributes contained in elements, functions and constants. All elements which do not match this logical expression are excluded from the set to analyse. For example if a set to analyse contains elements representing patients with an attribute *years of birth* then a computed set parameter could compute patient age and restrict the analysed set to patients younger than 65 years. Computed set parameters allow to negate the logical expression as well. Multiple values and ranges are also possible. The assigned values and ranges are dynamically modifiable.

A complex set parameter is used by PatAn to check if elements of a set to analyse match a more sophisticated logical expression than in the above kinds of set parameters. All elements which do not match the logical expression are excluded from the analysed set. For example a set to analyse can be restricted to diabetes mellitus patients over 65 years suffering from hypertension. Complex set parameters can be negated as well. The logical expressions to restrict the set are dynamically chosen.

Special set parameters are set parameters which cannot be assigned to any kind of set parameters above. An example is a set parameter which ensures to analyse only

the latest patient data record. This task can not be accomplished with the above kinds of set parameters because it compares elements with each other. Also special set parameters are dynamically modifiable.

Set parameters can be different for every system in which PatAn will be used. A system concerning cardiac patients will not have a parameter *diabetes mellitus type* which is very important in a system concerning patients with diabetes mellitus. The static part of set parameters is defined in the system-specific configurations of PatAn.

Also the structure of dynamically modifiable components of the set parameters is different for every system. A set parameter in one system can allow multiple values whereas in another system it will not. These input structures of set parameters are defined in a system-specific XML schema, which is imported with its own namespace in the analysis XML schema PatAnML. The system-specific XML schema for the set parameters has to conform to certain structural constraints so that PatAn can read the parameters independently of the system. For example an element of a parameter which is defined containing multiple values can have any name, but it must have one or more child elements `<element>`.

Considering parametrized set and query type leads to the following analysis XML document:

```
<!-- root element of the analysis XML document with
     defintion of analysis XML namespace
<patan:analysis xmlns:patan="http://www.healthgate.at/PatAnML">
  <!-- query -->
  <patan:query>

    <!-- parametrized set -->
    <patan:set>
      <!-- basic set and set parameters
           with definition of the system-specific XML namespace -->
      <foo:set name="FooBasic" xmlns:foo="http://www.foo.com/FooSetML">
        <!-- patients' healthcare centers -->
        <foo:center>
          <foo:element>I-001</foo:element>
          <foo:element>I-010</foo:element>
          <foo:element>I-100</foo:element>
            ...
```

```
        </foo:center>
          ... more set parameters ...
      </foo:set>
        ... more set specific information ...
    </patan:set>

    <!-- categories-type as example for an analysis type -->
    <patan:categories>
      ... definition of the analysis type ...
    </patan:categories>
  </patan:query>
  <patan:visualisation>
    ... definition of the visualisation ...
  </patan:visualisation>
</patan:analysis>
```

Entities

A result of an analysis shows information about entities. Entities are things or beings which one wants to analyse. All elements in a clinical patient data set which have one or more characteristics in common can belong to an entity. Entities could be for example health care centers or patients. In the former case all elements of an analysed set which belong to the same center belong to the same entity *center* and in the latter all elements of an analysed set belonging to a patient belong to the same entity *patient*. To specify entities in the analysis XML document the XML element `<entities>` was defined in the XML schema. Entities can be distinguished by a single attribute of the elements of a set or by grouping elements using a logical expression. An example for the distinction by a single attribute may be entities containing patients with the same value for the center attribute. For the distinction by groups an example may be the definition of entities by a group of male patients over 65 years, a group of male patients under 65 years and a group with the remaining patients. The definition of entities leads to join added sets if the required data is not yet contained in the elements of the set to analyse.

Longitudinal Analysis

In PatAn the set of clinical patient data can be analysed over time as well. An XML element `<time>` was defined for this reason in the analysis XML schema. `<time>` specifies which attribute in the elements of a clinical patient data set holds the information about the time. Moreover, the time can be partitioned. This means, the representation of time in the result is not just limited to a sequence of dates as they exist in the clinical patient data set, but it is also possible to summarize dates in larger units like for example quarters of a year. The definition of the time may lead to join an added set if the required data is not yet contained in the elements of the set to analyse.

After introducing these two elements, the analysis XML document looks as follows:

```
<!-- root element of the analysis XML document with
     defintion of analysis XML namespace
<patan:analysis xmlns:patan="http://www.healthgate.at/PatAnML">
  <!-- query -->
  <patan:query>

    <!-- parametrized set -->
    <patan:set>

      <!-- basic set and set parameters
           with definition of the system-specific XML namespace -->
      <foo:set name="FooBasic" xmlns:foo="http://www.foo.com/FooSetML">
        <!-- patients' healthcare centers -->
        <foo:center>
          <foo:element>I-001</foo:element>
          <foo:element>I-010</foo:element>
          <foo:element>I-100</foo:element>
            ...
        </foo:center>
          ... more set parameters ...
      </foo:set>

      <!-- the entity to be analyzed -->
      <patan:entity name="center">
```

```
        ... definition of the entity to be analyzed ...
      </patan:entity>

      <!-- the time period in which the data should be analyzed
      <patan:time name="time">
        ... definition of the time period in which the
                              data should be analyzed ...
      </patan:time>

    </patan:set>

    <!-- categories-type as example for an analysis type -->
    <patan:categories>
      ... definition of the analysis type ...
    </patan:categories>
  </patan:query>
  <patan:visualisation>
    ... definition of the visualisation ...
  </patan:visualisation>
</patan:analysis>
```

Query Base

PatAn reads the analysis XML document and generates an SQL statement from it. To generate the SQL statement PatAn needs to know the structure of the relational database where the clinical patient data is stored and the database management system (DBMS) used. The structure information is needed to know how to join the sets to extract the queried information. The DBMS information is needed to generate the SQL statement in the right SQL dialect, since almost every DBMS has its own SQL dialect.

The definition of the basic sets, the static components of the set parameters, the structure of the database and the type of the DBMS used are the same for all queries of one system, therefore, they need to be defined only once. For this purpose another XML schema was developed named PatAnQueryBaseML. XML documents following PatAnQueryBaseML contain all the information which form the base for the queries of one system. In other words the query base XML document holds the configuration

for the SQL statement generation for one system. Such a query base XML document looks as follows:

```
<!-- query base root element with specified system and
     prefix and namespace URI of the specific
     schema used for the set parameters -->
<queryBase
  system="test"
  prefix="test"
  namespaceURI="http://www.healthgate.at/TestSetML">

  <!-- used DBMS -->
  <dataSource>MSSQL</dataSource>

  <!-- definition of the basic sets -->
  <basicSets>
    <!-- basic set of patients-->
    <basicSet name="patient">
      <!-- definition of basic set elements
           and corresponding joins -->
      <element>
        <attributes>
          <attribute alias="id">Patient.id</attribute>
          <attribute alias="age">Patient.age</attribute>
          <attribute alias="center">Center.name</attribute>
        </attributes>
        <joins>
          <join>
            <joinCondition>Patient.center = Center.id</joinCondition>
          </join>
        </joins>
      </element>

      <!-- definition of basic set paths
           from basic set to added sets -->
      <paths>
```

```
        <!-- path to the added set Exam -->
        <addedSet name="Exam">
          <basicSetLink>id</basicSetLink>
          <addedSetLink>patient</addedSetLink>
        </addedSet>
          ...
        <!-- more paths -->
          ...
      </paths>
    </basicSet>
      ...
    <!-- more basic sets -->
      ...
  </basicSets>

  <!-- definitions of set parameters -->
  <setParams>

    <!-- simple set parameter -->
    <setParam name="timeframe">
      <simpleParam>
        <attributeName>Exam.date</attributeName>
        <date>
          <range />
        </date>
      </simpleParam>
    </setParam>
      ...
    <!-- more set parameter -->
      ...
  </setParams>
</queryBase>
```

The attribute `system` of the root element `queryBase` denotes the system for which the query base is valid. The attributes `prefix` and `namespaceURI` specify the prefix and the namespace URI of the system-specific XML schema for the set parameter.

The element `dataSource` contains the DBMS used and consequently determines the SQL dialect to use. Then the basic sets and their paths to the added sets are defined. The paths contain the information how to join added sets to the basic set. In other words, paths define the structure of the used database. Finally, the static parts of the set parameters are defined.

3.1.2. Visualisation

In PatAn there are two main ways to visualise the results of a query. The results can be visualised as

- chart and
- table.

The charts for visualisation of the results are drawn by a charting component called chart factory which uses a specific chart library. A chart factory has a generic interface through which the type and the appearance of charts can be determined. At this time following chart types are available:

- bar chart
- stacked bar chart
- pie chart
- line chart

In the future the range will be extended. The appearance of the chart is customizable. The customizable components of a chart are backgrounds, axes, the legend to mention a few. The charts are created as Portable Network Graphics (PNG), but PatAn could be easily extended to produce additional formats as Scalable Vector Graphics (SVG), Portable Document Format (PDF) and JPEG. The implemented chart factory uses the chart library JFreeChart.

A result cannot be visualised with every chart type. For instance a pie chart cannot visualise a result with average value and standard deviation. A configuration which maps analysis types with some properties to allowed chart types is needed. This configuration is contained in the chart base. The chart base is specified through a XML document following the XML schema PatAnChartBaseML. The chart base XML document is system-specific. A fragment of a chart base XML document showing the mapping looks as follows:

```
<pcb:analysisToChartsMapping>

  <pcb:analysis>
    <pcb:categories>
      <pcb:summable>true</pcb:summable>
      <pcb:overTime>false</pcb:overTime>
    </pcb:categories>
  </pcb:analysis>

  <pcb:charts>
    <pcb:barChart/>
    <pcb:pieChart/>
    <pcb:stackedBarChart/>
    <pcb:lineChart/>
  </pcb:charts>

</pcb:analysisToChartsMapping>
```

So far the visualisation as table is only possible in Hypertext Markup Language (HTML), but other formats like Comma-Separated-Values (CSV) are imaginable. The appearance of the HTML table can be customized through table properties.

In the analysis XML document default visualisation properties can be optionally set. For this purpose an XML element `<visualisation>` was provided. Within the XML element `<visualisation>` properties for the visualisation as chart and as table can be specified.

3.2. Analysis Module Architecture

The analysis module architecture was created following object-oriented design concepts. Inheritance and polymorphism were used where possible and needed to increase the modularity of PatAn. Moreover some commonly used design pattern were implemented.

3.2.1. AnalysisFactory

The first object to create when using PatAn is the analysis factory of class `AnalysisFactory`. The analysis factory must be initialised with configurations and analyses.

A configuration consists of the following parts:

- query base (see section 3.1.1);
- chart base (see section 3.1.2);
- the resource bundles used for internationalisation;
- the chart factory which creates charts representing the results of analyses by using a charting library;
- the table factory object which creates tables containing the results of analyses.

The analyses are XML documents following the XML schema PatAnML as described in section 3.1.

The analysis factory can hold configurations and analyses for multiple systems. This means, it can be initialised with the configurations and analyses for a system containing clinical patient data about diabetes mellitus patients and hepatitis C patients. Within the analysis factory different systems are called subsystems. Additionally the analyses for such a subsystem can be split up in user-defined categories. This is useful if a system has analyses which refer to different topics. For example in a system like Healthgate BARS which checks the quality of medical care, analyses can refer to structural, process and outcome quality. These analyses can be split up in categories like *structure*, *process* and *outcome*.

The analysis factory follows the Singleton pattern. The Singleton pattern ensures that only one object of class `AnalysisFactory` is created at runtime.

3.2.2. Analysis

The method `createAnalysis` of the `AnalysisFactory` object creates an object of class `Analysis` from a specific analysis XML document. The analysis object contains all data and methods required to apply the analysis to the clinical patient data and visualise the corresponding results. It is the main object in processing the analysis. All tasks from query creation to visualisation start at the analysis object.

3.2.3. Query

To apply the analysis to the clinical patient data set, first of all an object representing the query has to be created with the method `createQuery` of the analysis object. The class of the created query object has to implement the interface `IQuery`. Since

the implementation of `IQuery` slightly varies dependent on the analysis type a class hierarchy shown in figure 8 was designed. The common code for all analysis types is contained in the abstract class `Query` whereas the code which is specific for an analysis type is contained in one of the following subclasses of `Query`:

- `CategoriesQuery` for the *categories* type
- `AverageValueQuery` for the *averageValue* type
- `CountsQuery` for the *counts* type.

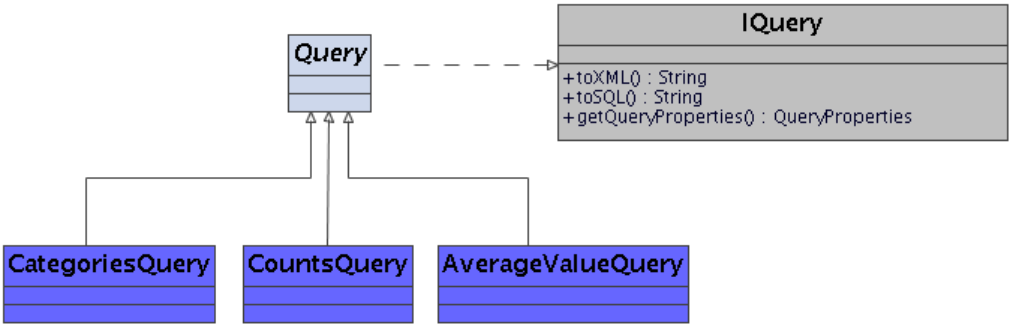


Figure 8: Hierarchy of query classes

The analysis type determines which of the children of `Query` will be instantiated.

If the developer using PatAn needs special functionality she can write a class which implements `IQuery` and put it in the package `healthgate.patan.query` and name it according to the pattern:

analysis type name + system identifier + `Query`

where the system identifier is the one specified in the query base. Each word of the pattern starts with an uppercase letter.

Before creating the query object the method `createQuery` looks for a class named corresponding to the pattern within the package. If there is such a class the method creates a corresponding object, otherwise it creates an object of the build-in classes mentioned above.

If the system is *diab* and the analysis type is *counts*, then the class for a query with special functionality created by the developer must have the name `CountsDiabQuery` and be contained in the package `healthgate.patan.query`.

The query object is used only inside the analysis object and is not visible outside.

3.2.4. Query Execution

Once the query object is created the query can be executed by calling the method `executeQuery` of the analysis object. The method `executeQuery` gets the SQL statement generated by the query object, executes it and stores the result set in a member field.

However, the actual execution of the query is performed outside the analysis object by the surrounding application. The surrounding application has to provide an implementation of the interface `IQueryExecuter` which executes the SQL statements and returns the result sets. This is needed because the SQL execution depends on the application architecture. For example in a J2EE application with Enterprise Java Beans (EJB) the database connection is managed by the application server, whereas in a stand-alone application the database connection has to be managed by the application itself. The implementation of the `IQueryExecuter` interface is passed to the method `executeQuery` as a method parameter.

To avoid the direct dependency on the SQL dialect of the DBMS used, the actual SQL code generation was encapsulated in classes. One class delivers SQL code for one SQL dialect. Such a class must implement the interface `ISQLDispenser`. At initialisation of a system in PatAn an object of the proper class for the DBMS type used specified in the query base XML will be created dynamically.

3.2.5. Result Dataset

After executing the query the results has to be transfered from the object returned by the implementation of `IQueryExecuter` to a PatAn-specific dataset object. The method `createDataset` of the analysis object accomplishs this task.

The created dataset object is a more generalised view of the result. It consists of a list of elements. An element contains the following attributes:

- analysed entity to which this element belongs;
- key which describes the meaning of the value;
- value retrieved from the clinical patient data;
- optionally time;
- number of elements analysed to retrieve the value.

In a *categories* type analysis on the alcohol consumption of patients of multiple centers the entity would be the name of a center, the key would be one of the defined categories and the value would be the amount of patients belonging to the category. The time is the moment or the time frame to which the element refers. The amount represents the number of patients analysed. In a *averageValue* type analysis the key describes if the value is either a average value or a standard deviation.

The generalised dataset allows some basic operations on the dataset as for example calculating the percent representation of the results for analysis of type *categories*. Since the operations on a dataset differ among analysis types and analyses with different properties, the hierarchy of dataset classes shown in figure 9 was designed.

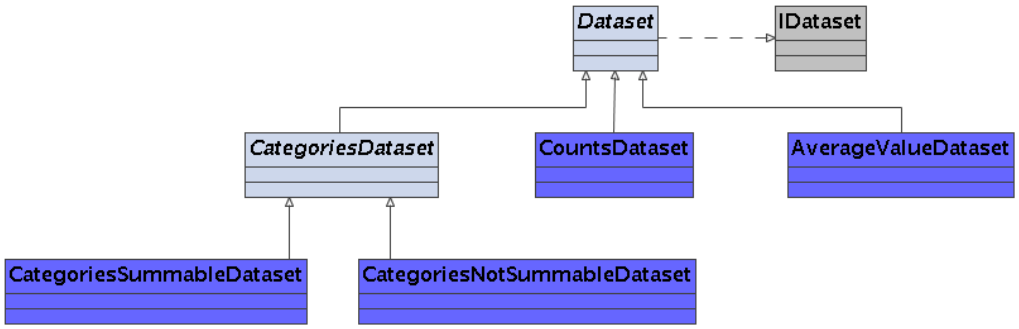


Figure 9: Hierarchy of dataset classes

All dataset classes extend directly or indirectly the class `Dataset` and implement the interface `IDataset`. There are two dataset classes for the *categories* analysis. One is `CategoriesSummableDataset` and the other is `CategoriesNotSummableDataset`. The reason for this distinction is the difference in calculating the percent representation of the results. For `CategoriesSummableDataset` the single categories sum to 100% whereas for `CategoriesNotSummableDataset` the categories do not.

The dataset object is – as the query object – used only inside the analysis object and is not visible outside.

3.2.6. Chart Creation

A chart in PNG format is created by calling the method `toPNG` of the analysis object. The parameters of the method are the dimension of the chart expressed as width and height in pixel and a Java `OutputStream` where to write the chart. The analysis object uses a chart factory which implements the interface `IChartFactory` to create a chart. A chart factory has a generic interface which is independent from the chart

type. It consists of the interface `IDataset` for the result dataset and the abstract class `ChartProperties` for the chart properties.

The chart properties are organised in different layers of abstraction resulting in the hierarchy shown in figure 10. `ChartProperties` is the abstract common ancestor of all chart properties classes. It contains properties valid for all chart types. Its subclasses contain more specific properties. The most specific classes of chart properties refer to a chart type. For example the background color is a property which belongs to all chart types, therefore it is contained in the common ancestor of all classes. The properties referring to axes belong to chart types which uses cartesian coordinates, therefore it is contained in the common ancestor of all chart types using cartesian coordinates (i.e. `SpecialChartProperties`). The chart properties class for the bar chart implicitly contains the information about the chart type which should be created in its class name and some properties too specific to be contained in its ancestor classes. The design of the chart properties allows to pass the common ancestor class `ChartProperties` to the charting factory, therefore the interface to the chart factory does not change dependent on the chart type.

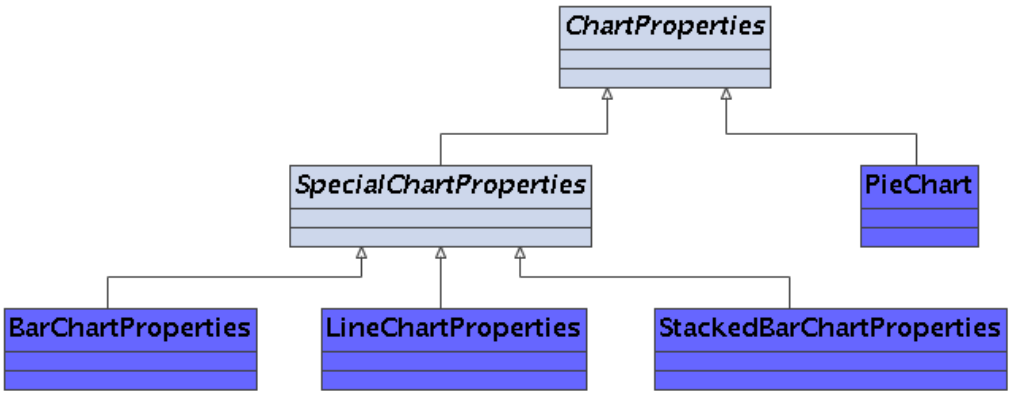


Figure 10: Hierarchy of chart properties classes

The generic interface is not specific to a charting factory so that it could be exchanged with another charting factory which understands it. The generic interface was designd by Daniel Laure [14] and improved during this work.

A chart factory is specific to a charting library. It is passed to the analysis factory object at initialisation. The chart factory reads the result dataset and the chart properties from its generic interface. It retrieves the chart type from the chart properties. Then the chart factory checks if the chart type is allowed with the given dataset. If the chart type is allowed, it maps the chart properties of the generic in-

terface to the chart properties of the charting library. Furthermore the chart factory transforms the result dataset in a form suitable for the charting library for further processing. The chart factory for the charting library JFreeChart in PatAn is of class `JFreeChartFactory`.

3.2.7. Table Creation

An HTML table containing the results is created by calling the method `toHTMLTable` of the analysis object. The analysis object creates an HTML table using a factory object called table factory similar to the chart factory. The class of the table factory implements the interface `ITableFactory`. `ITableFactory` has a method which creates the table for each table format. The method to create a HTML table is `writeHTMLTable`. Its parameters are the result dataset as `IDataset` interface, HTML table properties as an object of class `HTMLTableProperties` and the information how to transpose the table.

The HTML table properties contain the names of the Cascading Style Sheet (CSS) classes to use for formatting the table. CSS classes can be assigned to the cells containing attributes of the result dataset elements entities, keys, time, values and amounts (see section 3.2.5). Additionally a CSS class can be assigned to the header cells of each attribute.

The table can be transposed to differently display the results dynamically at runtime. For example the entities can be displayed in a column and after transposing the table as a row. Examples for transposed tables are shown in table 1 and table 2. The possible transpositions lead to either entities or keys or the time displayed in the first column of the table.

entities	**time**	**key1**	**key2**	**...**	**keyM**
entity1	time1	value111	value121	...	value1M1
entity1	time2	value112	value122	...	value1M2
⋮	⋮	⋮	⋮	⋮	⋮
entity1	timeN	value11N	value12N	...	value1MN
entity2	time1	value211	value221	...	value2M1
entity2	time2	value212	value222	...	value2M2
⋮	⋮	⋮	⋮	⋮	⋮
entity2	timeN	value21N	value22N	...	value2MN
⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮
entityL	time1	valueL11	valueL21	...	valueLM1
entityL	time2	valueL12	valueL22	...	valueLM2
⋮	⋮	⋮	⋮	⋮	⋮
entityL	timeN	valueL1N	valueL2N	...	valueLMN

Table 1: Transposed table regarding entities

keys	**time**	**entity1**	**entity2**	**...**	**entityL**
key1	time1	value111	value211	...	valueL11
key1	time2	value112	value212	...	valueL12
⋮	⋮	⋮	⋮	⋮	⋮
key1	timeN	value11N	value21N	...	valueL1N
key2	time1	value121	value221	...	valueL21
key2	time2	value122	value222	...	valueL22
⋮	⋮	⋮	⋮	⋮	⋮
key2	timeN	value12N	value22N	...	valueL2N
⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮
keyM	time1	value1L1	value2L1	...	valueLM1
keyM	time2	value1L2	value2L2	...	valueLM2
⋮	⋮	⋮	⋮	⋮	⋮
keyM	timeN	value1LN	value2LN	...	valueLMN

Table 2: Transposed table regarding keys

3.3. Example of an Analysis in PatAn

In the following an analysis of the gender distribution of two health care centers is presented.

The analysis XML document for PatAn:

```
<?xml version="1.0" encoding="UTF-8"?>
<patan:analysis
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:patan="http://www.healthgate.at/PatAnML"
    name="title.gender"
    i18n="true">

  <patan:query>
    <patan:set>
      <dmpddm2:set name="SheetCenterPatient" xmlns:dmpddm2="http://www.healthgate.at/DmpDDm2SetML">
        <dmpddm2:age>
          <dmpddm2:greater>
            <dmpddm2:limit>20</dmpddm2:limit>
          </dmpddm2:greater>
        </dmpddm2:age>
        <dmpddm2:center>
          <dmpddm2:element>DMPD-001</dmpddm2:element>
          <dmpddm2:element>DMPD-999</dmpddm2:element>
        </dmpddm2:center>
        <dmpddm2:bmi>
          <dmpddm2:less>
            <dmpddm2:limit>5</dmpddm2:limit>
          </dmpddm2:less>
        </dmpddm2:bmi>
      </dmpddm2:set>
      <patan:entities name="centres">
        <patan:attributeName>Center.sZentrumId</patan:attributeName>
      </patan:entities>
    </patan:set>
    <patan:categories>
      <patan:category name="female">
        <patan:contributionGroup contribution="1">
          <patan:logicalExpression>
            <patan:stringEquation>
              <patan:leftSide><patan:attributeName>Patient.sGender</patan:attributeName></patan:leftSide>
              <patan:rightSide><patan:value>W</patan:value></patan:rightSide>
            </patan:stringEquation>
          </patan:logicalExpression>
        </patan:contributionGroup>
      </patan:category>
      <patan:category name="male">
        <patan:contributionGroup contribution="1">
          <patan:logicalExpression>
            <patan:stringEquation>
              <patan:leftSide><patan:attributeName>Patient.sGender</patan:attributeName></patan:leftSide>
              <patan:rightSide><patan:value>M</patan:value></patan:rightSide>
            </patan:stringEquation>
          </patan:logicalExpression>
        </patan:contributionGroup>
      </patan:category>
    </patan:categories>
  </patan:query>

  <patan:visualisation>
    <patan:relative>true</patan:relative>
    <patan:chartProperties>
      <patan:defaultChartType>
        <patan:stackedBarChart/>
      </patan:defaultChartType>
    </patan:chartProperties>
  </patan:visualisation>

</patan:analysis>
```

The generated SQL statement from the analysis XML document:

```
SELECT ALL
  sum(
    case when
         (
           (Patient.sGender = 'W')
         )
         then 1.0
         else null
         end
  ) AS "female",
  sum(
    case when
         (
           (Patient.sGender = 'M')
         )
         then 1.0
         else null
         end
  ) AS "male",
  count(*) AS "total",
  Center.sZentrumId AS "centres"
FROM
(
  (
    SELECT DISTINCT
      Dmpddm2Patient.id AS "dmpddm2Patient",
      Patient.id AS "patient",
      BaseSheet.dBaDatum AS "dateOfExamination",
      Center.id AS "center",
      MetaObject.id AS "metaObject",
      BaseSheet.id AS "baseSheet"
    FROM
    (
      Patient INNER JOIN Center ON
      (
        Patient.center = Center.id
      )
      INNER JOIN BaseSheet ON
      (
        BaseSheet.patient = Patient.id
      )
      INNER JOIN MetaObject ON
      (
        MetaObject.patient = Patient.id
      )
      INNER JOIN Dmpddm2Patient ON
      (
        Dmpddm2Patient.metaObject = MetaObject.id
      )
    )
    WHERE
    (
      ((Center.sZentrumId = 'DMPD-001') OR (Center.sZentrumId = 'DMPD-999'))
      AND
      ((year(BaseSheet.dBaDatum) - Patient.nYearOfBirth) > 20)
    )
  ) AS "SheetCenterPatient"
  INNER JOIN "Center" ON
  (
    "SheetCenterPatient"."center" = "Center"."id"
  )
  INNER JOIN "Patient" ON
  (
    "SheetCenterPatient"."patient" = "Patient"."id"
  )
)
WHERE
(
  ((Dmpddm2Sheet.nAbGewicht / power(SheetCenterPatient.dmpddm2Patient, 2)) < 5)
)
GROUP BY
  Center.sZentrumId
```

The resulting chart:

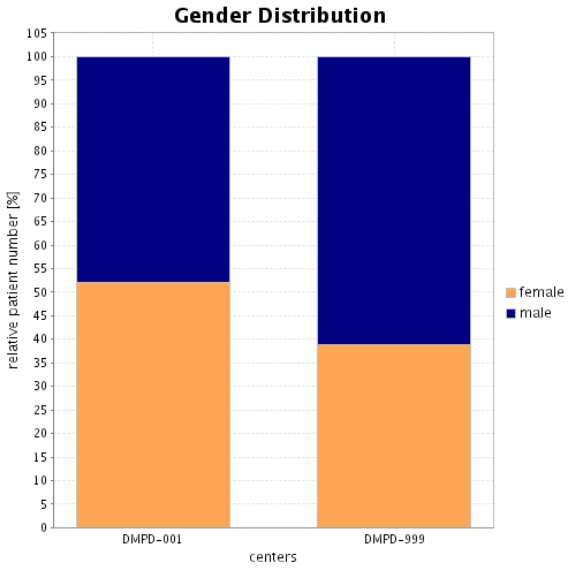


Figure 11: Chart representing the gender distribution of two health care centers

The analysis XML document is longer as the generated SQL statement. This could lead to the conclusion, that the analysis XML document is more difficult to develop and maintain, which is not true. If appropriate libraries and editors are given, XML documents are easier to develop and maintain as well as to modify programatically compared to plain SQL statements, because its modular structure simplifies processing.

4. Discussion

The developed data analysis module named PatAn handles analyses on a higher level of abstraction as plain SQL statements and the query XML schema used at the moment in Healthgate BARS presented in [13]. It generates the SQL statement which is executed on the relational database from an analysis XML document and hides its low-level aspects. This increases modularity and reusablity of the analysis process. Additionally, it visualises the results as customisable charts and tables.

4.1. Analysis structure

PatAnML and PatAnQueryBaseML reflect recurring patterns found in already existing queries in Healthgate BARS. These recurring patterns were generalised and made configurable. Hence the concept of basic sets dynamically joined at runtime with added sets was introduced. Through this concept it was possible to separate the database structure from the analysis, which allows to define the database structure once and use it in multiple analyses. This is a clear advantage over plain SQL statements and the query XML schema used in Healthgate which implicitly contain the database structure. However, the separation is not complete. An analysis represented as analysis XML document following PatAnML still contains table and column names referring to the underlying database, hence it is independent from the database structure but not from table and column names. In future a mapping file could map table and column names of the required data to variables used in the analysis XML document in order to be able to use the same analysis for different databases.

Parameters passed to analyses have to be dynamically inserted before execution. PatAnML collects all modifiable components of an analysis in one central place increasing its modularity. It simplifies and speeds up the dynamic modification of an analysis, because not the entire analysis XML document must be searched whereas with plain SQL statements the modifiable components are distributed over the whole query, therefore they are more difficult to process.

An analysis XML document following PatAnML contains information about the query and the default visualisation properties. Combining the query with the visualisation of its results is an extension to plain SQL statements and the query XML schema used in Healthgate BARS.

Using XML to structure the analyses allows to add information like access rights to an analysis XML document as it was already planned for the query XML schema

used in Healthgate BARS. Moreover XML documents can be validated with already existing tools and libraries. The validation reduces structural errors during the development of new analyses. However, PatAn could not be designed so that a XML validator could find all structural errors. For example an XML element containing child elements whose presence is dependent from each other is difficult or even not possible to define. A solution within XML could not be found during this work. At the moment all structural errors which cannot be found with XML validation are checked programmatically in PatAn.

4.2. Reusability

PatAn can be integrated in applications with different architectures and different database structures. During the development of PatAn special attention was paid to its reusability.

The aforementioned separation of database structure and analysis increases reusability, because changes in the underlying database structure only requires modification of the configuration in the query base instead of every single analysis XML document. Even if the table and column names additionally changed, the modifications in the analysis XML documents would be minor compared to the effort of modifying plain SQL statements.

The set parameters which allow to modify analyses by restricting the set of clinical patient data can be different for each system. If a system changes its set parameters or PatAn is integrated in a new system the structure and type of the set parameters can be changed in the query base. At the moment also the system-specific XML schema for the set parameters has to be changed, but in the future it could be generated from the information about the set parameters in the query base.

In the query base the database management system (DBMS) used must be specified. Dependent on the DBMS used an appropriate SQL dialect will be selected. PatAn reads the specified DBMS name and loads dynamically the corresponding SQL dialect dispenser object.

Applications with different architectures like Java 2 Enterprise Edition (J2EE) using Enterprise Java Beans (EJB) and stand-alone Java applications manage their database connections differently. PatAn leaves the task of executing the generated SQL statement to the surrounding application through a well-defined interface. This makes PatAn application architecture independent.

4.3. Development and Maintenance of Analyses

Handling analyses on a higher level of abstraction as plain SQL saves development time. The analysis types defined in PatAnML (see section 3.1.1) can be used as templates for new analyses. Filling in such a template with the required data for the analysis represents the configuration of the pre-implemented components resulting from the abstraction (see section 1.4.1).

The separation of database structure and analysis allows to copy analyses of one system and use it in another system after minor modifications. For example porting the popular patient's gender distribution analysis from system A to system B could change the corresponding SQL query completely. In PatAn just the table and column names in the analysis would change, assumed that the database structure for system B is already configured. Even if the database structure is not already configured, starting from the second analysis ported to system B, development time is saved, anyway.

The separation of database structure and analysis is an advantage especially if the database structure was designed according to the entity-attribute-value (EAV) approach [15]. Using the EAV approach different attributes are stored in table rows instead of table columns. Hence structure of stored data can be easier modified than with the traditional approach. However, SQL statements for such a database structure become more complex. Tables have to be joined in order to assemble data structures which can be queried with common SQL statements. PatAn is a middle layer between the EAV database structure and the user which allows automatic execution of the required preprocessing steps transparent to the user. So PatAn is an abstraction layer which can hide the EAV complexities from the user.

XML editors can be used to develop and maintain analyses. Some XML editors provide code-highlighting and code-autocompletion making development and maintainance of analyses more user-friendly. Examples for such editors are Oxygen XML Editor and XML Spy. The implementation of an application to develop and maintain analyses for PatAn is simplified through existing libraries and tools to process XML documents.

4.4. Visualisation

Putting query and visualisation together in an analysis XML document allows to define a default visualisation for the analysis in one single place.

The visualisation of results as customisable charts and tables increases the usability

of PatAn. The user can choose chart type and the form of the table as well as customise their appearence. Moreover, the results can be represented as absolute and relative values, and the total amount of analysed records can be displayed.

The developed generic interface for chart visualisation in PatAn allows to use a charting library by merely implementing a chart factory without changing other code. This simplifies using different charting libraries. A future extension to PatAn could be using multiple charting libraries in case the set of desired chart types cannot be provided by one single chart library.

To avoid senseless visualisations of results such as average values and standard deviations represented in a pie chart, a mapping of analyses with certain properties to chart types is possible.

4.5. Outlook

PatAn is already integrated in a Healthgate application for benchmarking of hepatitis C care centers. A new Healthgate BARS version will be released soon. It will not just contain diabetes mellitus patient data but also patient data of other chronic diseases like cardiovascular disease. PatAn will be used by this new Healthgate BARS version for open benchmarking.

In future, also other applications provided by the Institute of Medical Technologies and Healthmanagement of JOANNEUM RESEARCH Forschungsgesellschaft m.b.H. – like the Electronic Patient Record – will integrate PatAn to analyse data. Through the daily use of these applications new requirements will appear which will lead to further functionality and flexibility of PatAn.

4.6. Conclusion

The developed data analysis module PatAn allows handling analyses on a higher level of abstraction compared to plain SQL due to recurring patterns structured with XML. The developed XML structure increases modularity and reusability by pre-implementing configurable components of an analysis. PatAn combines query and visualisation in one analysis XML document which allows to define default visualisation properties for an analysis. PatAn saves development and maintainance time for analyses with similar or equal patterns as required by a medical quality management system. However, for very specific analyses which do not follow common patterns or for which no query type has been provided yet, one will have to resort to SQL.

A. Four-year Targets of the Saint Vincent Declaration

"Five-year targets:

- Elaborate, initiate and evaluate comprehensive programmes for detection and control of diabetes and of its complications with self-care and community support as major components.
- Raise awareness in the population and among health care professionals of the present opportunities and the future needs for prevention of the complications of diabetes and of diabetes itself.
- Organise training and teaching in diabetes management and care for people of all ages with diabetes, for their families, friends and working associates and for the health care team.
- Ensure that care for children with diabetes is provided by individuals and teams specialised both in the management of diabetes and of children, and that families with a diabetic child get the necessary social, economic and emotional support.
- Reinforce existing centres of excellence in diabetes care, education and research. Create new centres where the need and potential exist.
- Promote independence, equity and self-sufficiency for all people with diabetes children, adolescents, those in the working years of life and the elderly.
- Remove hindrances to the fullest possible integration of the diabetic citizen into society.
- Implement effective measures for the prevention of costly complications:
  - Reduce new blindness due to diabetes by one third or more.
  - Reduce numbers of people entering end-stage diabetic renal failure by at least one third.
  - Reduce by one half the rate of limb amputations for diabetic gangrene.
  - Cut morbidity and mortality from coronary heart disease in the diabetic by vigorous programmes of risk factor reduction.
  - Achieve pregnancy outcome in the diabetic woman that approximates that of the non-diabetic woman.

- Establish monitoring and control systems using state of the art information technology for quality assurance of diabetes health care provision and for laboratory and technical procedures in diabetes diagnosis, treatment and self-management.

- Promote European and international collaboration in programmes of diabetes research and development through national, regional and WHO agencies and in active partnership with diabetes patients organisations.

- Take urgent action in the spirit of the WHO programme, "Health for All"to establish joint machinery between WHO and IDF, European Region, to initiate, accelerate and facilitate the implementation of these recommendations."[16]

References

[1] *JFreeChart*. http://www.jfree.org/jfreechart/index.php, visited on 2006-03-11.

[2] *JUnit*. http://www.junit.org, visited on 2006-03-11.

[3] *179. Bundesgesetz, mit dem das Bundesgesetz über Krankenanstalten und Kuranstalten, das Allgemeine Sozialversicherungsgesetz, das Gewerbliche Sozialversicherungsgesetz, das Bauern-Sozialversicherungsgesetz, das Beamten-Kranken- und Unfallversicherungsgesetz, das Sozialversicherungs-Ergänzungsgesetz, das Ärztegesetz 1998 und das Bundesgesetz über die Dokumentation im Gesundheitswesen geändert sowie ein Bundesgesetz zur Qualität von Gesundheitsleistungen und ein Bundesgesetz über Telematik im Gesundheitswesen erlassen werden (Gesundheitsreformgsetz 2005)*, December 2004. http://www.bmgf.gv.at/cms/site/attachments/0/9/9/CH0329/CMS1104313005110/gesundheitsreformgesetz05.pdf, visited on 2006-03-11.

[4] Apache Software Foundation. *Apache Ant Project*. http://ant.apache.org, visited on 2006-03-11.

[5] Apache Software Foundation. *Apache XMLBeans*. http://xmlbeans.apache.org, visited on 2006-03-11.

[6] K. Beck. *Test-Driven Development*. Addison-Wesley, 2003.

[7] P. Beck. *Entwicklung einer sicheren, web-gestützten Client-Server Anwendung für Diabetes Management*. Master's thesis, Graz University of Technology, October 2000.

[8] T. Bray, D. Hollander, and A. Layman, editors. *Namespaces in XML, W3C Recommendation*. World Wide Web Consortium, January 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114, visited on 2006-03-11.

[9] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, editors. *Extensible Markup Language (XML) 1.0, W3C Recommendation*. World Wide Web Consortium, 3rd edition, February 2004. http://www.w3.org/TR/REC-xml, visited on 2006-03-11.

[10] B. Cadonna, P. Beck, I. Rakovac, T. Truskaller, and T. R. Pieber. Implementierung eines Moduls zur strukturierten und modularen Auswertung klinischer Patientendatensätze. In *Elektronischer Abstractband der 50. Jahrestagung der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie (gmds), 12. Jahrestagung der Deutschen Arbeitsgemeinschaft für Epidemiologie (dae)*. Deutsche Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie und Arbeitsgemeinschaft für Epidemiologie, German Medical Science e-journal, September 2005. `http://www.egms.de/en/meetings/gmds2005/05gmds386.shtml`, visited on 2006-03-14.

[11] Das Informations- und Fortbildungsprogramm für Qualitätsmanagement in der Ambulanten Versorgung. *Glossary*. `http://www.q-m-a.de/7sonstigeinfos/glossar/glossar/#Q`, visited on 2006-03-03.

[12] D. C. Fallside and P. Walmsley, editors. *XML Schema Part 0: Primer, W3C Recommendation*. World Wide Web Consortium, 2nd edition, October 2004. `http://www.w3.org/TR/xmlschema-0`, visited on 2006-03-11.

[13] F. Kirchmeir. *Development of a Java-based SQL query editor for analysis of disease management data*. Master's thesis, Graz University of Technology, January 2004.

[14] D. Laure. *Entwicklung einer Softwarekomponente zur Erstellung von Diagrammen durch eine einheitliche Schnittstelle in Java*. Master's thesis, Fachhochschule Technikum Klagenfurt, June 2004.

[15] P. M. Nadkarni, C. M. Brandt, and L. Marenco. WebEAV: Automatic Metadata-driven Generation of Web Interfaces to Entity-Attribute-Value Databases. *Journal of the American Medical Informatics Association*, 7(4):343–356, July/August 2000.

[16] NHS Scotland "Scotland's health on the web". *Saint Vincent Declaration*. `http://www.show.scot.nhs.uk/crag/topics/diabetes/vincent.htm`, visited on 2006-03-03.

[17] T. Pfeifer. *Qualitätsmanagement: Strategien, Methoden, Techniken*. Hanser, 2nd edition, 1996.

[18] H. C. Shah. *XML-Java Data Binding Using XMLBeans*. ONJava.com, July 2004. `http://www.onjava.com/pub/a/onjava/2004/07/28/XMLBeans.html`, visited on 2006-03-11.

[19] Sun Microsystems. *Java 2D API*. `http://java.sun.com/docs/books/tutorial/2d/index.html`, visited 2006-03-11.

[20] Sun Microsystems. *Java 2D FAQ*. `http://java.sun.com/products/java-media/2D/reference/faqs/index.html#xvfb`, visited 2006-03-13.

[21] Sun Microsystems. *Java Technology*. `http://java.sun.com`, visited on 2006-03-11.

[22] Union of Japanese Scientists and Engineers. *The Deming Prize*. `http://www.juse.or.jp/e/deming/03.html`, visited on 2006-03-10.