Martin Pirklbauer

Design and Development of a Database for Tissue Micro Array Experiments

Master Thesis

Institute of Biomedical Engineering,
Graz University of Technology
Krenngasse 37
A-8010 Graz, Austria
Head of Institute: Univ.-Prof. Dipl.-Ing. Dr. techn. Gert Pfurtscheller

Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. techn. Zlatko Trajanoski
Advisor:	Dipl.-Ing. Gerhard Thallinger

Graz, September 2003

Martin Pirklbauer

Entwurf und Entwicklung einer Datenbank für Tissue Micro Array Experimente

Diplomarbeit

Institut für Elektro- und biomedizinische Technik,
Technische Universität Graz
Krenngasse 37
A-8010 Graz
Vorstand: Univ.-Prof. Dipl.-Ing. Dr. techn. Gert Pfurtscheller

Begutachter:	Univ.-Prof. Dipl.-Ing. Dr. techn. Zlatko Trajanoski
Betreuer:	Dipl.-Ing. Gerhard Thallinger

Graz, September 2003

Diese Arbeit ist in englischer Sprache verfaßt.

für meine Mutter

for my mother

Abstract

Tissue microarrays (TMA) provide a method for high-throughput molecular profiling of tissue specimens (e.g., cancer tissue). Up to 1000 specimens can be analysed in a single experiment. Data generated during TMA production, processing and evaluation has to be maintained to ensure experiment reproducibility and analysis. For this purpose a database application and a web interface for tissue microarray experiments was implemented.

The application was implemented based on a three-tier architecture using state-of-the-art Java technologies. Java-based technologies present a number of advantages, such as platform independence and a proven enterprise component architecture.

In addition, a web client was developed to provide easy access to the database application. Besides the standard maintenance functions, an integrated query editor allows to create custom queries in order to retrieve data in a flexible way.

Currently, basic image analysis algorithms are integrated in the application. In the future more sophisticated algorithms will be added. New analysis algorithms can be embedded into the applications business logic using a plug-in mechanism, easily configured in the web application.

Keywords: Tissue microarray, high-throughput molecular profiling, biomedical informatics, Java three-tier architecture, databases

Kurzfassung

Tissue Microarrays (TMA) ist ein Verfahren für Hoch-Durchsatz Untersuchungen von (z.B.: Krebs-) Gewebeproben. Bis zu 1000 Gewebeproben können dabei in einem einzigen Experiment untersucht werden. Die erzeugten Daten, welche bei der Produktion, Verarbeitung und Auswertung anfallen, müssen gesammelt und gewartet werden, um die Reproduzierbarkeit von Experimenten und Analysen zu garantieren. Zu diesem Zweck wurde eine Datenbankapplikation und eine Webschnittstelle für Tissue Microarray Experimente entwickelt.

Die Applikation wurde basierend auf der 3-Stufen Architektur konzipiert und mit Hilfe der neuesten Java Technologien implementiert, um eine flexible und erweiterbare Anwendung zu erhalten. Auf Java aufbauende Technologien sind vorteilhaft, da sie Plattformunabhängigkeit gewähren und technisch ausgereifte Unterstützung von serverseitigen Komponentenarchitekturen (Enterprise-Technologie) zur Verfügung stellen.

Um möglichst einfachen Zugang zur Datenbank zu erlangen, wurde eine Weboberfläche entwickelt. Ein integrierter Abfrageeditor bietet die Möglichkeit benutzerdefinierte Abfragen zu erstellen, um Daten und Ergebnisse einfach und flexibel aus der Datenbank abfragen zu können.

Elemtentare Bildverarbeitungsalgorithmen zur Auswertung der TMAs sind in der Applikation integriert. In Zukunft werden neue Analysealgorithms entwickelt werden, welche fortgeschrittener und ausgereifter sein werden. Diese können mit Hilfe eines Plug-In Mechanismus einfach in die Applikation hinzugefügt und über die WEB Oberfläche konfiguriert werden.

Schlüsselwörter: Tissue Microarray, Molekularanalysen mit großem Durchsatz, verteilte Java Softwarearchitektur, Bioinformatik, Datenbanken

Contents

List of Figures

Chapter 1

Introduction

1.1 Brief overview of tissue microarray technology

The concept and use of tissue microarrays (TMAs) for high-throughput molecular profiling of tumour specimen, such as cancer, was first introduced in 1998 [4], and has received a significant amount of attention from the research community ever since. This new technique was originally developed at the Cancer Genetics Branch of the National Human Genome Research Institute (NHGRI) in Bethesda, USA, in collaboration with the University of Basel, Switzerland, and is now being increasingly used throughout the world.

The underlying profiling technology is similar to other existing molecular profiling methods, but the use of TMAs and a special experimental arrangement enable high-throughput molecular analysis across *hundreds* of tissue samples at once and in a single experiment. In this new approach, a large number (e.g., 500-1000) of cylindrical core tissue biopsy samples is first extracted from the original tissue specimen, and then deposited into a recipient paraffin block, using a rectangular arrangement that makes it easy to track each sample individually. Such a tissue microarray block is then further sliced into many tissue microarray sections, each containing to several hundred tissue samples. This arrangement makes it possible to perform a wide range of analyses, such as immunohistochemistry or in situ hybridisation, on a large number of specimen at once. Construction of multiple TMA blocks from the same original tissue specimen can further scale this method up to several thousands of tissue sections being profiled concurrently. Fig. 1.1 shows the basic principle of TMA technology.

The construction of tissue microarrays starts with the extraction of tissue samples from morphologically representative regions of a set of paraffin-embedded tissue blocks, called *donor blocks*. In

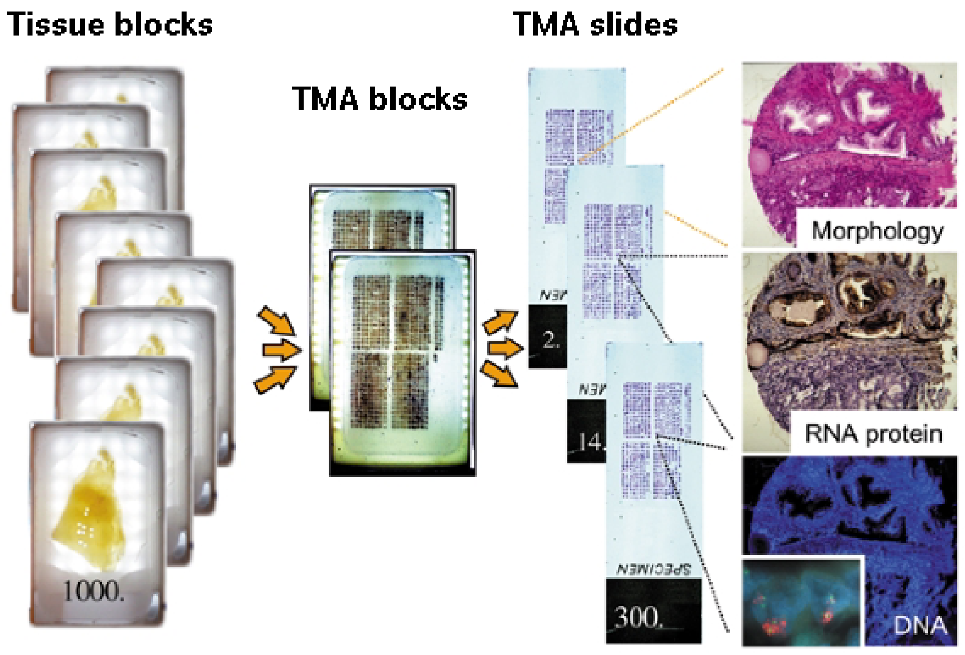
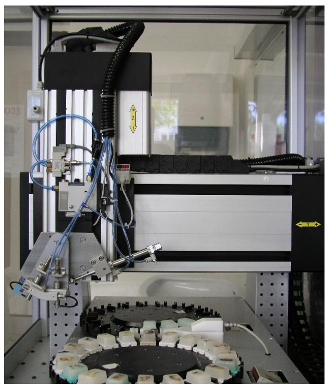


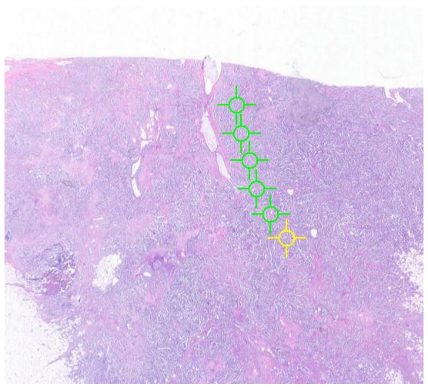
Figure 1.1: Principle of Tissue microarray (TMA) analysis: Cylindrical core biopsies are obtained from paraffin-embedded tissue blocks. These are transferred to a TMA block. Multiple TMA blocks can be generated at the same time. Each TMA block can be sectioned up to 300 times. All resulting TMA slides have the same tissues in the same coordinate positions. The individual slides can be used for a variety of molecular analyses, such as hematoxylin and eosin (HE) staining to discover tissue morphology, mRNA in situ hybridisation (ISH) or protein immunostaining or analysis of genetic alterations using fluorescence ISH (FISH). [3]

this process, a set of core tissue biopsy samples, with a diameter of 0.6-0.8 mm and height of 2-3 mm each, are taken from the individual donor blocks and then deposited into a new recipient paraffin block, called *array block*, which have a size of about 45 x 20 mm (see Fig. 1.2(c)), using a tissue "microarraying instrument". In order to capture enough histological information, the cylinders holding the core tissue samples should have a minimum diameter of 0.6 mm (which is considered ideal). This way, up to 1000 specimens can be lined up in an arrayblock, without significantly damaging the original donor blocks. The donor block is manually positioned for sampling based on a visual alignment with the corresponding hematoxylin and eosin (HE) stained section on a slide. The region of interest for punching in each tissue is carefully selected from the HE-stained slide. Finally, a microtome, a precision instrument designed to cut uniformly thin sections of a variety of materials for detailed microscopic examination, is used to cut the resulting tissue microarray block into 2-4 μm-wide sections.

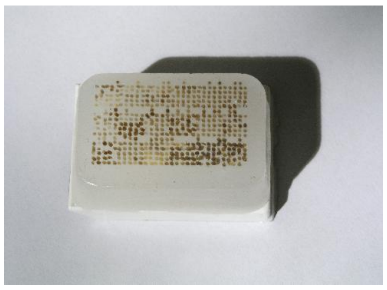
On average, about 200 slides containing tissue samples can be cut from a single tissue microarray block, see Fig. 1.3.



(a)



(b)



(c)

Figure 1.2: TMA Photos: Fig. 1.2(a) shows the TMA Arrayer, which constructs the array blocks. Fig. 1.2(b) illustrates how the positions of punches are chosen from the hematoxylin and eosin (HE) stained donor block section and Fig. 1.2(c) illustrates an arrayblock constructed by the TMA Arrayer.

Once constructed, TMAs can be used for a range of molecular analyses that can also be performed on regular tissue sections, such as immunohistochemistry (IHC), fluorescence *in situ* hybridisation (FISH) and messenger ribonucleic acid (mRNA) *in situ* hybridisation. In general, the same hybridisation and staining protocols as for sections obtained using traditional methods can be used [5]. Analysis and scoring of TMA slides can be carried out with regular microscopes. Therefore, pathologists can easily perform these tasks on up to several hundreds of tissue spots per hour, without the need for any special or more expensive instruments. [3]

To be able to perform high-throughput analysis, researchers also create digital images of the slides

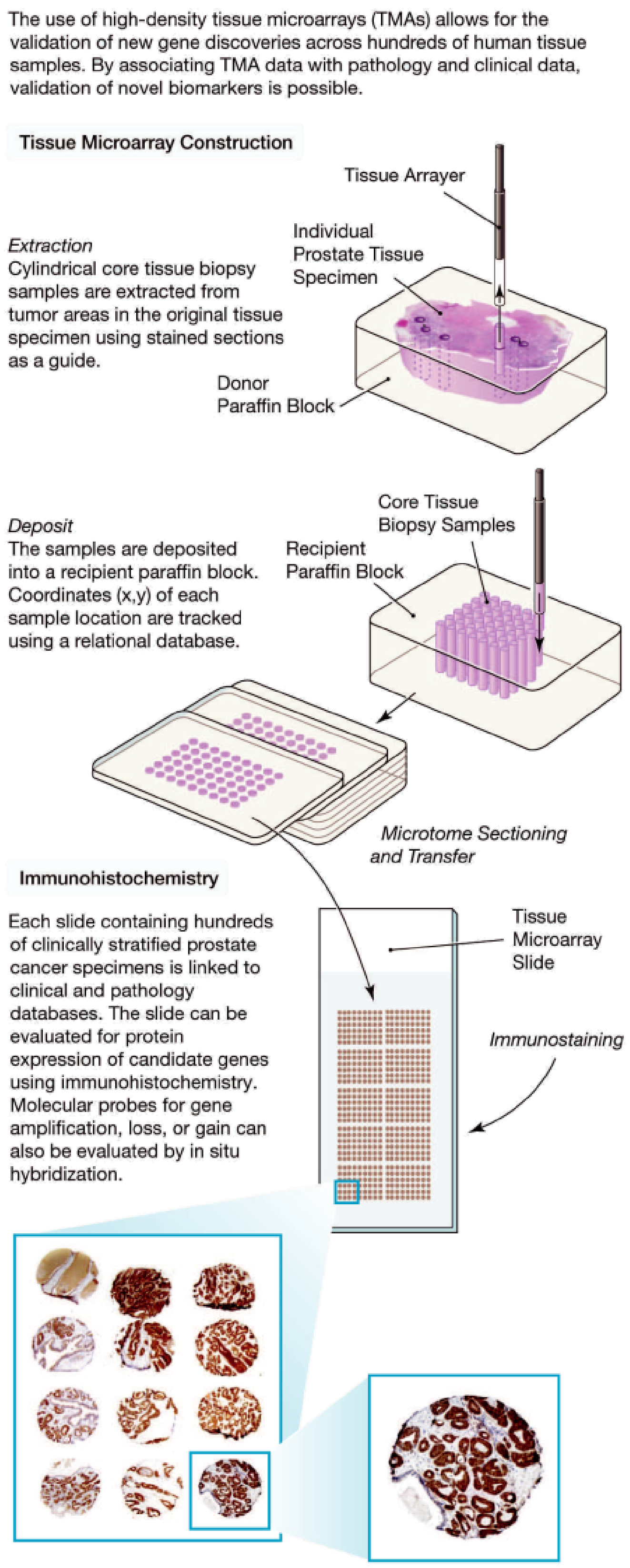

The use of high-density tissue microarrays (TMAs) allows for the validation of new gene discoveries across hundreds of human tissue samples. By associating TMA data with pathology and clinical data, validation of novel biomarkers is possible.
Tissue Microarray Construction
Tissue Arrayer
Individual Prostate Tissue Specimen
Extraction
Cylindrical core tissue biopsy samples are extracted from tumor areas in the original tissue specimen using stained sections as a guide.
Donor Paraffin Block
Core Tissue Biopsy Samples
Recipient Paraffin Block
Deposit
The samples are deposited into a recipient paraffin block. Coordinates (x,y) of each sample location are tracked using a relational database.
Microtome Sectioning and Transfer
Immunohistochemistry
Each slide containing hundreds of clinically stratified prostate cancer specimens is linked to clinical and pathology databases. The slide can be evaluated for protein expression of candidate genes using immunohistochemistry. Molecular probes for gene amplification, loss, or gain can also be evaluated by in situ hybridization.
Tissue Microarray Slide
Immunostaining

Figure 1.3: The TMA block is constructed by depositing core tissue samples into a recipient paraffin block, which is sliced up to 300 times. Each slide can be evaluated using molecular analysis. [2]

(Fig. 1.4), typically by using a laser scanner (see Fig.1.4(d)).

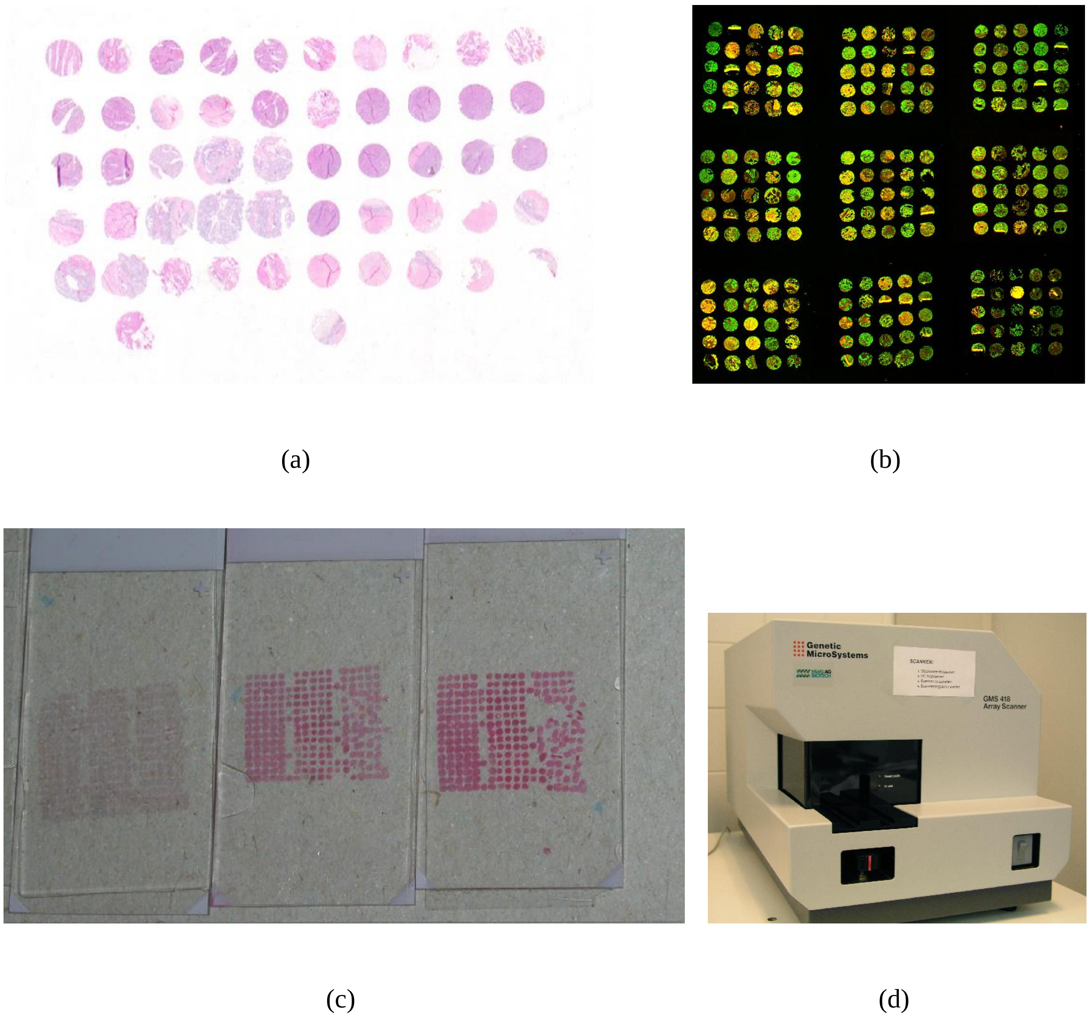


Figure 1.4: TMA slides and scanner: Fig. 1.4(a) shows a HE stained TMA slide, in Fig. 1.4(b) a double stained (red-green) TMA slide is illustrated, scanned with a laser scanner (Fig. 1.4(d). Fig. 1.4(c) shows some plates with TMA slides.

In sum, tissue microarray technology enables the rapid visualisation of molecular targets in thousands of tissue specimens at once, analysing a variety of molecular structures, such as DNA, RNA (with some restrictions) and proteins. [3]

TMAs are not an alternative to conventional microarray techniques, because high-density cDNA Microarrays enable analysis of gene expression levels of thousands of genes and proteins at once, but rather complement them by providing additional and more detailed information on a very large set of tissue samples (e.g., cancer tissues). For instance, tissue microarrays can be extensively used to validate gene targets that have been found in cDNA microarrays analysis.

1.2 The need for a database application for tissue microarray experiments

As outlined above, TMA technology essentially consists of depositing a set of tissue biopsy samples into array blocks, and subsequently cutting these array blocks into a large number of slides. In order to be able to track each sample, and perform *high-throughput* profiling on many or all of them, a methodical approach to store and process the large amount of data involved is required.

So far, several research groups have built tools to store and process TMA related data, but most of these consist of relatively simple spreadsheet models ([6]) that only work in a specific environment, or in combination with proprietary applications. In some cases, these tools have been combined with image analysis tools to obtain more precise results than by using a regular microscope ([1]). Furthermore, these tools generally do not meet the basic workflow requirements of a typical TMA process. Usually, many people are involved in TMA construction and analysis, often at different times with defined handoff points.

Extensive studies of literature and interviews revealed that to date no adequate academic or commercially available system exist to mange tissue microarray experiments and the related management process in an efficient and fully satisfying way.

The conception of this thesis therefore is to design and build a database-driven computer application that:

- Allows to manage TMA experiments in an efficient, scalable and sharable way.
- Responds to the workflow and synchronisation requirements that are typical in a TMA production and analysis environment.
- Establishes a defined vocabulary and exposes open system interfaces, so as to reduce the dependencies on specific or proprietary tools, and to render the system more widely usable.

Tissue microarray technology is considered to be a high-throughput method, and many persons are typically involved in the TMA construction process. Therefore it is clear, that anybody dealing with TMA slides within a working group should follow a certain protocol, so experiments can be exactly reproduced.

The idea of this thesis is based on the obvious requirements of synchronising workflow in the TMA production. By enforcing a defined interface, vocabulary and certain experimental protocol, the system ensures that the results can be shared and reproduced in a consistent and reliable fashion. Only with a well defined vocabulary, a consistent way of producing, storing and retrieving data and the results of different analysis steps, it is possible to make the work transparent to others within the workgroup and to other groups of pathologists and biologists.

Chapter 2

Goals and Objectives

As stated earlier, the overall goal of this thesis is to design and develop a database for tissue microarray experiments and its related data, as well as to provide a web-based application and user interface to access this database. From a functional perspective, the application should allow users to easily *produce* tissue microarray images, *store* them in a relational database for later *retrieval* and *processing*.

More specifically, the application is designed to meet the following objectives:

Support for the overall TMA process: It should assist users across the entire TMA process and life cycle, to avoid many of the drawbacks mentioned above. The application should guide users through all the steps necessary for constructing TMA slides, and should have a web based interface. As almost everybody today is familiar with a web browser, little to no detailed IT knowledge is needed to use the system. Furthermore, people working on the same project do not need to be bound to a specific location. Entering data, uploading images and other common tasks can be done from any workstation connected to the Internet.

Extensibility: It should be possible to integrate external applications easily. A popular example is the "gridding algorithm", which extracts images of individual punches from a section image, for later use and processing by other analysis algorithms. The system should be flexible enough to add new (or replace existing) application modules at any time, to generate new results with them, and to use these results within other (existing) modules.

User access control: There should be a flexible user and access control mechanism that grants different levels of rights to different users, based on defined criteria (e.g., qualification, role in the

TMA process).

Easy database access: Users should be able to browse through stored data easily, and a special query should also allow more complex web-based queries on the database.

Minimal need for user interaction: It should be possible to start multiple analyses of TMA images “in the background” (i.e. simultaneously), without the need for further user interaction. This *asynchronous mode* of operation is key to fully exploit the high-throughput approach.

A user requirements document should describe the main features of the tissue microarray database application. It should target multiple audiences involved in the TMA process, and establish a common terminology, to be used by researchers, scientists and pathologists alike.

The entire system should be implemented in the Java language. One of the best-known features of Java is its platform independence. As the underlying application and web server are also written in Java, the system as a whole is platform independent and can be installed on top of nearly every state-of-the-art computer operating system in existence today.

Chapter 3

Techniques and Methods used to build the TMA database application

This chapter will review commonly used techniques and methods of applied computer science for development of large-scaled database applications including its clients. Most of these concepts are well described in the literature ([39],[13],[35]), only a brief overview of the concepts relevant to the design and development of the TMA database application for high-throughput analysis of tissue microarrays and to achieve the objectives outlined in Chapter 2 is provided here.

To build a robust and reliable database the, system consists of three major components, which are outlined in more detail below:

- The database,
- the application logic (running on an application server) and
- a web based client interface.

The database is designed to store all data related to the TMA production process, as well as the results of different analyses applied to stored images. Furthermore, it should be easy to integrate new algorithms, and to add database capacity for the results they produce.

The application logic of the database application is encapsulated into an application server, whose purpose is to “serve” multiple clients connected to it. The application server constitutes the middle tier of a three-tier architecture and should run the bulk of the code of the database application, such as server-side analysis algorithms as well as the database access routines.

End users can access the application through a “client” program which interfaces with the “server” through a defined interface, while delegating most of the application logic to the server. Clients access the database only through the application server. This way, any additional operations on the database, such as pooling or persistence mechanisms are hidden from the client or the end users.

To enable easy access to the TMA database, the client is web-based. No installation of a client application is necessary, and the TMA database is accessible from anywhere on the Internet. This web-based client application manages all interactions with the end user, and communicates with the application server that runs the application logic (e.g., converts database queries to HTML format). It also contains the user management and access control mechanisms for the database.

In the following sections, first the Java 2 Enterprise Edition (J2BE) framework (3.1.1) and the Enterprise JavaBeans (EJBs) component technology (3.1.2) are described in general. EJBs constitutes the core components of the implemented TMA application logic. Second Java Servlets (3.2.1) and JavaServer pages (3.2.2) are introduced, which are used to build the web-based client interface, which is based on the Struts framework (3.2.4), a light-weight technology for the creation of web clients.

3.1 Developing the application logic for the TMA process

3.1.1 The Java 2 enterprise edition (J2EE) for designing multi-tier enterprise applications

The J2EE platform enables and employs a multi-tier distributed application model. Application logic is divided into components according to function, and the various application components that make up a J2EE application can be installed on different machines, depending on the tier in the J2EE environment to which the application component belongs.

The J2EE framework offers the following advantages for the development of the TMA database application:

- The implementation of the presentation code and the routines for the application logic can be completely separated, only the interfaces between them need to be defined.
- Each server component, as described in more detail below, can be maintained independently and can in fact be distributed to different server machines (if so desired or needed).

- The J2EE framework itself contains software to handle commonly used tasks, such as database connection pooling or instance pooling of J2EE components, which enhances performance of the TMA database apparently, and frees the application developer from implementing this functionality.

Figure 3.1 shows a basic model of a multi-tier J2EE application, divided into three tiers.

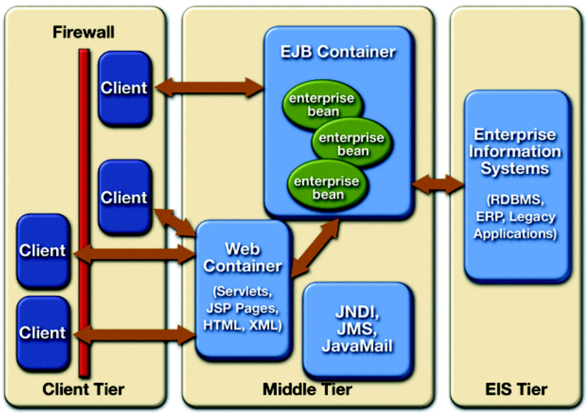


Figure 3.1: A typical J2EE Environment [26]

The client tier components run on the client machine, which is typically a desktop computer.

The middle tier components run on the J2EE server which in turns is based on an application server.

The enterprise information tier (EIS) or data tier is usually a (relational) database management system.

J2EE applications consist of components. A J2EE component is a self-contained functional software unit that is integrated into a J2EE application, together with its related classes and files, and that communicates with other components. The J2EE specification, as described in [39] defines the following J2EE components:

- Application clients and applets are components that run on the client.
- Java Servlets and JavaServer pages (JSP) technology components are Web components that run on the J2EE server or on a separate web server.
- Enterprise JavaBeans (EJB) components (enterprise beans) are application components that run on the application server.

3.1.2 Enterprise JavaBeans for defining application components within J2EE

Written in the Java programming language, an *Enterprise JavaBean* (EJB) is a server-side component that encapsulates the application logic of an application. The application logic is the code that fulfils the purpose of the application. EJBs run within the context an application server, which provides a runtime environment and manages important tasks for the EJBs, such as life cycle management and persistence mechanisms. As EJBs often imply a lot of unnecessary overhead, they are typically not used for small-scale, or single-user applications.

But EJBs can simplify the development of larger and/or distributed software applications. Because the EJB container provides a relatively complete set of commonly used system-level services, such as transaction management and security services, the enterprise bean developer can focus on solving the business problems at hand, without having to worry about these low-level system services. A further advantage is that the enterprise beans and not the clients contain the application logic allowing the client developer to concentrate on the presentation of the data to the end user. For instance, the client developer does not have to code the routines that implement specific application rules or the database access. As a result, the clients are “thinner” (i.e. contain less code), a benefit that is particularly important for clients that run on small devices. Thin clients are preferred in general, because they are easier to update and avoid network traffic on their best behaviour

Given that enterprise beans are portable components, an application developer can rapidly “assemble” new applications from existing beans. If properly defined, applications can run on any J2EE-compliant application server.

Large applications are often organised in multi-tier architectures. A typical scenario is a three-tiered web-based deployment, as illustrated in Fig. 3.2.

A three-tier architecture is usually designed as follows ([37]):

The presentation tier runs within one or more web servers. It typically consists of Java Servlets, additional scripts to customise the applications look-and-feel (such as JavaServer pages, etc.), and some workflow logic.

The application logic tier runs within one or more *application servers*. Application servers are now commonplace. Many established software companies offer their version of a J2EE-compliant application server (e.g., IBM Websphere, BEA Weblogic, and Sun ONE), but there are also

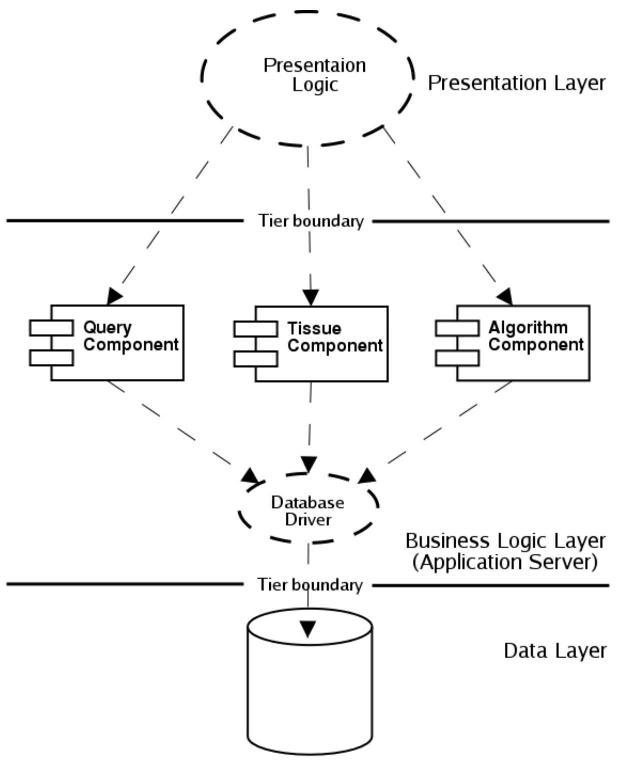


Figure 3.2: A typical scenario for a 3-tier architecture [37]

open-source implementations available for free (e.g., JBoss, JOnAS, and OpenEJB). Application servers are considered necessary to provide a suitable runtime environment for the application logic components. The application server is responsible for managing these components efficiently, and for providing a number of services to the components. For example, an application server offers a database access layer for use by the application components, allowing them to make data persistent and to load data from the data tier into the application. The application server is also responsible for instantiating application components as necessary.

The data tier consists of one or more databases and may also contain data related logic in the form of stored procedures.

There are three different types of enterprise JavaBeans defined in the EJB Specification[1]. [13], which are summarised in table 3.1.

In the sequel, each part is described in more detail. In the TMA database application described in this document, *Session beans* are used to provide input/output routines for the clients communicating

[1]The current stable release is EJB v2.0, but there also exist a “Proposed Final Draft”- specification for EJB v2.1, which will be released towards the end of 2003

Enterprise Bean Type	Purpose
Session Bean	Acts as agents to the client and is controlling workflow.
Entity Bean	Represents a application entity object that exists in persistent storage
Message Driven Bean	Acts as a listener for the Java Message Service API, processing messages asynchronously

Table 3.1: Types of Enterprise JavaBeans

with the application logic, to perform complex operations on the data generated during the TMA production and analysis process, and to hide database specific operations from the client. *Entity beans* were selected to encapsulate database elements into Java Objects, and a *message driven bean* was chosen to invoke integrated analysis algorithms in an asynchronous way. Each of these three types of Enterprise JavaBeans have their specific properties, which made them most suitable for the tasks described. Some of these type specificities are described below:

Session beans

A *session bean* represents a single client inside the J2EE server. To access an application that is deployed on the server, the client invokes the session bean's methods. The session bean then performs work on behalf of the client freeing it from additional complexity by executing the invoked tasks inside the server.

Unlike entity beans (see 1), session beans do not represent shared data in the database, but are able to access shared data. This means, session beans can be used to read, update, and insert data.

Session beans are divided into two basic types: stateless and stateful. [36]

Stateless Session Beans: A stateless session bean is a collection of related services, each represented by a method; the bean maintains no state from one method invocation to the next. When a method on a session bean is invoked, it executes the method and returns the result without knowing (or caring) whether other requests have been issued before or might follow. A stateless session bean can be seen as a set of procedures or batch programs that execute a request based on some parameters and return a result. Stateless session beans tend to be general-purpose or reusable, such as a software service.

Stateful Session Beans: A stateful session bean is an extension of a client application. It performs tasks on behalf of the client and also maintains state related to that client. This state is called *conversational state* because it represents a continuing conversation between the stateful session bean and the client. Methods invoked on a stateful session bean can write and read data to and from this conversational state, which is shared among all methods in the bean. Stateful session beans tend to be specific to one scenario.

Because tasks needed for the TMA database application are self-contained and therefore need not to manage a conversational state stateless session beans are used to implement these tasks.

Entity beans

An *entity bean* represents an application object in a persistent storage mechanism. Some examples of application objects are customers, orders, and products. In the J2EE SDK, the persistent storage mechanism is a relational database. Typically, each entity bean has an underlying table in a relational database, and each instance of the bean corresponds to a row in that table.

Because the state of an entity bean is saved in a storage mechanism, it is persistent. Persistence means that the entity bean's state exists beyond the lifetime of the application or the J2EE server process. The data in a database is persistent because it still exists even after the database server or the applications it services are shut down.

There are two types of persistence for entity beans: *bean-managed persistence* and *container-managed persistence*. [8]

Bean-managed persistence (BMP) For BMP the bean developer is entirely responsible for all the synchronising steps to connect the entity beans with the database. All the necessary SQL statements and JDBC calls must be coded by the programmer. The advantage of this kind of persisting entity beans is the full control over all actions pertaining the database, allowing an access optimisation. Additionally, tasks, which require several database access operations can only be implemented by using the bean-managed persistence mechanism.

Container-managed persistence (CMP) For CMP the container supplies the full synchronisation between the database and the entity layer. The container ensures the consistency and integrity during the beans entire lifetime. The developer does not care about how the beans access the database, but it is important that the underlying database is relational in nature. CMP is typically used for simple data access operations.

A typical entity object has the following characteristics ([13]):

- It provides an object view of data in the database.
- It allows shared access from multiple users.
- It can be long-lived (lives as long as the data in the database).
- The entity, its primary key, and its remote reference survive a crash of the EJB Container. If the state of an entity was being updated by a transaction at the time the container crashed, the entity's state is automatically reset to the state of the last committed transaction. The crash is not fully transparent to the client; the client may receive an exception if it calls an entity in a container that has experienced a crash.

Because the data related the TMA process, and the results of the TMA analyses need to be persistent, entity beans were chosen to implement them. As data operations on TMA data are simple, the TMA database application only uses the container-managed persistence mechanism.

EJB QL - Enterprise JavaBeans query language The enterprise JavaBeans query language defines the queries for the finder and select methods of an entity bean with container-managed persistence. EJB QL queries are defined in the deployment descriptor of the entity bean. Typically, a tool will translate these queries into the target language of the underlying data store. Because of this translation, entity beans with container-managed persistence are portable; their code is not tied to a specific type of data store.

EJB QL was introduced in version EJB 2.0 and will be expanded in v2.1. For complete documentation of the EJB QL it is referred to the EJB specification [13], and to the J2EE tutorial [17].

Using EJB QL increases portability of the application server components, because the components get more independent from the database access driver and the database itself. In the TMA application EJB QL is used for this purpose, but is not used for very complex database operations using database specific functionality, which can not be implemented with EJB QL.

Message driven beans

A message-driven bean is an enterprise bean that allows J2EE applications to process messages asynchronously. Since the TMA database features some time consuming tasks like the analysis of the individual punches of a complete array, this type this type of enterprise bean is used to allow the user to invoke integrated analyses "in the background" mode.

A message-driven bean acts as a Java Messaging Service (JMS) message listener, which is similar to an event listener except that it receives messages instead of events. The messages may be sent by any J2EE component (an application client, another enterprise bean, or a Web component) or by a JMS application or system that does not use J2EE technology. [19]

A typical message-driven object has the following characteristics ([13]:

- It executes upon receipt of a single client message.
- It is asynchronously invoked.
- It supports transactions
- It may update shared data in an underlying database.
- It does not represent directly shared data in the database (although it may access and update such data).
- It is relatively short-lived.
- It is stateless.
- It is removed when the EJB Container crashes. The container has to re-establish a new message-driven object to continue computation.

3.1.3 The JBoss application server

JBoss is the highly popular, free J2EE compatible Open Source application server. JBoss provides "JBossServer" (or "EJB/JBoss"), the basic EJB container, and Java Management Extension (JMX) infrastructure. JMX is a powerful interface for integration of software. Support for web components, such as Java servlets and JavaServer pages is provided by this integration layer. Due to JMX JBoss has a completely modular server design, which simplifies management of both JBoss/Server components and the applications deployed on it. EJB/JBoss is an EJB 2.0-compliant application server and is continously improved. [18]

The JBoss application server has been chosen to develop the TMA application, because for all practical purposes it is a viable alternative to commercially available application servers (e.g., IBM

Websphere, BEA Weblogic, and Sun ONE) and an adequate solution to achieve all objectives for the TMA database.

3.1.4 Summary of technologies for designing the application logic

The *Enterprise JavaBeans* (EJB) standard is a component architecture for deployable server-side components in Java. It is an agreement (in the form of an interface specification), between components and application servers that enables any component to run on any application server. EJB components (enterprise beans) are deployable, and can be loaded into an application server.

The application server provides commonly used base services, such as persistence and state management, so that the bean developer can concentrate on application specific business logic.

In sum, application servers and EJBs are used to build the TMA application because they:

- provide easy database access,
- allow to extend the application with new algorithms easily and flexible,
- enable the system to run high-throughput analysis *without* user interaction
- separate application logic from the client, allowing it to be accessed from anywhere on the Internet,
- are freely available and widely used.

3.2 Designing the interface for TMA end users

3.2.1 Java servlets to build Java-based web components

Java Servlets are part of the middle tier in the J2EE three-tier environment and are running within a so-called servlet container. A servlet is a Java-based web component. The main purpose of Java servlets is to allow the dynamic generation of web content, while an application is running. They are pure server-side components and can improve server-side web development substantially. The detailed specification can be found at [12].

In the TMA application data changes frequently and new data can be added by any number of users. Therefore Java servlets, in combination with JavaServer pages (see section 3.2.2), are ideal for generating a representation of the data stored inside the database.

Servlet container

The *Servlet container* is part of a web- or application server and holds the Java servlets described above. In addition, it provides services for handling client requests and for processing the output from servlets (response). The container is also responsible for servlet administration and guidance through their life cycle. A servlet container is either integrated in a separate web server, as for instance in Jakarta's Tomcat web server, or is an extension to another web server (e.g., the Apache web server). The container must implement at least the HTTP protocol in order to meet the specification, but HTTPS (HTTP over SSL) is typically provided as well. [9]

Servlet life cycle

Each servlet has a well defined life cycle. It is controlled and supervised by the servlet container. To work properly, the servlets have to implement either the *javax.servlet.Servlet* interface, the basic servlet interface provided by the servlet software bundle, directly or indirectly via the abstract classes *GenericServlet* or *HttpServlet*. The 3 most important methods of the interface are ([22]) init(), service(), and destroy().

Fig. 3.3 shows the life cycle of a servlet. The state changes from *available for service* to *servicing request* when the *service()* method is called. One parameter of the service() method is the request object. While the servlet processes the request it changes to state *unavailable for service* from where it changes to state *available for service* after finishing. A detailed documentation of the servlet life cycle can be found in [22].

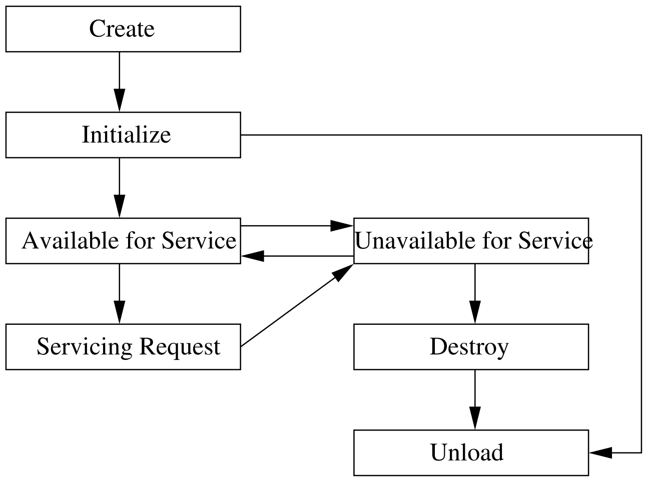


Figure 3.3: Life cycle of a Java Servlet

To summarise, servlets are pure Java components for generating dynamic web content. Servlets run in a container and have a well-defined life cycle. The javax.servlet software package contains several important functions useful for handling servlets, some of which were described. A full package description can be found in [27], and many tutorials dealing with Java servlets are available in the world wide web (e.g., at [10] or [22]).

Although Java servlets offer excellent capabilities, there is also an important disadvantage. Java Servlets do not satisfy the requirement for separation of content and presentation components. To remedy this limitation, a new technology based on Java servlets was introduced as part of the J2EE framework - JavaServer pages (JSPs) with their tag libraries, which is described in the next section.

3.2.2 JavaServer pages (JSPs) to enable the creation of dynamic web sites

This section gives a brief overview of JavaServer pages (JSPs) which is a server side technology that is also part of the J2EE framework. JSPs allow quick development and easy maintenance of *dynamic* web sites and supports many different development techniques. Within a JSP, Java code can be mixed with HTML code, it is possible to use JavaBeans easily, and, in addition, custom-defined tags can be used.

As JavaServer pages separate presentation and content, they are key to the web based interface of the TMA application. With JSPs the appearance of the TMA web application can easily be changed and customised later on, without requiring detailed knowledge of the underlying application logic and data structure.

JavaServer pages are text files that contain standard HTML elements and new scripting tags. JSPs are set up like HTML. After creation they are converted into servlets which are compiled on-the-fly (i.e. when they are called for the first time). The resulting servlet contains pure HTML elements and dynamic JSP code. [9]

The JSP container

Figure 3.4 shows a typical web application server, consisting of several layers (web server, servlet container and JSP container). [9]

The usage of a servlet container in the middle layer (i.e. between the JSP container and the web server) has several advantages:

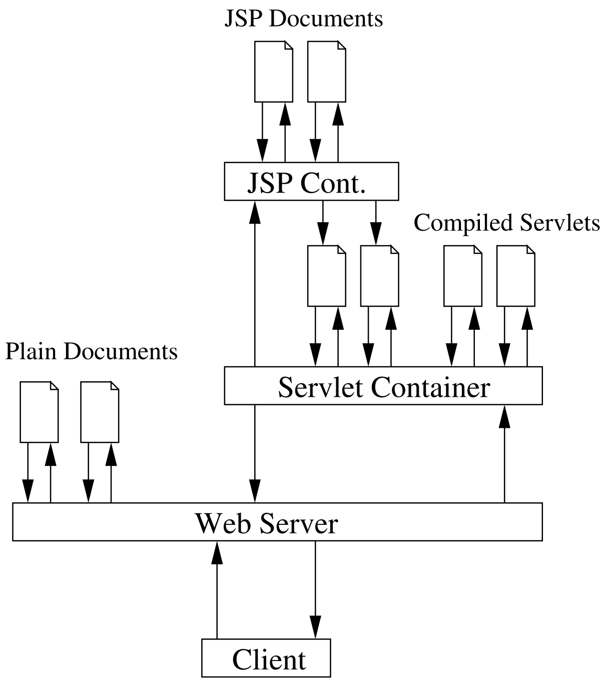


Figure 3.4: Layer structure of a typical Web Application Server

Standardised API: Developers of web applications have to learn only one single application programming interface (API), namely the servlet API. The servlet API is a relatively small extension to J2EE and can be learned quickly.

Layer separation: A JSP container is build up on the servlet API. The developer does not need to understand the code contained it the servlet container. This reduces the complexity for implementing a JSP container, while it is possible to test the various layers separately.

Code reuse: Developers can re-use code from servlets or other classes in their JSPs and vice versa with great ease.

The web server recognises the extension .jsp inside a web page reference (URL). A requested resource with this extension can therefore be identified as a JavaServer page and be passed on to the JSP processing engine ([9]), where the JSP is transformed into a Java class which is subsequently compiled to a Java servlet. This phase of transformation and compilation is invoked upon the first request of the client to this resource or if the content (code of the web page) changes. Subsequent requests are faster, because the web server caches the Java byte code inside its memory.

Basic structures of JSPs

As mentioned above, JSPs consist of standard HTML code and dynamic script parts. The dynamic part of a JSP is build using JSP elements. Major JSP elements include JSP directives, JSP scripting elements, standard actions and JSP tags for which a brief overview is given here. A more detailed description can be found in [20].

JSP directives for initialisation and declaration

JSP directives are messages from a JSP page for the JSP container. They are used to set global values, valid inside a whole JSP page, e.g., class declarations, output content types and so on. Directives generate no output for the client. They are characterised by the @ symbol. The general syntax of a directive is ([20]):

```
<%@ directive-name attribute="value" attribute="value" %>
```

There are three directives for JSPs: *page*, *include*, and *taglib*. They are discussed in detail in [9].

JSP scripting elements to include pure Java code

Scripting elements integrate script code into a JSP page. They allow declaration of variables and methods, integration of code and evaluation of expressions ([20]). There are three different scripting elements:

Declarations: A declaration is Java code, which resides inside a JSP page and it defines class wide variables and methods. Declarations are initialised when the JSP page is initialised and they are valid in the whole servlet class. Each declaration is defined by enclosing it with the special characters <%! and %>.

Scriptlets: A scriptlet is a Java code sniplet, executed when a request is processed. It is enclosed between the characters <% and %>. The challenge of scriptlets depends on the particular code. The scriptlet can write something into the output stream, which is directed to the client. Scriptlets can, of course, change objects which are in scope or valid inside the scriptlet. Several scriptlets are combined and compiled in the order as they appear in the JSP page.

Expressions: An expression is a short form of a scriptlet. It evaluates an expression and writes the result to the output stream, directed to the client.

An expression is enclosed by the characters <%= and %>. If a part of an expression is an object, it is converted to a character string by calling the toString()-method on this object.

Standard actions for workflow control

Actions are specific tags, which have effects on the runtime behaviour and on the response, sent to the client. Standard actions provide functionality for the page writer and have to be implemented by each container provider. Some standard actions are: <jsp:useBean:>, <jsp:setProperty:>, <jsp:getProperty:>, <jsp:include:>. A complete list can be found in [9].

3.2.3 JSP tags for including custom functionality

As stated earlier it is possible to use JavaBeans in JSP pages. Beans usually encapsulate application logic. Standard actions enables the writer to use this beans, but for more complex operations scriptlets are necessary. Separation of logic and presentation is lost and the JSP code becomes inhomogeneous and difficult to read. Custom JSP tags allow provide an easy to use interface for complex operations.

How custom tags are created

For using a custom tag in JSP pages, three steps are necessary. First a Tag Handler Class has to be implemented, second this class has to be connected to a custom tag, and third, the tag has to be used in a JSP page. [20]

Tag Handler Class: The tag handler class implements the functionality for the custom tag. This class must at least implement the interface *Tag*, which is defined in the package *javax.servlet.jsp.tagext*.

The Tag Library: The tag library is usually an XML file and has the extension .tld. This file defines all custom tags related to a specific library. Each library has it's own XML document. Each Tag is defined in a <tag>entry and must have at least the following body tags: <name>, <tagclass>, <bodycontent>, <info>, <attribute>. [20]

Including the Tag Library: The tag library is announced to the JSP container by using the *taglib* directive. After that, the custom tags are known inside the JSP page.

For the TMA application a JSP tag library for access control (provided by the integrated user management) is used. This tag library provides a special JSP tag (`<login:checkLogin/>`), which is used inside each JSP to check the correct authentication and authorisation of a logged in user.

To summarise, JSPs allow to include dynamic content into a web page. Four major groups of elements were briefly outlined:

- *Directives* for initialisation and declaration
- *Scripting elements* to include pure Java code
- *Standard actions* for work-flow control
- *JSP tags* for including custom functionality

Because it is useful to produce reusable code it was shown how to write modular code with custom tags. Numerous collections of already existing taglibs as well as further links to other resources are shown in [33] and [24].

The complete JavaServer pages specification can be found at [32].

3.2.4 The Jakarta Struts framework for building web interfaces

The Jakarta Struts framework is used for the TMA application to improve both the usability of the web interface and the web development process itself. Although the Struts framework does not introduce fundamentally new technologies, it offers several advantages:

- It provides automatic web form validation and error handling,
- A Struts-based web application can be easily internationalised,
- It provides additional flexibility.
- The Struts framework uses commonly-used Java technologies and has therefore gained widespread use in the development community.
- It optimises the development process, and
- It is generally useful for developing multi-tier web applications.

For these reasons, the Struts framework is an ideal fit for the development of the TMA application. Some elements of this framework are described in more detail below.

The framework is relatively small and implements the Model-View-Controller pattern (MVC). Struts combines the Java Servlet and the JavaServer pages technologies, and provides additional custom tag libraries. Struts therefore helps to create an extensible development environment for Java web applications, based on published standards and proven design patterns. This section will focus on the three main parts the this framework, the model, the view, and the controller. Figure 3.5 illustrates how the three components work together.

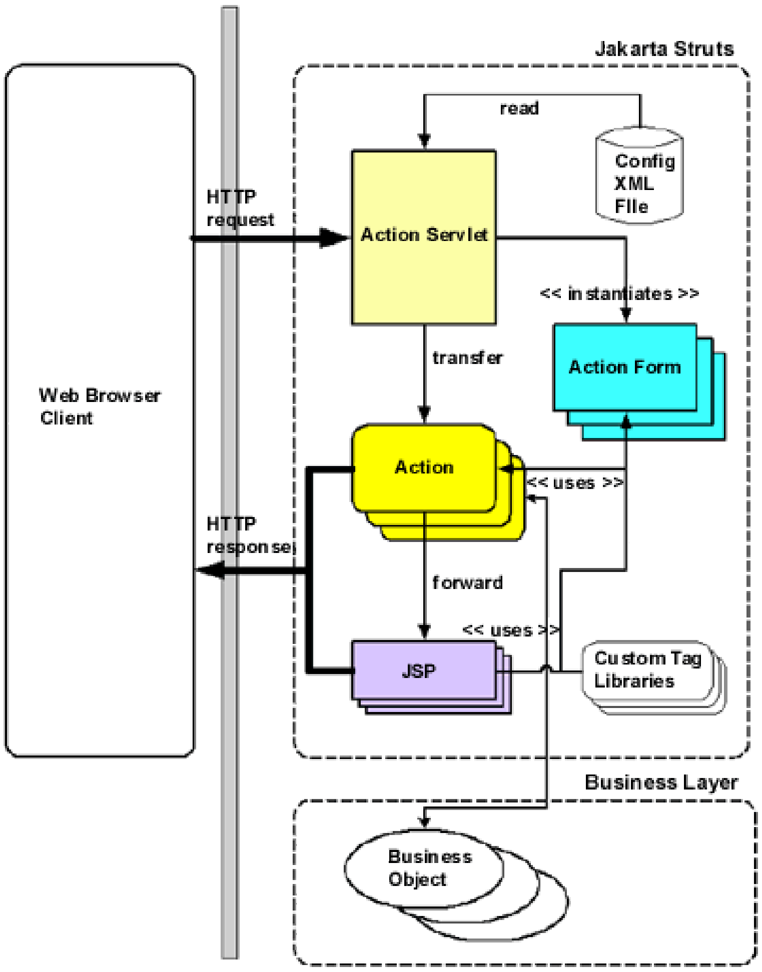


Figure 3.5: Jakarta Struts: The action servlet and the actions represent the controller (yellow) in terms of the Model-View-Controller pattern; the ActionForms represent the model (cyan), while the JSPs belong to the view (purple). [35]

Requests from a client (typically an end user using a web browser) are received and handled by a so-called *Action servlet*. The Action servlet acts as a controller, and decides which function from the application logic should be invoked. This servlet is configured and driven by defining a set of mappings (described in a class of type ActionMapping). Each mapping defines a fully qualified class name of the action class and a path, on which a incoming request-URI is checked. The action class (i.e. the class, which extents the “Action” class) implements the desired action of the application logic.

The view components in Struts based applications are usually constructed with the JavaServer pages technology (see above). In addition, an extensive tag library is included in the Struts framework. These tag libraries allows the creation of user interfaces, which support internationalisation, and interact well with the ActionForm-Beans, which are also part of the model. In addition to JSP pages containing actions and custom tags, it is often necessary for application objects to be able to transform themselves into HTML (or XML) code at run-time. The output that is rendered from such an object is easily inserted into the resulting JSP page via the standardised <jsp:include>- action tag.

The model in a MVC-based system is partitioned into two areas: One is the internal state of the system, the other are the actions used for changing the internal state. The components *ActionForm* and *Business Object* in Fig. 3.5 are thus part of the model.

The Model

Most of the Struts objects are designed as JavaBeans. Although JavaBeans[2] were first created for visual elements, their object design patterns have been found to be generally useful for building any reusable components, such as those used by the Struts framework. The actual state of a web application is also contained in JavaBeans.

Within a web application, JavaBeans can be bound to one of four different *collections*. A *collection* defines the rules for their lifetime and scope. Four different scopes are possible:

Page: The beans are visible within a single JSP page for the lifetime of the current request.

Request: The beans are also visible in pages to which it is forwarded from the current page.

Session: The beans are visible in all JSP pages, associated to a certain user session.

Application: The beans are visible for all JSP pages, that belong to a certain web application

[2]Even if they have similar names, JavaBeans [38] and Enterprise JavaBeans [13] are different technologies.

The View

The view is the representation of the system state (i.e. the model) to the user. View components are mostly build with JSPs, but the Struts framework contains some useful extensions that simplify the development process significantly including internationalisation for providing multiple languages, (i18n), Form- and FormBean-interaction for error handling, presentation and custom tags for building and controlling the visual representation of the view.

Internationalised Messages: Struts builds upon the standard classes available on the Java platform to build internationalised and localised applications. The key concepts to become familiar with are Locale, ResourceBundle, PropertyResourceBundle, MessageFormat and MessageResource.

To build an internationalised application, a property file for each language must be created. The property files contain a set of key-value pairs. Only the keys are inserted in the view components, and Struts automatically takes the proper value, depending on the browsers language settings, and replaces the key with the corresponding value. A full step-by-step documentation for building internationalised applications can be found at [34].

Forms and FormBean interaction: Struts supports a powerful mechanism for form validation. If a user enters some data into a form element, that does not conform with defined requirements (e.g., “a password must have at least 7 characters”) Struts can generate an error page, where the user is prompted to correct the field. Requirements are defined in the validate()-method of the ActionForm class. An ActionError object is generated if the validation fails. A special tag (`<html:errors/>`) in the JSP page checks the existence of such an error object and generates an error page, if one exist. Further details of this feature can be found in the Struts user guide [35].

Presentation tags: Four presentation tag libraries are included in the Struts framework: HTML tags, logic tags, bean tags, and template tags. HTML tags create links, images and so on. Logic tags manage the control flow. Bean tags are for interaction with the application logic, and template tags are used for dynamic content having the same format.

Custom tags: As in the JSP framework itself, a user can create his/her own tag libraries for Struts.

3.2.5 The Controller

The controller is responsible for mapping a request passed on in the URI to an action class. The controller is configured and driven by an application-wide configuration file (“struts-config.xml”), which contains all data relevant for the controller, including:

Action classes: A action class handles a request and returns an ActionForward object, which identifies the JSP page to generate the proper response. Therefore it is essential for the controller to know the full qualified class name of the action class.

Form beans: A form bean represents the data entered into a HTML form by an end user.

Action mapping: The association of a request to a certain action class is defined through action mapping.

Some other properties of the web application can be configured in the struts-config.xml file, such as global forwards, data source definitions and several others.

3.3 User management and access control

This section describes the user management system, that was developed at the Institute of Biomedical Engineering at Graz University of Technology by Dieter Zeller. This user management system is used by the TMA application, because it allows to implement the flexible user access control mechanism required in the TMA process. This is key to the TMA application because:

- Many people typically participate in the TMA process.
- The TMA application needs to separate them and provide different levels of access to different groups.
- Sensitive user and application data should be protected, and
- the user administration should be as easy as possible.

The user management system uses the same technologies as the TMA database application, and can therefore be easily integrated to secure the application against unauthorised access. A brief overview of the user management and access control system is provided here, a more detailed description can be found in [41].

The user management system constitutes an enterprise application that includes a web client for administration. It consists of several different parts and provides access administration to any computer systems using the Open Lightweight Directory Access Protocol (LDAP), a Samba server and several other technologies.

Fig. 3.6 illustrates the basic architecture of the user management system:

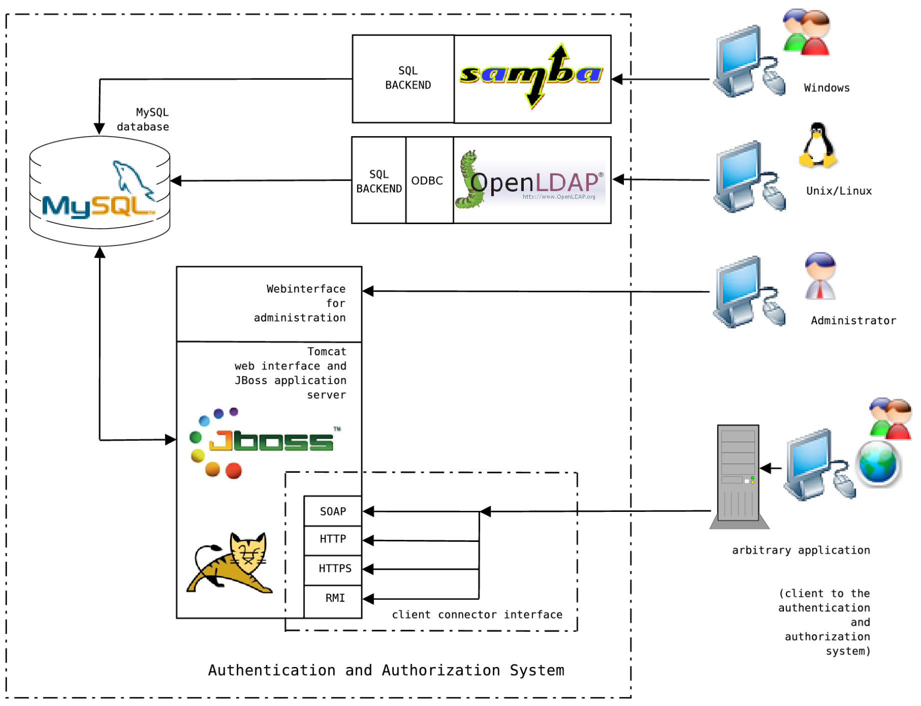

SQL BACKEND
samba
MySQL database
MySQL
SQL BACKEND
ODBC
OpenLDAP
Windows
Unix/Linux
Webinterface for administration
Tomcat web interface and JBoss application server
JBoss
Administrator
SOAP
HTTP
HTTPS
RMI
client connector interface
arbitrary application
(client to the authentication and authorization system)
Authentication and Authorization System

Figure 3.6: Conceptual overview of the user management system [41]

A user management client is provided to manage the access for other web based applications such as the tissue microarray database application described in this document. The connection can be established using any of four different protocols: Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS), Simple Object Access Protocol (SOAP), and Remote Method Invocation (RMI).

Once the access control levels are defined (by virtue of using the user management web interface), a Struts library is provided to perform commonly used tasks related to providing security for web applications. Some examples are shown below:

`<login:checkLogin/>`: This tag connects to the user management system and verifies, whether a user is logged in and has access to the web page. If verification fails, the user is redirected to a separate page (e.g., a login page).

`<permission:hasPermission resourceKey="del"accessLevel="RW">`: This tag implies that content within this tag is only displayed if the current user has read or write permission to the predefined resource (*del*). A delete button for instance can be placed here, to ensure that only authorised user can see the delete facility.

`<permission:noPermission resourceKey="del"accessLevel="RW">`: The body of this tag is processed only if the current user does *not* have permission to the given resource.

In sum, the user management system used in the TMA application provides the ability to manage and secure web applications in an easy way and with little development overhead.

Chapter 4

Review of commonly-used database technologies and systems

The database is an essential part of the TMA database application. A database management system (DBMS) is generally useful for a variety of requirements, including:

- *Reduction of data duplication:* to provide data integrity.
- *Efficient data entry, updates and deletions.*
- *Data independence:* Users can access data only through a standardised interface to the database, regardless how the data is physically stored.
- *Application independence:* The database is independent of the applications accessing the database.
- *Data sharing among multiple users*

The following sections summarise some basic principles and techniques that are essential for effective database development, and that are used extensively in the TMA database application:

4.1 The theory of relational databases

The relational database model was conceived by E. F. Codd in 1969, then a researcher at IBM ([14]). The model is based on branches of mathematics called set theory and predicate logic. The basic idea behind the relational model is that a database consists of a series of unordered tables (or relations) that can be manipulated using non-procedural operations that return tables. This model stood in stark contrast with more traditional database theories that were in use at that time and which were much more complicated, less flexible and highly dependent on the physical storage methods of the data.

Tables hold information about instances of entities, their attributes and their relations among them. Every entity instance consisting of a unique record (tuple) of data represents a row in the table . The uniqueness of each data record is based on one well-defined primary key (PK) whose occurrence must be unique within each table. To realise one-to-many (1:N) or many-to-one (N:1) interrelations, data records must contain primary keys of foreign tables, called foreign keys (FK). An implementation of many-to-many (N:N) relations requires an additional table storing explicit allocations of different foreign keys. The definition of relations allows the stored data to be broken down into smaller logical units, that are easier to maintain.

When designing a database, the designer has to make decisions regarding how best to take some system in the real world and model it in a database. This consists of deciding which tables to create, what columns they will contain, as well as the relationships between the tables. While it would be nice if this process was totally intuitive and obvious, or even better automated, this is simply not the case. A well-designed database takes time and effort to conceive, build and refine.

4.2 Normalisation

As mentioned earlier, designing databases requires a series of choices. How many tables are necessary and what will they represent? Which columns belong to which tables? What will the relationships between the tables be? The answers to each of these questions lies in normalisation. Normalisation is the process of simplifying the design of a database to achieves the optimal structure.

Normalisation theory provides the concept of normal forms to assist in achieving the optimal structure. The normal forms are a linear progression of rules that are applied to a database, with each higher normal form achieving a better, more efficient design.

First Normal Form (1NF): *The First Normal Form (1NF) requires that all column values must be atomic.* 1NF dictates that, for every row-by-column position in a given table, there exists only one value (i.e. not an array or list of values). The benefits from this rule should be fairly obvious. If lists of values are stored in a single column, there is no simple way to manipulate those values.

Second Normal Form (2NF): *A table is said to be in Second Normal Form (2NF), if it is in 1NF and every non-key column is fully dependent on the (entire) primary key.* Tables should only store data related to one thing (or entity) and that entity should be described by its primary key. To put a database into 2NF it is often necessary to split up tables into two ore more tables.

Third Normal Form (3NF): *A table is said to be in Third Normal Form (3NF), if it is in 2NF and if all non-key columns are mutually independent.* An obvious example of a dependency is a calculated column. For example, if a table contains the columns Quantity and PerItemCost, you could opt to calculate and store in that same table a TotalCost column (which would be equal to Quantity*PerItemCost), but this table would not be in 3NF. It s better exclude this column from the table and make the calculation in a query or on a form or a report instead. This saves space in the database and avoids having to update TotalCost every time Quantity or PerItemCost changes.

Higher Normal Forms After Codd defined the original set of normal forms it was discovered that Third Normal Form, as originally defined, had certain inadequacies. This led to several higher normal forms, including the Boyce-Codd Normal Form (BCNF), which have more restrictive constraints on data dependencies than 3NF. Fourth Normal Form separates independent multi-valued facts stored in one table into separate tables. Fifth Normal Form breaks out data redundancy that is not covered by any of the previous normal forms. For more information on higher normal forms it is referred to [15] and [30].

4.3 Integrity rules

The relational model specifies two general integrity rules. They are referred to as general rules, because they apply to all databases. They are: entity integrity and referential integrity. [25]

The entity integrity rule is very simple. It says that primary keys cannot contain null (missing) data. The reason for this rule should be obvious. A row in a table can not be uniquely identified or referenced, if the primary key of that table can be null.

The referential integrity rule says that the database must not contain any unmatched foreign key values. This implies that:

- A row may not be added to a table with a foreign key unless the referenced value exists in the referenced table.

- If the value in a table that s referenced by a foreign key is changed (or the entire row is deleted), the rows in the table with the foreign key must not be orphaned.

All integrity constraints that do not fall under entity integrity or referential integrity are termed database-specific rules or application rules. These type of rules are specific to each database and come from the rules of the application being modelled by the database. It is important to note that the enforcement of application rules is *as* important as the enforcement of the general integrity rules discussed in this section.

4.4 The structured query language (SQL) to access relational databases

The implementation of Codd's ideas, sponsored by IBM, led to the emergence of the Structured Query Language (SQL) [16]. SQL has evolved into a standard open language, without cooperative ownership. Almost all currently available database implementations are designed to meet the SQL standards. [11]

SQL belongs to the category of non-procedural languages called *declarative languages*. As opposed to procedural languages which can result in many lines of code, an SQL command consists always of just one statement. A single database query consists of an SQL statement which includes all desired requests. This statement is sent to the database management system (DBMS) which then executes internal procedures and finally returns the desired dataset. [16]

The language consists of 3 major components:

The data definition language (DDL) defines the way in which data is stored. The DDL is used to create, change the structure or remove whole tables and other relational structures. For instance, the DDL CREATE TABLE statement creates a new table in the database. Other important DDL commands whose names are fairly self-descriptive include CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, DROP INDEX.

The data manipulation language (DML) enables the insertion, manipulation and retrieval of data. The DML of SQL allows the insertion, update and removal of rows stored in relational database tables. The most important commands of DML are SELECT, UPDATE, DELETE, and INSERT INTO.

The data control language (DCL) is used to define actions that are not covered by the other two components.

The DCL include actions, such as granting privileges to users, defining when proposed changes to a databases should be irrevocably made. Important statements of the DCL include GRANT, CHECK, PRIMARY KEY, COMMIT, ROLLBACK, etc.

4.5 The "Java Database Connectivity" (JDBC) application programming interface

JDBC[1] is a Java application programming interface (API) for executing SQL statements. It consists of a set of classes and interfaces written in the Java programming language, and makes it possible to write database applications using Java. [21]

As the TMA application was developed using Java-based technologies, the choice of using JDBC for accessing the TMA database was natural.

The three main features of JDBC that are also relevant to the TMA application are:

- establishing a connection with a database
- sending SQL statements
- processing the results

In the three-tier model, commands are sent to a "middle tier", which converts these commands to SQL statements and sends them to the database. The database processes the SQL statements and then sends the results back to the middle tier, which in turn passes them on to the user. Developers find the three-tier model very attractive because the middle tier makes it possible to control access to the data in a flexible way. Another advantage is that with the presence of a middle tier, the user can employ an easy-to-use higher-level API which is translated by the middle tier into the appropriate low-level calls. Finally, in many cases a three-tier architecture can provide performance advantages, because

[1]As a point of interest, JDBC is a trademarked name and is not an acronym; nevertheless, JDBC is often thought of as standing for "Java Database Connectivity".

the middle tier is typically written in C++ or with an application server using Enterprise JavaBeans, which both offer good performance.

In summary, the JDBC API is a pure Java interface to the basic SQL abstractions and concepts. It builds on the concepts of the Open Database Connectivity (ODBC) model rather than starting from scratch. Programmers familiar with ODBC will find it very easy to learn JDBC, which retains the basic design features of ODBC. The big difference is that JDBC builds on and reinforces the style and virtues of Java, and it is easy to use. [21]

In sum, the key concepts associated with relational databases (normalisation, integrity rules, and the structured query language), are a de-facto industry standard for accessing databases. Although the SQL implementation may vary between different database vendors, it is important to be familiar with these key concepts of SQL.

JDBC connectors (drivers) exist for nearly all known database management systems. In the TMA application, extensive use is made of both SQL and JDBC.

4.6 The MySQL database system and InnoDB

MySQL is a popular SQL database developed by the open source community, that is powerful, flexible and efficient ([28]). Since MySQL does not support important DBMS features itself, there are several extensions, which are being increasingly integrated into newer versions of the MySQL DBMS. The most important extension is InnoDB ([23]), which provides transactions, foreign-key constraints, performance monitoring and several other features. InnoDB also allows to store Binary Large Objects (BLOBS) in a database.

The TMA database application use both MySQL and InnoDB, primarily because:

- The TMA-DB needs to store very large images directly into the database.
- MySQL is a lightweight SQL implementation.
- Foreign-key constraints are supported by InnoDB. This is essential for the TMA-DB, and required to implement dependencies among objects.
- It is a free and open-source database system.

Chapter 5

Design and Implementation of the TMA database and the web interface

This chapter describes the design and development of the tissue microarray database application, which manages data collected during the TMA production process. It also shows how external algorithms can be integrated. The first part of this chapter describes the database model and design. This is followed by a description of the TMA web interface.

5.1 The database model for the TMA application

A relational database management system is used to store all TMA-related data, such as the tissue images and experimental results. Initially an Oracle database ([29]) was considered for this task, as there is extensive know-how available for this system within the bioinformatics group at the Graz University of Technology. Oracle is the leading database technology available today, but it is expensive and generally requires a fairly high-end computing infrastructure in order to run efficiently. However, the TMA application was designed to be simple and inexpensive, and is to be installed on a single machine inside the hospital, where the TMA application is used (LKH[1] Graz - Institute of Pathology). Therefore, MySQL a more light-weight database system was chosen to implement the TMA database. MySQL is an open source database that is free, simple to use, yet powerful and flexible enough to run the TMA applications.

The images originating from the TMA slides are up to 50MB in size and are to be stored directly

[1]LKH = Landeskrankenhaus (district hospital)

into the database. Storing the images digitally allows for the automated analysis of staining intensities and features on TMA slides, using sophisticated image analysis techniques.

The main challenge in designing the database design is to define the most appropriate data definition and the relationships between the various data elements (tables). At the minimum, the database should hold data from the original tissue specimen, the donor and the recipient arrayblock, the sections, as well as the results of different analyses on the punch images.

Mapping the real world as closely as possible, the database is structured into six main components which correspond to the main steps in the tissue microarray production process, namely (the colors in parenthesis colours refer to the colours in fig. 5.1)[2]:

- Tissue tables (red and light red)
- Donor tables (blue and light blue)
- Arrayblock tables (yellow and light yellow)
- Section tables (orange and light orange)
- Staining tables (purple and light purple)
- Punch tables (green and light green)

There are also four additional database tables that are not directly related to the main components, storing independent data (Projects- and Persons-table) or link parts together (DonorArrayAssociation- and ArrayPunchAssociation-table). These are not coloured in fig. 5.1.

The resulting database structure is shown in figure 5.1, where each box represents a database table and each arrow denotes a foreign-key constraint. Each foreign-key constraint runs from a foreign-key (FK) in a table to a primary-key (PK) in an other table.

The following sections describe these components in more detail:

5.1.1 Tissue tables

The *Tissues* table (red table) describes a general tissue specimen, specified by its species (in general ”homo sapiens”) and the diagnosis, using the standardised International Classification of Disease for Oncology (ICD-O) code for denoting an oncological diagnosis ([7]). An ICD-O code consists of

[2]As from now colours in parenthesis always refers to the coloured tables in fig. 5.1

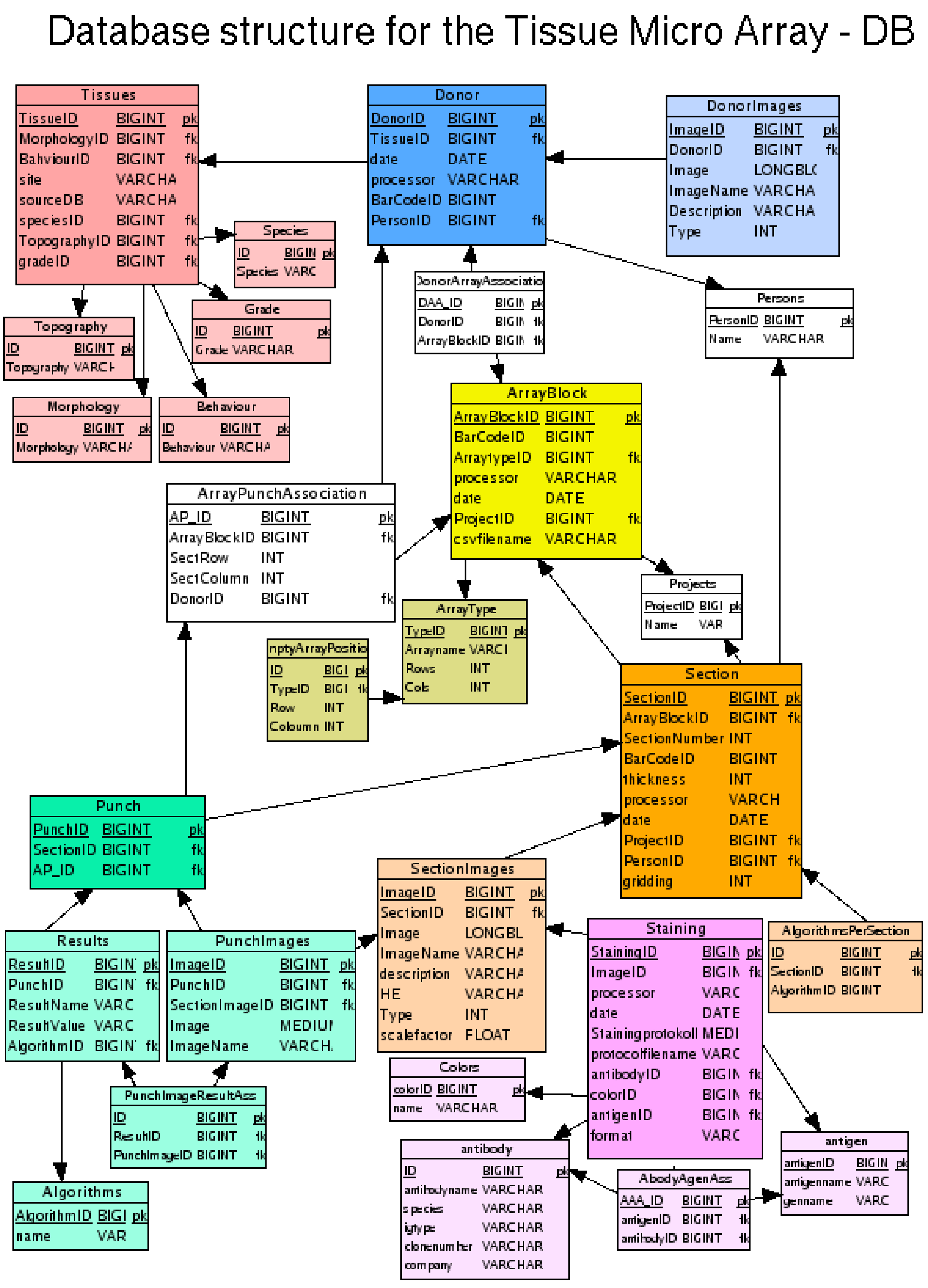


Figure 5.1: *The TMA database structure is divided into six main parts which correspond to the main steps in the TMA production process: Tissues tables (red), Donor tables (blue), Array-block tables (yellow), Section tables (orange), Staining tables (purple), and Punch tables (green).*

four different values: Topography, morphology, behaviour and grade. To meet the rules of normalisation regarding to database design (see section 4), only the IDs of these values are stored in the Tissues table, which are linked to the surrounding tables, holding the values itself (see light red tables *Topography*, *Morphology*, *Behaviour*, *Grade*, and *Species*).

Each tissue is linked to a donor block, which is manifested by a *foreign-key constraint* reference in the database. This simply means that a certain tissue *has* to exist, before it is referred to by an entry in the donor table.

To increase flexibility and maintain compatibility with other data sources holding general tissue data, the *tissues* table is separated from the *donor* table, which contains more TMA specific data.

5.1.2 Donor tables

Each donorblock (short: Donor) refers to exactly one tissue. As a donor is a more TMA specific entity than a general tissue specimen, some additional TMA related data is stored in the *Donor* Table (blue table), such as the barcode-ID, the date, the persons name creating the donor, and an ID from the supervisor responsible for the donor (e.g., a pathologist). One or more images can be generated from such a donor, which is enabled through a link to a *DonorImages* table (light blue). The barcode-ID can be used to link the data with patient related data, which is stored in an external histology database. For legal reasons, there is no patient information stored in the TMA database and the information is fully anonymised. This ensures that only authorised users can access information about patient related data by virtue of an external lookup table, which is linking the barcode-ID to entries in histology databases.

5.1.3 Arrayblock tables

During the TMA production process, array blocks are created. The corresponding *ArrayBlock* table (yellow) holds additional data associated to an arrayblock, such as the date, the person who created the arrayblock, and so on. An arrayblock is created by a machine, the TMA arrayer (see fig. 1.2(a)). Besides the arrayblock itself, the TMA arrayer creates a comma separated values datafile (CSV-file), which is also inserted into this table. This file is parsed during upload and holds the following two important pieces of information for the TMA database application. First, information regarding the donorblocks related to the punches on the arrayblocks is extracted and stored in an n-n-relation lookup table (*DonorArrayAssociation*). Second, the coordinates of each punch on the array are extracted and stored in table *ArrayPunchAssociation*. The CSV-file is only stored for the sake of completeness to maintain flexibility; it holds additional data which is currently not used, but may be used by future algorithms or extensions.

Several array layouts are defined for TMAs, which all differ in the number of rows and columns of an array. Therefore, the TMA database provides an additional *ArrayType* table (light yellow) to store this information. In addition, not all positions of the arrayblock are occupied by a punch. As this information is important for further image analysis, the positions *not* containing a punch are also stored, in the trailing *EmptyArrayPositions* table (light yellow).

5.1.4 Section tables

The *Section* table (orange) holds information related to each slide (section) of an arrayblock, such as meta data, such as the creation date, the barcode ID, the thickness of the slide and whether a section is already gridded or not. Because it is essential to know which analysis algorithms have been applied to each section, an additional *AlgorithmsPerSection* table (light orange) is attached to the section table to be able to hold this information. One or more digital images can be acquired by a laser scanner, and are stored in a separate table, the *SectionImages* table (light orange). Since the section images can be very large (up to 50MB TIFF images), an additional image is automatically generated for each image, which is also stored in this table. The generated images are scaled down in it’s graphical resolution and are converted to the JPEG format. These images can be displayed much faster by the clients (e.g., a web browser) because they do not have to convert the uncompressed TIFF images to a format they can display (generally, web browsers cannot display TIFF images, but support JPEG images) each time an image is displayed.

The *Section* table has a one-to-many relation to the staining information corresponding to each slide.

5.1.5 Staining tables

Staining information describes how a slide is prepared before it is scanned into a digital image. For pathologists, this information is important for the interpretation of results that are generated by the integrated analysis algorithms. Information about the staining procedure is stored in the *Staining* table (purple), and contains the name of the executing person, the date, and an optional staining protocol (the staining process is often predefined and described in technical descriptions of the process called “Standard Operating Procedures” - SOPs) which can be an arbitrary file. Furthermore the antibody, the antigen and the fluorescence dyes used for staining are important values here and are also stored into the *Staining* table and it's associated tables *Colors*, *antibody* and *antigen* (light purple). Antibodies bind to certain antigens, this fact is represented by an additional lookup table (*AbodyAgenAss*) between the *antibody* and the *antigen* table.

5.1.6 Punch tables

A key feature of the TMA application is the ability to analyse each punch of a section separately. Therefore each section image is further split up by the gridding algorithm into smaller individual punch images. The TMA application uses the *Punch* table (green) to store information about a punch itself and the *PunchImages* table (light green), for storing one or more sub-images from a punch. To provide the possibility for analysis algorithms (table *Algorithms* stores the titles of available algorithms) to store their generated results for each punch, the *Results* table and the *PunchImageResultAss* table (light green) are attached to this group of tables. The latter table is important, because sometimes an algorithm requires more than one punch image to produce a result (e.g., an algorithm that computes an overlap of common areas on the punches; such an algorithm was developed for and integrated into the TMA application by Elke Pauritsch at Graz University of Technology, see [31]). The information in this table describes the correspondence of punch images to a certain result.

5.1.7 User management

As mentioned earlier, the flexible user management and access control system, developed by Dieter Zeller [41] at Graz University of Technology is integrated into the tissue microarray database application. Therefore, an additional *usermanagement* database is needed. This comprehensive system allows for effective authentification and authorisation of the TMA web application. The *usermanagement* database stores user data, access rights, security resources as well as passwords in a safe way and is completely independent from the TMA database structure. It does not restrict the design of this TMA database application in any way.

5.2 The TMA web application

As described in section 3, the web application for the TMA database application is based upon the Struts Framework, and uses JavaServer pages (JSPs) to generate the users view inside a web browser.

The web server Tomcat v4.1, an open source project from the Jakarta[3]-Group as well as the Struts framework itself, is used.

With appropriate access rights (username & password) a user is allowed to log into the TMA web page (Fig.5.2). The web page provides the ability to insert, edit and view TMA related data.

The web page is fully internationalised and supports the languages English and German. Depending on the browsers language settings the web page appears in the proper language. The default language is English.

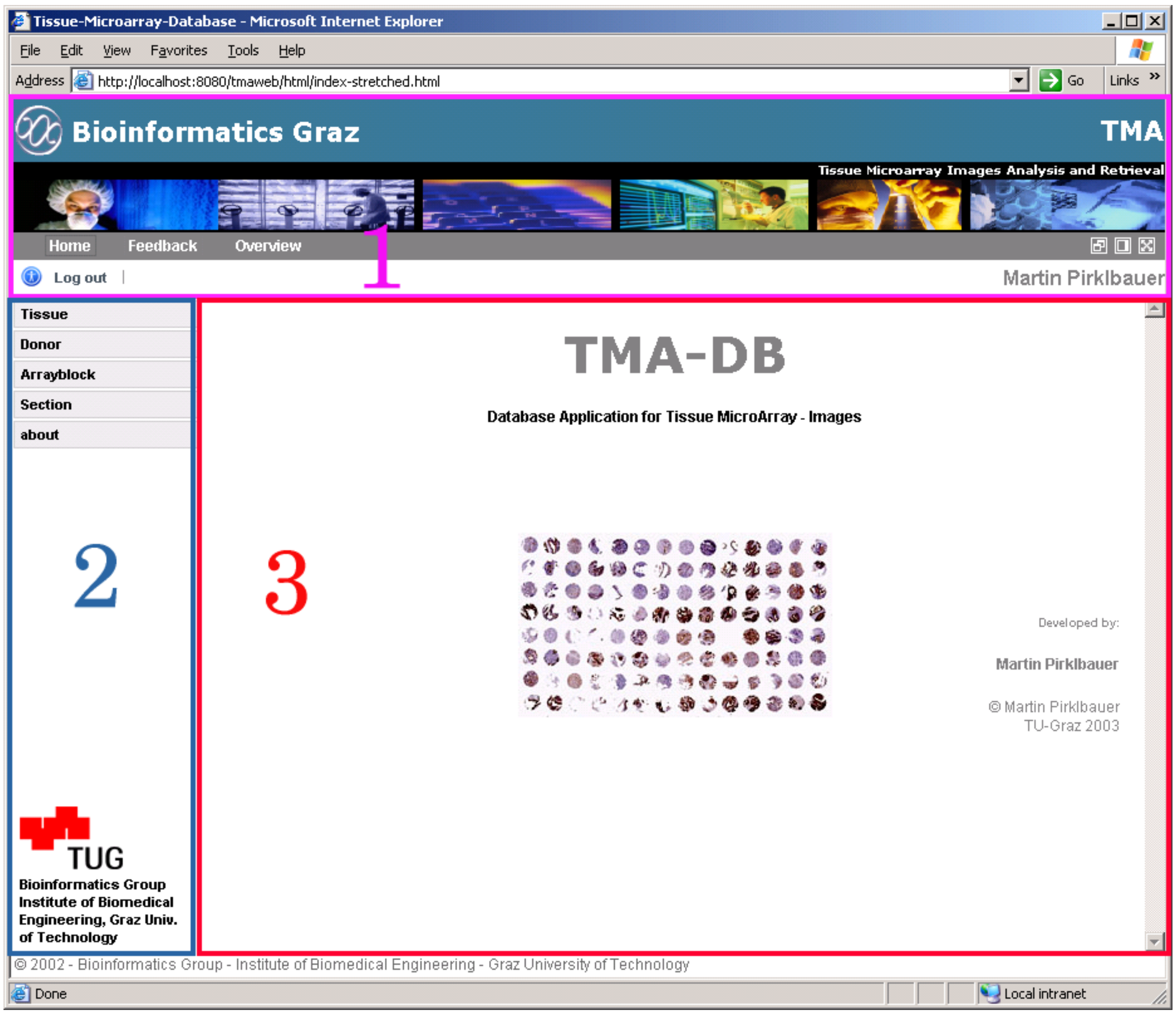

Tissue-Microarray-Database - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:8080/tmaweb/html/index-stretched.html
Go Links
Bioinformatics Graz
TMA
Tissue Microarray Images Analysis and Retrieval
Home Feedback Overview
1
Log out
Martin Pirklbauer
Tissue
Donor
Arrayblock
Section
about
2
3
TMA-DB
Database Application for Tissue MicroArray - Images
Developed by:
Martin Pirklbauer
© Martin Pirklbauer
TU-Graz 2003
TUG
Bioinformatics Group
Institute of Biomedical
Engineering, Graz Univ.
of Technology
© 2002 - Bioinformatics Group - Institute of Biomedical Engineering - Graz University of Technology
Done
Local intranet

Figure 5.2: The TMA web page: Top area (part 1) containing the main menu, a navigation menu on the left (part 2) and main area for displaying data (part 3).

Each TMA web page is divided into three main areas:

First, the top area (1 - see fig. 5.2) shows the title, some logos and basic features for the web

[3]`http://jakarta.apache.org`

application like a “Home”-, “Feedback”- and “Overview”-Link. On the right side three features for customising the appearance of the page are available: “full-screen”, “stretch to window width”, and “restore to default size”. The line below provides menu entries like login/logout and the full name of the person logged in is displayed.

Second, a navigation menu on the left side of the screen (2) is displayed for main navigation inside the TMA web application.

The third part, down right is the main area (3), displays all data, information and input screens.

5.2.1 Tissue actions

When selecting the menu item “Tissue” from the menu on the left side, two actions are available. One is for inserting a new tissue into the database, the other for viewing already inserted tissues. All parameters, which form the ICD-O code of the tissue can be selected via a drop-down input field. The values of these fields (topography, morphology, classification, and grade) are extracted from the database and appear in mnemonic form, so there is no need for the user to know the ICD-O cancer coding by heart.

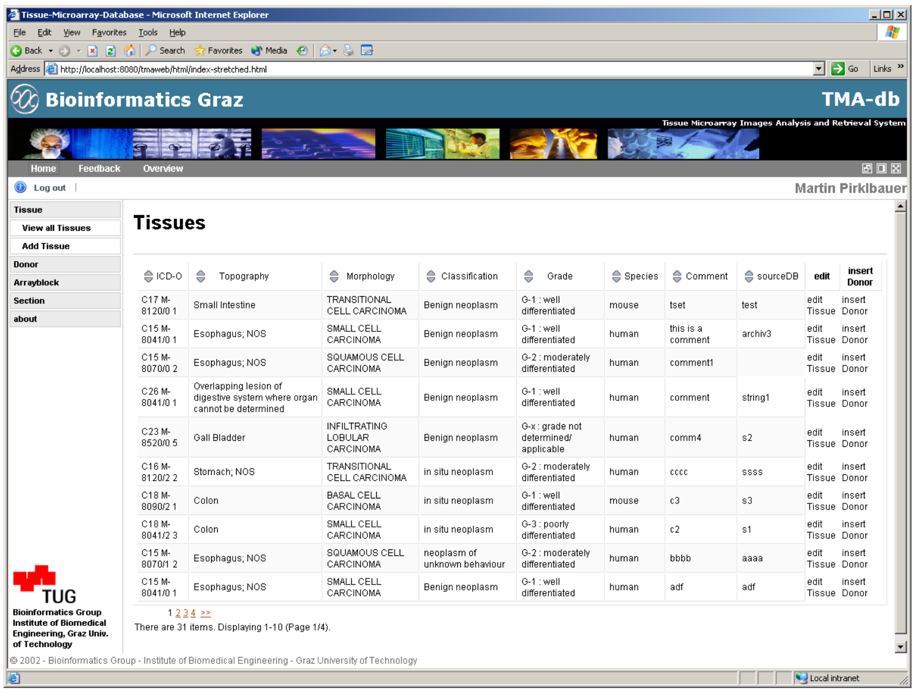
Fig. 5.3 shows the screen, when the user views already inserted tissues. The ICD-O code in the left column is automatically composed by the application logic and the mnemonic form of the code is displayed. It is possilbe to sort the table for each column by clicking on the arrows in the head of a individual column.

For each tissue (one tissue is represented by one row in the table) two actions are available: “Edit tissue”, allows the user to change the field, if a mistake happened. With “Insert donor”, the user can jump directly to the “Insert donor” input screen, where the current tissue is preselected and displayed.

Tissues can grouped and displayed as groups. A status line on the bottom informs about the number of tissues in the database, and the user has a convenient ability to jump between the individual pages.

5.2.2 Donor actions

The same actions available for tissues can be applied to donors. The input screen for donors is shown in Fig. 5.4. Some fields are pre-selected, such as the date, which can be changed with a click on the calendar icon, where a little calendar pops up.


Tissue-Microarray-Database - Microsoft Internet Explorer
http://localhost:8080/tmaweb/html/index-stretched.html
Bioinformatics Graz
TMA-db
Tissue Microarray Images Analysis and Retrieval System
Home
Feedback
Overview
Log out
Martin Pirklbauer
Tissue
View all Tissues
Add Tissue
Donor
Arrayblock
Section
about
TUG
Bioinformatics Group Institute of Biomedical Engineering, Graz Univ. of Technology
Local intranet

Tissues

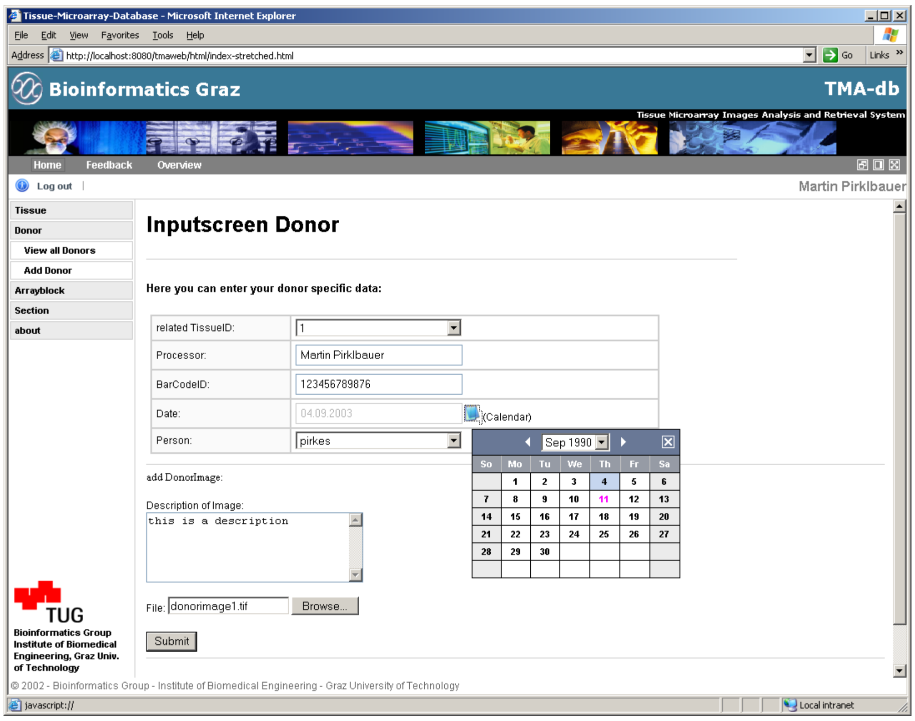
ICD-O	Topography	Morphology	Classification	Grade	Species	Comment	sourceDB	edit	insert Donor
C17 M-8120/0 1	Small Intestine	TRANSITIONAL CELL CARCINOMA	Benign neoplasm	G-1 : well differentiated	mouse	tset	test	edit Tissue	insert Donor
C15 M-8041/0 1	Esophagus; NOS	SMALL CELL CARCINOMA	Benign neoplasm	G-1 : well differentiated	human	this is a comment	archiv3	edit Tissue	insert Donor
C15 M-8070/0 2	Esophagus; NOS	SQUAMOUS CELL CARCINOMA	Benign neoplasm	G-2 : moderately differentiated	human	comment1		edit Tissue	insert Donor
C26 M-8041/0 1	Overlapping lesion of digestive system where organ cannot be determined	SMALL CELL CARCINOMA	Benign neoplasm	G-1 : well differentiated	human	comment	string1	edit Tissue	insert Donor
C23 M-8520/0 5	Gall Bladder	INFILTRATING LOBULAR CARCINOMA	Benign neoplasm	G-x : grade not determined/ applicable	human	comm4	s2	edit Tissue	insert Donor
C16 M-8120/2 2	Stomach; NOS	TRANSITIONAL CELL CARCINOMA	in situ neoplasm	G-2 : moderately differentiated	human	cccc	ssss	edit Tissue	insert Donor
C18 M-8090/2 1	Colon	BASAL CELL CARCINOMA	in situ neoplasm	G-1 : well differentiated	mouse	c3	s3	edit Tissue	insert Donor
C18 M-8041/2 3	Colon	SMALL CELL CARCINOMA	in situ neoplasm	G-3 : poorly differentiated	human	c2	s1	edit Tissue	insert Donor
C15 M-8070/1 2	Esophagus; NOS	SQUAMOUS CELL CARCINOMA	neoplasm of unknown behaviour	G-2 : moderately differentiated	human	bbbb	aaaa	edit Tissue	insert Donor
C15 M-8041/0 1	Esophagus; NOS	SMALL CELL CARCINOMA	Benign neoplasm	G-1 : well differentiated	human	adf	adf	edit Tissue	insert Donor

1 2 3 4 >>

There are 31 items. Displaying 1-10 (Page 1/4).

© 2002 - Bioinformatics Group - Institute of Biomedical Engineering - Graz University of Technology

Figure 5.3: Screenshot: “View Tissues”


Tissue-Microarray-Database - Microsoft Internet Explorer
http://localhost:8080/tmaweb/html/index-stretched.html
Bioinformatics Graz
TMA-db
Tissue Microarray Images Analysis and Retrieval System
Home
Feedback
Overview
Log out
Martin Pirklbauer
Tissue
Donor
View all Donors
Add Donor
Arrayblock
Section
about
TUG
Bioinformatics Group Institute of Biomedical Engineering, Graz Univ. of Technology
Browse...
Submit
javascript://
Local intranet

Inputscreen Donor

Here you can enter your donor specific data:

related TissueID:	1
Processor:	Martin Pirklbauer
BarCodeID:	123456789876
Date:	04.09.2003 (Calendar)
Person:	pirkes

Sep 1990

So	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

add DonorImage:

Description of Image:
this is a description

File: donorimage1.tif

© 2002 - Bioinformatics Group - Institute of Biomedical Engineering - Graz University of Technology

Figure 5.4: Screenshot: “Donor inputscreen”

The Struts framework offers a great facility for form validation. For instance, rules can be defined for the structure of legal values in certain input fields (e.g., “it is only possible to upload *.tiff or *.jpg images or the barcode must contain 13 digits”). A proper error message is displayed, if such a rule is not satisfied, and the user is invited to correct the error.

5.2.3 Arrayblock actions

For inserting arrayblock-related data it is important to upload the CSV-file, which is created by the TMA arrayer (see Fig. 1.2(a)). This file defines which specimens of certain donorblocks are on the actual arrayblock, including it’s coordinates on the array. This file is mandatory.

The other functions (view, edit) are similar to the actions of tissue and donor.

5.2.4 Section actions

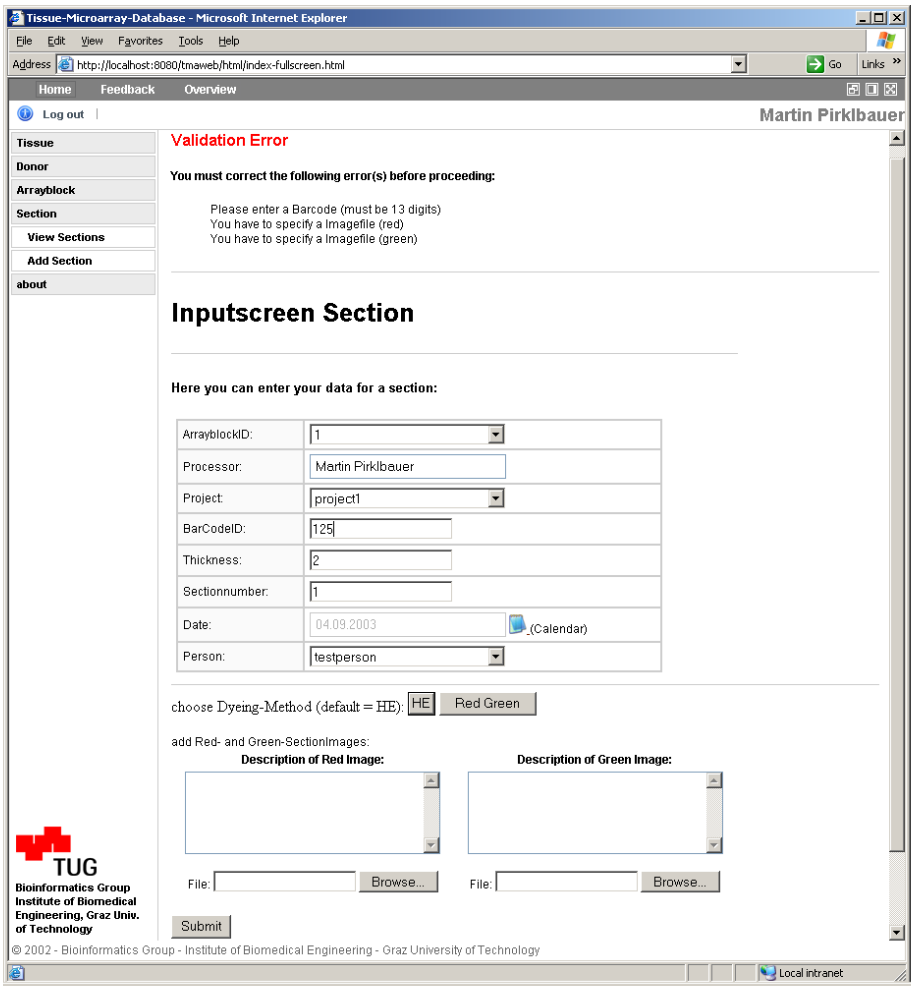
The input screen for inserting section data of a specific arrayblock is illustrated in Fig. 5.5. Depending on the used staining method, HE staining on one side and Cy3-Cy5-staining (red-green) on the other side, there are either one or two section images available. With two buttons in the middle of the screen, the user can change the input screen to get either one or two areas for the upload of the section images.

Fig. 5.5 illustrates how the form validation works in this system. In this example the user entered a barcode ID, which does not match the constraints defined within the Struts ActionForm class for this input screen. As the images are also required, the user is prompted to correct this too. Form validation is implemented for all input fields used in the TMA database application.

Editing a section

When a list of sections is displayed, there is an additional link (“more”), allowing to edit and view section parameters and images. A very important feature is the integrated gridding applet, which can be started from that screen.

The user can see all section-related meta data (such as the barcode ID, the date) as well as the uploaded images for this section. The user has the possibility to view the automatically generated JPEG images and is able to download each image separately, to the local hard disk.


Tissue-Microarray-Database - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:8080/tmaweb/html/index-fullscreen.html
Go Links
Home Feedback Overview
Log out
Martin Pirklbauer
Tissue
Donor
Arrayblock
Section
View Sections
Add Section
about
TUG
Bioinformatics Group Institute of Biomedical Engineering, Graz Univ. of Technology

Validation Error

You must correct the following error(s) before proceeding:

Please enter a Barcode (must be 13 digits)
You have to specify a Imagefile (red)
You have to specify a Imagefile (green)

Inputscreen Section

Here you can enter your data for a section:

ArrayblockID:	1
Processor:	Martin Pirklbauer
Project:	project1
BarCodeID:	125
Thickness:	2
Sectionnumber:	1
Date:	04.09.2003 (Calendar)
Person:	testperson

choose Dyeing-Method (default = HE): HE Red Green

add Red- and Green-SectionImages:

Description of Red Image: Description of Green Image:

File: Browse... File: Browse...

Submit

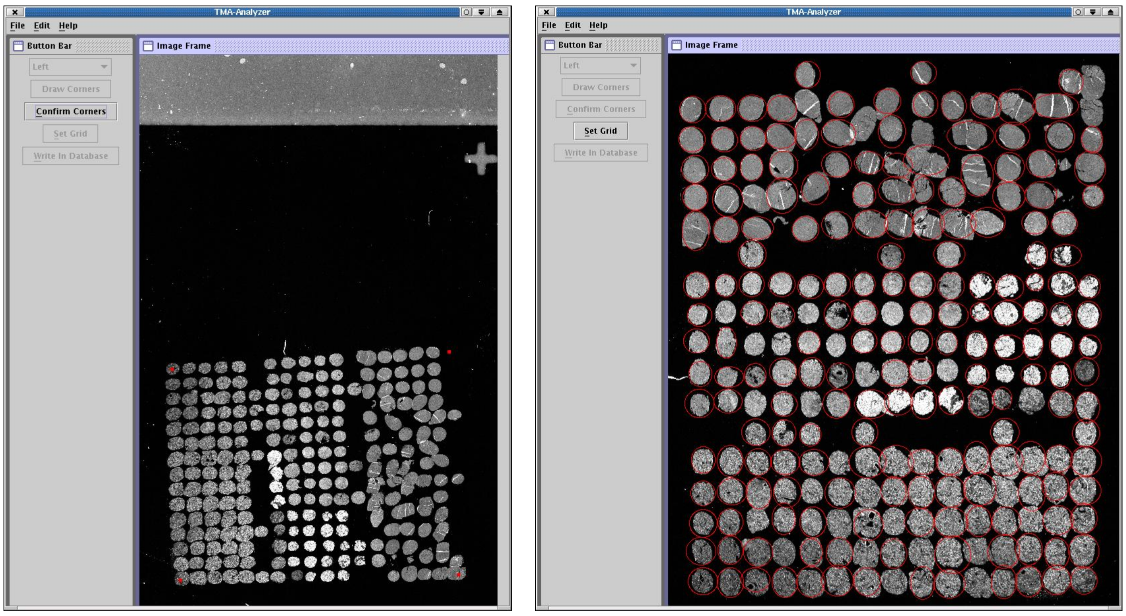
© 2002 - Bioinformatics Group - Institute of Biomedical Engineering - Graz University of Technology

Figure 5.5: Screenshot: “Insert Section” with form validation messages

Gridding algorithm

The gridding algorithm is very important for further analysis. By clicking on the “start gridding applet” link an Java applet is started. The applet fetches the uncompressed section images from the database and displays it on the screen (after an adequate resampling and resizing process). The user sets the corner spots manually in the image (see Fig. 5.6(a)). The algorithm then finds all punches in the array, and the section image will be split into small punch images, which are stored back to the database, including their correct meta data (the section number, the coordinates on the array, etc.).

The gridding algorithm only locates the punches which have corresponding entries in the CSV-file, produced by the TMA arrayer (see Fig. 5.6(b)). Punches without an entry in the CSV-file can not be associated to a specific coordinate on the array, and are therefore discarded by the system to avoid inconsistent data sets.



(a) (b)

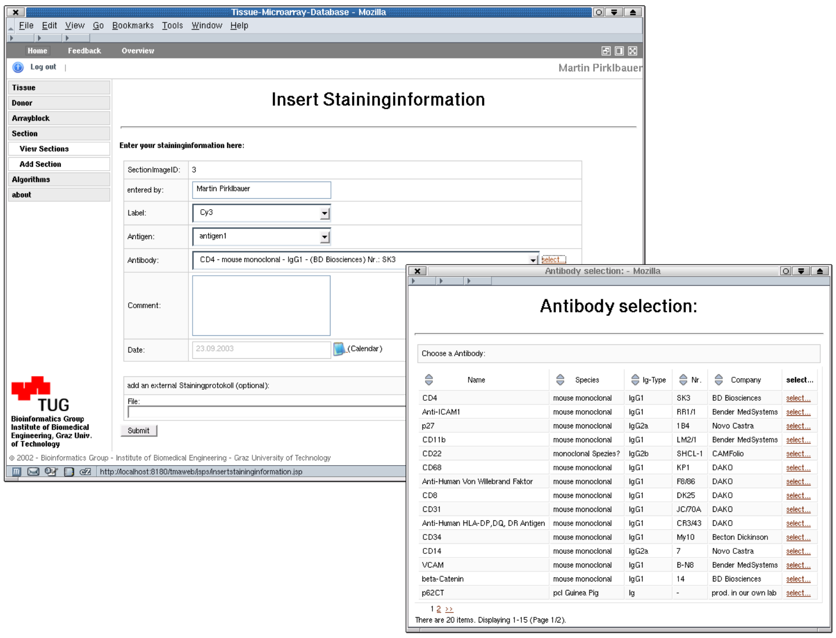
Figure 5.6: Gridding applet: Fig. 5.6(a) shows, how a user can set the corner points of a loaded section image; Fig. 5.6(b) illustrates the punches found by the gridding algorithm.

The gridding algorithm was implemented by Martina Uray at Graz University of Technology as a part of her master thesis [40].

Staining

For each uploaded section image it is possible to store additional staining information. Adequate input, view and edit screens are provided for the staining information (see 5.1.5).

As the list of defined antibodies can become very long, there is a need for a comfortable way to select the antibody at the “insert staining information” screen. An ordinary drop-down input field is displayed, where all information of an antibody (name, species, Ig-Type, clone number and antibody) is shown as a long text. For convenience if finding the right antibody, a further link (“choose”) is placed next to the drop-down field. Clicking on it pops up a new window, where all known antibodies are shown in tabular form. Each row represent an antibody and the columns are the different characteristics of an antibody mentioned above. The user can sort for each column independently, and is therefore able to find the desired antibody quickly. A click on the found antibody closes the pop-up window and changes the value of the originally displayed drop-down field (see Fig. 5.7).


Tissue-Microarray-Database - Mozilla
File Edit View Go Bookmarks Tools Window Help
Home Feedback Overview
Log out
Martin Pirklbauer
Tissue
Donor
Arrayblock
Section
View Sections
Add Section
Algorithms
about
TUG
Bioinformatics Group Institute of Biomedical Engineering, Graz Univ. of Technology
Submit
(Calendar)
select...
Antibody selection: - Mozilla

Insert Staininginformation

Enter your staininginformation here:

SectionImageID:	3
entered by:	Martin Pirklbauer
Label:	Cy3
Antigen:	antigen1
Antibody:	CD4 - mouse monoclonal - IgG1 - (BD Biosciences) Nr.: SK3
Comment:	
Date:	23.09.2003

add an external Stainingprotokoll (optional):

File:

© 2002 - Bioinformatics Group - Institute of Biomedical Engineering - Graz University of Technology

http://localhost:8180/tmaweb/jsps/insertstaininginformation.jsp

Antibody selection:

Choose a Antibody:

Name	Species	Ig-Type	Nr.	Company	select...
CD4	mouse monoclonal	IgG1	SK3	BD Biosciences	select...
Anti-ICAM1	mouse monoclonal	IgG1	RR1/1	Bender MedSystems	select...
p27	mouse monoclonal	IgG2a	1B4	Novo Castra	select...
CD11b	mouse monoclonal	IgG1	LM2/1	Bender MedSystems	select...
CD22	monoclonal Spezies?	IgG2b	SHCL-1	CAMFolio	select...
CD68	mouse monoclonal	IgG1	KP1	DAKO	select...
Anti-Human Von Willebrand Faktor	mouse monoclonal	IgG1	F8/86	DAKO	select...
CD8	mouse monoclonal	IgG1	DK25	DAKO	select...
CD31	mouse monoclonal	IgG1	JC/70A	DAKO	select...
Anti-Human HLA-DP,DQ, DR Antigen	mouse monoclonal	IgG1	CR3/43	DAKO	select...
CD34	mouse monoclonal	IgG1	My10	Becton Dickinson	select...
CD14	mouse monoclonal	IgG2a	7	Novo Castra	select...
VCAM	mouse monoclonal	IgG1	B-N8	Bender MedSystems	select...
beta-Catenin	mouse monoclonal	IgG1	14	BD Biosciences	select...
p62CT	pcl Guinea Pig	Ig	-	prod. in our own lab	select...

1 2 >>

There are 20 items. Displaying 1-15 (Page 1/2).

Figure 5.7: Screenshot: "Insert Staininginformation with the pop-up window for easy antibody selection

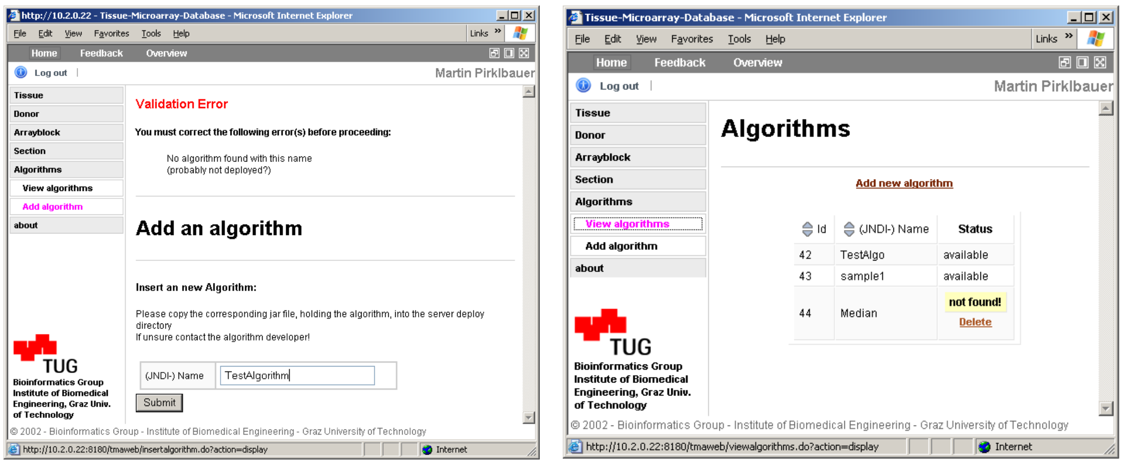
5.2.5 Inserting and maintaining analysis algorithms

When a new image analysis algorithm is implemented, the developer can configure the algorithm with the web interface. The algorithm developer has to copy a Java Archive file (jar) containing the algorithm into the deploy directory of the application server (see appendix B.6). To activate the algorithm, it is necessary to enter the Java Naming and Directory Interface (JNDI-) Name in a TMA web page. The entered name is verified, and the user is prompted for the name again, if the system can not find the algorithm (see Fig. 5.8(a)).

It is possible to display all currently registered algorithms. The system automatically checks the existence of each algorithm, displays the status, and offers the ability to delete the algorithm from the database, if an algorithm is not found (Fig. 5.8(b)).

5.2.6 Query options

For the data stored in the database, certain query procedures are available for accessing information in an appropriate way. Two types of queries are available: "basic queries" and "custom" queries. The basic query option allow to perform predefined, often required queries. More powerful queries can be set up by using the custom query page (Fig. 5.9(b)).


http://10.2.0.22 - Tissue-Microarray-Database - Microsoft Internet Explorer
File Edit View Favorites Tools Help Links
Home Feedback Overview
Log out
Martin Pirklbauer
Tissue
Donor
Arrayblock
Section
Algorithms
View algorithms
Add algorithm
about
TUG
Bioinformatics Group Institute of Biomedical Engineering, Graz Univ. of Technology
Submit
Internet
Tissue-Microarray-Database - Microsoft Internet Explorer
Add new algorithm
Delete

Validation Error

You must correct the following error(s) before proceeding:

No algorithm found with this name
(probably not deployed?)

Add an algorithm

Insert an new Algorithm:

Please copy the corresponding jar file, holding the algorithm, into the server deploy directory
If unsure contact the algorithm developer!

(JNDI-) Name: TestAlgorithm

© 2002 - Bioinformatics Group - Institute of Biomedical Engineering - Graz University of Technology

http://10.2.0.22:8180/tmaweb/insertalgorithm.do?action=display

Algorithms

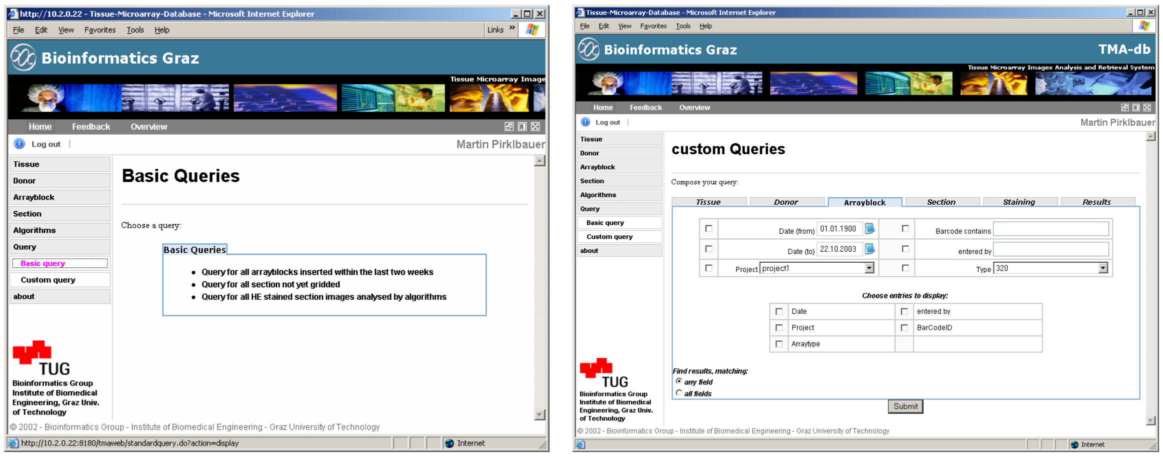
Id	(JNDI-) Name	Status
42	TestAlgo	available
43	sample1	available
44	Median	not found! Delete

© 2002 - Bioinformatics Group - Institute of Biomedical Engineering - Graz University of Technology

http://10.2.0.22:8180/tmaweb/viewalgorithms.do?action=display

(a) (b)

Figure 5.8: Fig. 5.8(a) shows the registration screen for a new algorithm. Fig. 5.8(b): A status page for algorithms.


http://10.2.0.22 - Tissue-Microarray-Database - Microsoft Internet Explorer
File Edit View Favorites Tools Help Links
Bioinformatics Graz
Tissue Microarray Image
Home Feedback Overview
Log out
Martin Pirklbauer
Tissue
Donor
Arrayblock
Section
Algorithms
Query
Basic query
Custom query
about
TUG
Bioinformatics Group Institute of Biomedical Engineering, Graz Univ. of Technology
Internet
Tissue-Microarray-Database - Microsoft Internet Explorer
TMA-db
Tissue Microarray Images Analysis and Retrieval System
Tissue
Donor
Arrayblock
Section
Staining
Results
Submit

Basic Queries

Choose a query:

Basic Queries

- Query for all arrayblocks inserted within the last two weeks
- Query for all section not yet gridded
- Query for all HE stained section images analysed by algorithms

© 2002 - Bioinformatics Group - Institute of Biomedical Engineering - Graz University of Technology

http://10.2.0.22:8180/tmaweb/standardquery.do?action=display

custom Queries

Compose your query:

☐	Date (from) 01.01.1900	☐	Barcode contains
☐	Date (to) 22.10.2003	☐	entered by
☐	Project project1	☐	Type 320

Choose entries to display:

☐	Date	☐	entered by
☐	Project	☐	BarCodeID
☐	Arraytype		

Find results, matching:
[✓ any field]
all fields

© 2002 - Bioinformatics Group - Institute of Biomedical Engineering - Graz University of Technology

(a) (b)

Figure 5.9: Predefined queries can be performed easily using the “basic query web page” (Fig. 5.9(a)). Complex queries can be set up using the “custom query page”. Fig. 5.9(b) illustrates the available options for donor parameters.

The custom query page is split up into six parts, where several query options can be configured. For each part, the user can choose the fields affected and the fields returned by the query. The following options are available for each part:

Tissue: The user can narrow down the results to a specified ICD-O code.

Donor: By defining a period of time the user can reduce the number of returned results to datasets, which are inserted within the given period of interest. Additionally the user can enter parts of the barcode or can search for datasets, which was inserted by a certain person.

Arrayblock: Options for arrayblock related data are: Person, entered the arrayblock, Arraytype, period of time, and barcode-ID

Section: For section related data the user can select to refine the query by specifying constraints on thickness of a section, barcode-ID, date range, person, who have inserted the arrayblock, related project, and whether a section is already split up into punch images or not.

Staining: Options for staining informations are: The used antibody, antigen, dye. Further a period of time, the a persons name can be specified.

Results: The user can control the query with constraints on the following parameters: Type and value of the result and the type of algorithm, which has produced a result.

When composing a custom query, the user can select for each part separately, whether a query should match *any* (logic OR) or *all* (logic AND) of the given constraints. The various parts themselves are combined through a logical AND operation.

Chapter 6

Results and Discussion

In this project, a tissue microarray database application for storing and maintaining the data generated during the tissue microarray production process was developed. The information comprises pathological information, images from TMA slides acquired by a laser scanner, and results of analyses, generated from the collected data. Special emphasis was given to the flexible and efficient storage of results from the analysis of the punch images to support currently known analyses, but to also allow for easy integration of newly developed methods.

As the system basically depends on the stored data, the most crucial part was to develop a robust and flexible database structure, together with a powerful business logic layer, processing this data. The combination of a relational database management system, the latest Java technologies for server-side computing and web-based development offered a rich development environment to achieve the required flexibility and efficiency.

Since all computational intensive tasks have been embedded within an application server using the Enterprise JavaBeans (EJBs) technology, the system becomes scalable and fail-safe. It can be maintained and upgraded without the need of client side reconfiguration or re-installation. Because analyses run on the server-side, the client machines can be less powerful, even if multiple series of analyses are activated on thousands of punch images concurrently. Clients can log out after the analysis jobs are initiated, and it is not necessary to wait for completion.

This application is designed for pathologists and clinicians, working at different locations. For this reason it is important to provide an easy way to access the application and it's stored data. Thus a web client application is the best solution to meet this objective.

Access to the application has to be granted to authorised personnel only, which is ensured by the integration of a powerful user management system. It allows restriction of database access to only those people having appropriate admissions. It is possible to define different access levels. With simple reconfiguration (via a web interface) a new user can be created, who can only view some data, but is not allowed to perform data manipulation. The user management was also developed at the Institute of Biomedical Engineering at Graz University of Technology by Dieter Zeller. [41]

In future work, new digital image analysis algorithm will be added to the system, to collect more results and refine the conclusions drawn from it.

The user interface will be improved and features will be added, which come up during extensive usage of the system.

This web-based database application offers the ability for groups of pathologists and other clinical staff to collect TMA related data on a central database host. All participants can always view and query the data, which is produced in their working group. Hence, this database should become a valuable tool for all scientists dealing with tissue microarrays, for collecting data on one side, and for gaining new results by running the integrated analyses on thousands of punch images automatically. The latter feature enables the system to perform high-throughput molecular profiling of tissue specimens.

The system is currently installed and in use at the Institute of Pathology at the district hospital in Graz (LKH - Landeskrankenhaus).

Acknowledgements

I am indebted to my colleagues at the Institute of Biomedical Engineering at Graz University of Technology, the Institute of Pathology at Graz University, and to ORIDIS Biomed, GmbH, Graz, who have provided invaluable help and feedback throughout the course of this project. I especially wish to thank my supervisor, Prof. Zlatko Trajanoski, and my thesis advisor, Gerhard Thallinger, for their readiness and throughtfulness in answering questions.

Special mention also goes to my brother, Andreas Pirklbauer, and the many other colleagues and friends who have spent countless hours providing insightful comments and reviewing early draft versions of this thesis.

Last but not least, I would like to thank my family, girlfriend Susanna, and all my other friends. Without their continued support, understanding and patience, this thesis would not have been possible.

Martin Pirklbauer

Graz, Austria, September 2003

Appendix A

User Requirements Document

Before starting implementation of the TMA database application a user requirements document was written. It contains first drafts of the needed input screens as well as some definitions.

The document is printed here for the sake of completeness on the next 11 pages.

As it is in nature of a development process, many things happen during it. Therefore the resulting application became much more comprehensive than it is drawn in the URD. The database structure has changed more than one time, because during development it becomes clear that various adaptations are necessary to implement all those features not considered during the design phase. The resulting database structure is illustrated, for comparison, on page 40, while the first draft can be seen at the end of the URD.

As a final notice, it was very important to spend time (approximately 2 month) creating a user requirements document. During many, many sessions with pathologists, computer scientists and chief executives, *before* starting implementation, the main goals of this project were defined and the vocabulary used by pathologists and computer scientists was synchronised. While this effort was very painful at the beginning, the time spent for this purpose paid off multiple times during the implementation.

Database for Tissue MicroArray Images

User Requirements Document

Martin Pirklbauer [*]

20. February 2003

Examiner:

Ao.Univ.-Prof., Dipl.-Ing. Dr.techn. Zlatko Trajanoski

Advisor:

Dipl.-Ing. Gerhard Thallinger

[*]MatNr: 9530997

Table of Contents

1 Introduction

The intention is to design and create a database application for Tissue Micro Array Images. It should be possible to collect all relevant data which occurs during the creation process of tissue micro arrays and store them into a database.

These data are related to the tissue itself, the donor, the arrayblock, sections and the punches. For each of these items there will be a possibility to enter characterisation information and accordingly images into the database.

2 Glossary

Tissue General Tissue

Donor(block) To enable further process on the tissue it is enclosed into an paraffin block. This block is called the Donorblock.

ArrayBlock The TMA-Arrayer takes samples cores from the donorblock and insert them into a recipient block where the sample cores form a grid. The recipient paraffin block is called "Arrayblock".

Section Each Arrayblock is cut into several Slides which are "Sections" with about 6mm thickness.

Punch A Punch is one Spot on a slide. The diameter of one Punch is about 0,6mm and the center-to-center distance between two punches is about 0,8mm.

Staining Method of preparation of each section.

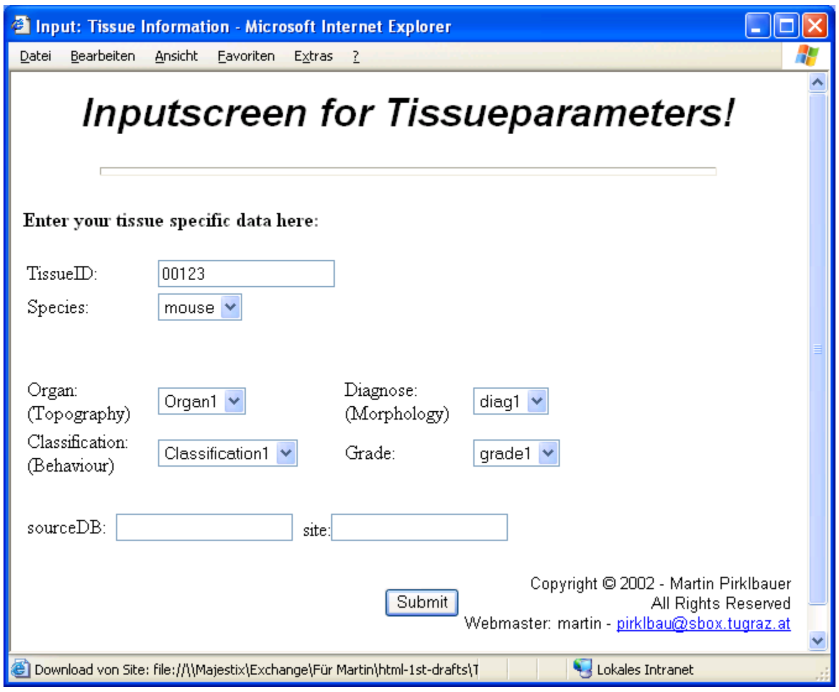
3 Userrequirements

UR 1: Inputscreen for register tissue related Information

After building the paraffin block (Donorblock) this inputscreen should be used to enter the follwing tissue related data:

- TissueID (automatically generated)
- species (human, mouse, other)
- Topography (organ)
- Morphology (diagnose)
- Behaviour (classification)
- grade
- site (localisatition of the tissue, e.g. lungs
- sourceDB

For topography, morphology, behaviour and grade a dropdown-list is provided to select one of a set of predefined possibilities for a diagnose. For site and sourceDB a textfield is provided. Figure 1 shows a first draft of this inputscreen. This screen (and all the following) will be embedded in a main site wich will include basic navigation features.


Input: Tissue Information - Microsoft Internet Explorer

Inputscreen for Tissueparameters!

Enter your tissue specific data here:

TissueID:	00123		
Species:	mouse		
Organ: (Topography)	Organ1	Diagnose: (Morphology)	diag1
Classification: (Behaviour)	Classification1	Grade:	grade1

sourceDB: site:

Submit

Copyright © 2002 - Martin Pirklbauer
All Rights Reserved
Webmaster: martin - pirklbau@sbox.tugraz.at

Download von Site: file://\\Majestix\Exchange\Für Martin\html-1st-drafts\T

Lokales Intranet

Figure 1: Inputscreen for tissueinformationen

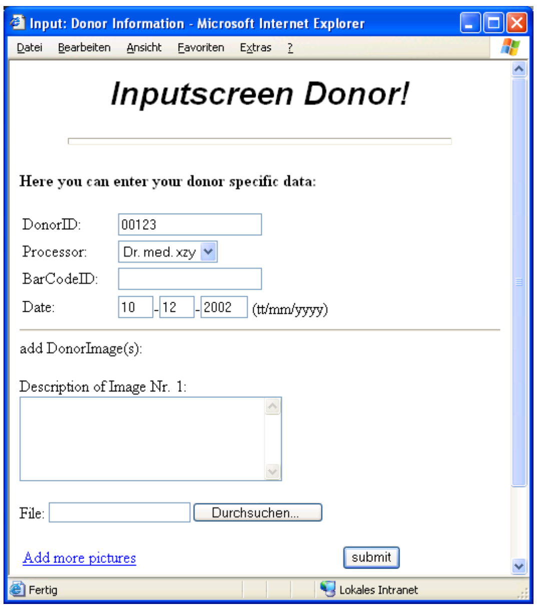
Once the tissue related informations are registered a screen for donor specific data will be provided:

UR 2: Inputfacility: Donorinformation

- Date of Donorcreation
- Processor: the Person who has created the donorblock
- BarCode: each Donor gets a label with an Barcode. The number which is coded into the Barcode will be stored into the database.
- Ability to add one or more Donorimages.

Fig. 2 show a possible inputscreen for this task.

A special coding convention for the barcode is agreed. More Details are explained in UR 8.


Input: Donor Information - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?

Inputscreen Donor!

Here you can enter your donor specific data:

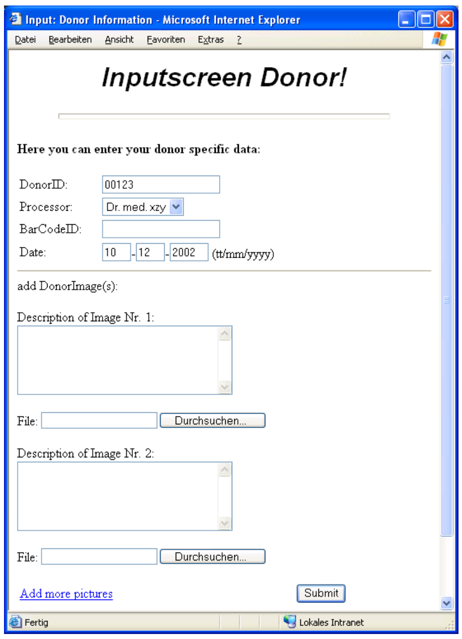
DonorID: 00123
Processor: Dr. med. xzy
BarCodeID:
Date: 10 - 12 - 2002 (tt/mm/yyyy)

add DonorImage(s):

Description of Image Nr. 1:

File:

Durchsuchen...
Add more pictures
submit
Fertig
Lokales Intranet


Input: Donor Information - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?

Inputscreen Donor!

Here you can enter your donor specific data:

DonorID: 00123
Processor: Dr. med. xzy
BarCodeID:
Date: 10 - 12 - 2002 (tt/mm/yyyy)

add DonorImage(s):

Description of Image Nr. 1:

File:

Description of Image Nr. 2:

File:

Durchsuchen...
Durchsuchen...
Add more pictures
Submit
Fertig
Lokales Intranet

Figure 2: Inputscreen for donorinformation (one/more Images)

UR 3: Arrayblockinformation

The TMA-Arrayer robot creates from several donorblocks several arrayblocks. Each created Arrayblock should be marked up with the follwing information:

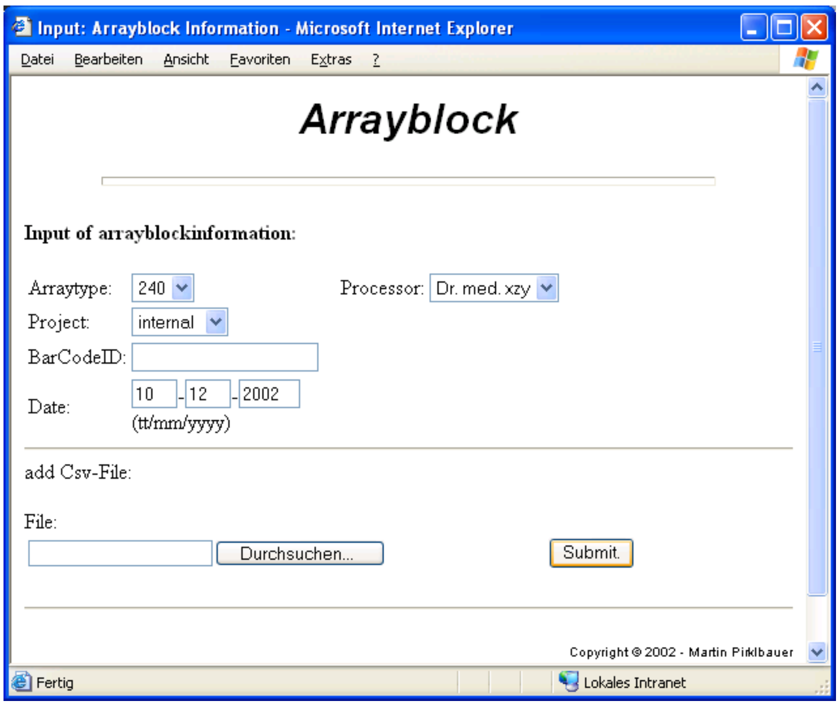
- ArrayBlockID

- BarCode: also each arrayblock gets an Barcode. The Barcodenumber is stored into the databese
- Arraytype: Amount of samples on an arrayblock (240, 322, 407 or 487)
- Processor: the Person who is working on it (Textfield)
- Project: related project (drop-down)
- Date
- csv: The Arrayer generates a csv-file. This file can also be uploaded into the database.

The ability for entering this information is provided.

The unique assignment between donor- an arrayblock is also stored, this means the informations which samples of which donor is on one arrayblock and on which arrayblocks are the samples of one donorblock. These informations are generated from the csv-file.

Figure 3 shows an inputscreen for the necassary information.


Input: Arrayblock Information - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Durchsuchen...
Submit
Fertig
Lokales Intranet

Arrayblock

Input of arrayblockinformation:

Arraytype: 240 Processor: Dr. med. xzy
Project: internal
BarCodeID:
Date: 10 - 12 - 2002 (tt/mm/yyyy)

add Csv-File:

File:

Copyright © 2002 - Martin Pirklbauer

Figure 3: Inputscreen for arrayblockinformation

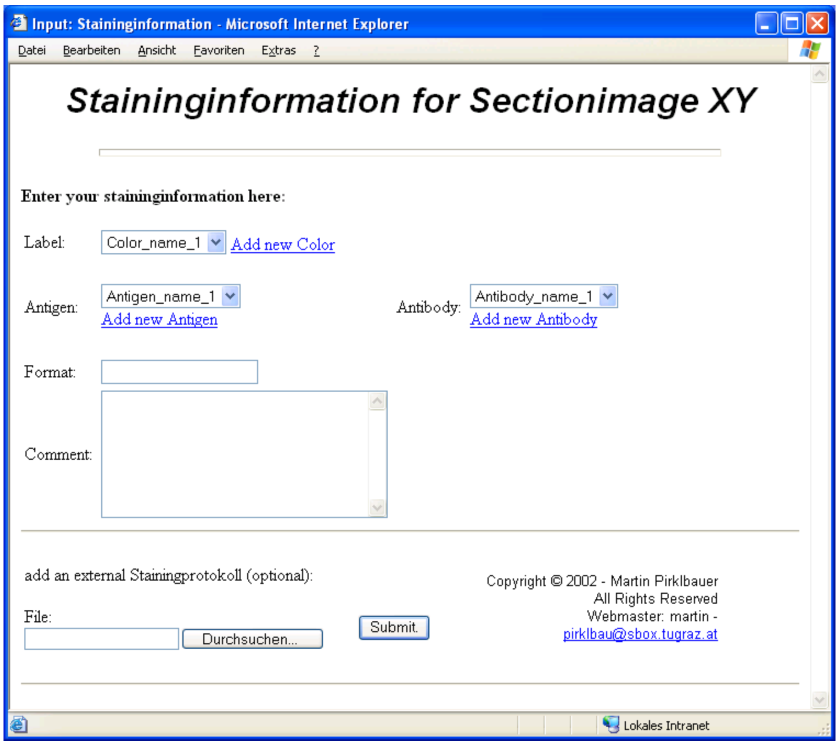
UR 4: Sectioninformation

Per originating section the follwing data should be entered into the datenbase:

- A unique allocation to a specific arrayblock (ArrayBlockID)
- SectionNumber: a consecutive number for each section of an arrayblock.
- Barcode
- Thickness of the specific slide
- Processor: Person who is working on it
- Project: related project (drop-down)
- Date
- One or more images of the section.
- Staininginformation per sectionimage (see UR 5)

The inputscreen for section related data will be similar to the screen for donor information. For sure, with the parameter for section information and including the abitlity to add one or more images per section. Additionally you can add H&E-Information to a specific section image (textfield).

UR 5: Staininginformation



Staininginformation for Sectionimage XY

Enter your staininginformation here:

Label: Color_name_1

Antigen: Antigen_name_1

Antibody: Antibody_name_1

Format:

Comment:

add an external Stainingprotokoll (optional):

File:

Copyright © 2002 - Martin Pirklbauer
All Rights Reserved
Webmaster: martin - pirklbau@sbox.tugraz.at

Figure 4: Inputscreen for staininginformation

The Staininginformation consists basically of a unique assignment to a specific sectionimage and the follwing parameters: (see Fig. 5)

- Label: The used dye. It will be chooseable (drop-down) with the ability to add new labels the list of already stored labels.
- Antigen: The used Antigen; it will be chooseable and you also have the possibility to add new antigens to the stored list in the database. To each antigen you can enter the correspondig gen.
- Antibody: The used Antibody; it will be chooseable and you also have the possibility to add new antigens to the stored list in the database.
- Format (Textfield)
- Comment: To be flexible for future, a big commentfield will be provided in which you can add additional rarly used Stainingparameters.
- Stainingprotocol-File: optional you can add an external File which can contain a more detailed stainingprotocol

UR 6: Punchimages and correlation of punches to an arrayblock

The several punches per section will be stored in the database. This takes place using direct access to the database (over jdbc). An external program will read the sectionimages and the corresponding csv-file from the database, generates the punchimages and stores them back into the database with some additional information per punchimage:

- a unique relation to a specific section
- a unique relation to a donorblock
- number of row and column of the punch on the current section

UR 7: Results, Analysis per Punch

Availiable algorithms for further analysis of punchimages should be embedded into the database application. There will be a userinterface where you can fire up already embedded algorithms.

A framework for embedding new algorithms as easy as possible will be provided.

The external program ImageGene produces also a csv-file. This file will be parsed and the content of this file is automatically stored as results in the database. The following information will be extracted from the ImageGene-csv-file:

- Signal Mean
- Signal Median
- Signal Area
- Signal Total
- Signal Stddev (standard deviation)
- Background Mean
- Background Median

- Background Area
- Background Total
- Background Stddev
- Shape Regularity
- Diameter

UR 8: BarCode Codingconvention

The donorblock, the arrayblock and each Section from an arrayblock will receive a barcodelabel. The numbers of these barcodes are also stored in the database.

Each barcode consists of 13 digits, 12 of them are for the code and one is a check digit. The 12 code digits denotes as described below:

	Type	Sectionnumber	Year	consecutive. Nr.
Nr. of digits	2	3	2	5
Example	03	008	03	00002

Figure 5: Strukture of a barcode

The type-digits denotes the kind of material.

Type	kind
01	Donor
02	Arrayblock
03	Section

If we are dealing with sections, then the sectionnumber is the consecutive number of the sectionslides of an arrayblock. If we are dealing with a donorblock or an arrayblock, the sectionnumber will be 0.

The next two digits are reserved for the year of the production of the materila. The latter 5 digits denotes a general consecutive number.

In the example above, we deal with the 2^{nd} sectionimage of the 8^{th} section. The year of production is 2003.

UR 9: Queries

There will be a set of simlpe and some complex possibilities of queries onto the collected data. A simple query will be for example: Show all sectioninformation from section with barcode xy. A bit more complex query could be for instance: Show all punch related data from punches which have as result xy having value between a and b.

Here a List of some defined queries:

- Show all cases which have a specified diagnose and where the value of a certain parameter is between a and b.

- List all antibodies which match a certain antigen.
- Search all arrays dyed over a specific period.

All query results should be displayed in a convenient fashion preferable in tabular form. The recovered results will be linked with more detailed information related to the specific result, which can be viewed if required, e.g. several punchimages.

UR 10: Datenbankstructure - DBMS

The used database structure is illustratet in the appendix and should not be changed anymore (as far as possible). The choosen DBMS is the mySQL-database with an extension of innoDB-table support. This choice provides a relatively slim database system which can be easily installed on the target computer. The innoDB extension is important due to technical reasons (foreign-key-constraints, "huge" tables).

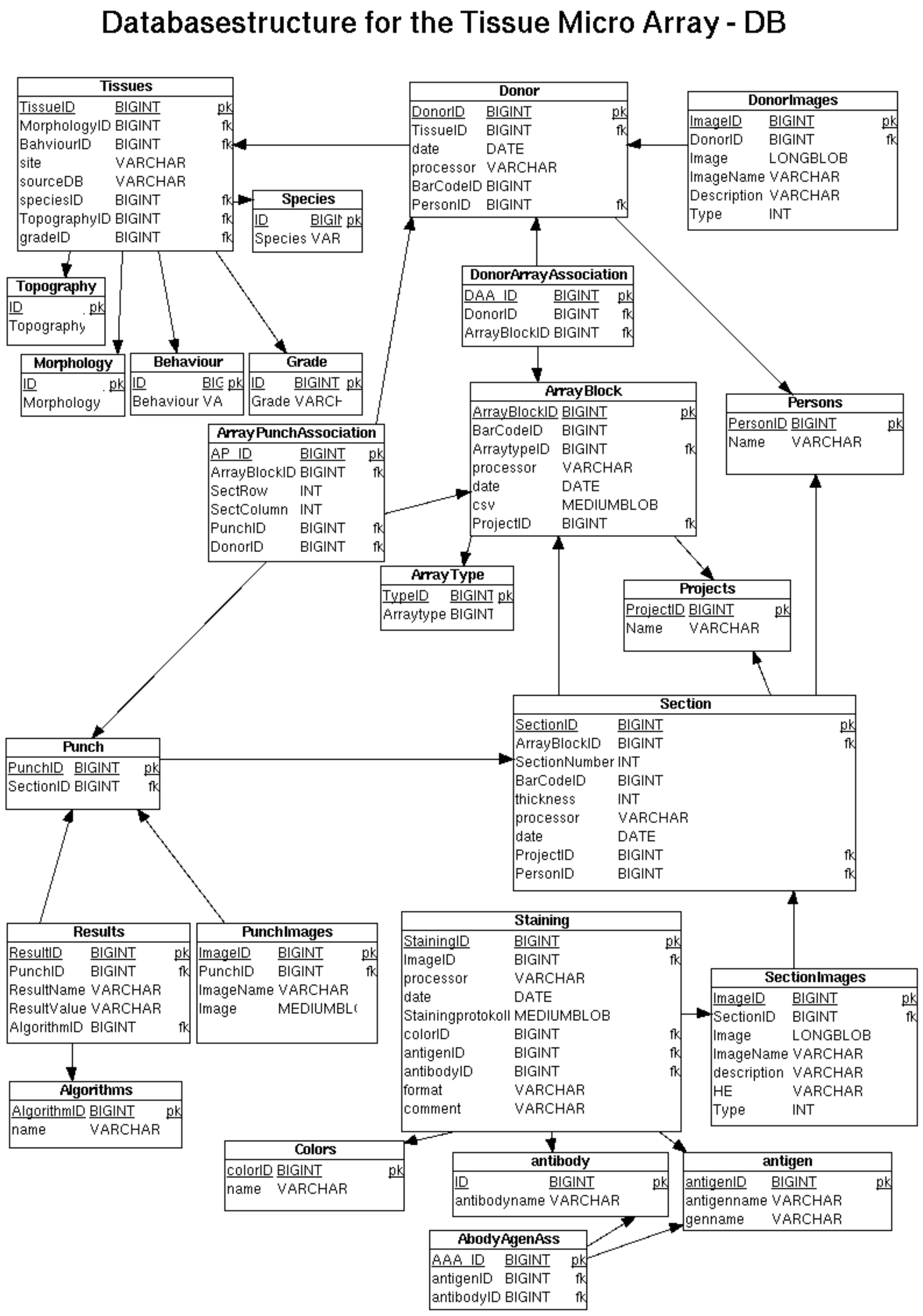


Figure 6: Databasestructure

Appendix B

Installation instructions

This section provides detailed installation instruction for setting up the complete TMA database application on a “Microsoft© Windows .NET Server 2003 Enterprise Edition” operating system. The installation is divided into three main parts: Database-, application server- and web application-installation. As the application server and the web server are pure Java programs, installation of the Java runtime environment (JRE) is mandatory. The database is a MySQL server with InnoDB support. InnoDB support is already integrated into MySQL since release 4.0, thus in this project MySQL 4.0.13 is used. For the application server JBoss 3.2.1 is used and Tomcat 4.1.24 from the Jakarta Group is appointed as a web server, both are available in a bundled package provided by the JBoss Group. This combined package is used for the installation.

B.1 Prerequisites

Make sure to have access to the `serverinstall` directory, which is located at the CVS-server at the Institute of Biomedical Engineering at Graz University of Technology, which is from now abbreviated with `<install-dir>`. The path at the CVS server is `tmaEjb/documentation/server-install`. Copy all files from this directory to a temporary `<install-dir>` folder on your local hard disk.

B.2 MySQL 4.0.13 - Setup

Installation:

```
*) Download MySQL 4.0.13 from
   http://www.mysql.com/downloads/mysql-4.0.html and install it
   using the install wizard.  When the installation program asks
   for the installation directory, choose: C:\MySQL.
*) Run C:\MySQL\bin\winmysqladmin.exe.
   MySQL is now installed as a Windows Service and is started
   automatically at boot time.
```

Configuration:

```
*) Create a directory at E:\TMADBDATA for the database files.
   This directory will contain all data stored into the database
   and should therefore have enough free space available.
*) Open the MySQL main configuration file C:\Windows\my.ini
   with your favoured text editor (e.g., with the Windows notepad)
   and set the path of variable ’datadir’ to datadir=E:\TMADBDATA.
*) Edit variable ’set-variable = max_allowed_packet’ and set it to
   ’set-variable = max_allowed_packet=64MB’. Close the
   configuration file C:\Windows\my.ini.
```

Configuration 2:

```
*) Open a Windows console and change to the directory:
   C:\MySQL\bin (type: ’cd C:\MySQL\bin’).
*) Run the following command in given order:
    mysql -u root < <install-dir>\TMAdb-CreateTables-only.sql
    mysql -u root < <install-dir>\data.sql
    mysql -u root < <install-dir>\TMAdb-fkContraints-only.sql
    mysql -u root < <install-dir>\usermanagement-full.sql
    mysql -u root < <install-dir>\mysql-addtmauser.sql
*) Restart the MySQL Server using the "Services" Snap-In from the
   Administrator Console (Start -> Settings -> Control Panel;
   Administrative Tools -> Services; Choose "MySQL" and select
   restart form the context menu.)
```

B.3 Java runtime environment - Setup

```
*) Download the Java runtime environment (JRE) from
   http://java.sun.com/j2se/1.4.1/download.html and install it to
   C:\Java\J2SDK1.4.1
*) Set the system environment variable ’JAVA_HOME’ and point it to
   the JRE installation directory (C:\Java\J2SDK1.4.1).  This can
   be done in the Start -> Settings -> Control Panel; System ->
   Advanced -> Environment-Variables dialog. Set the environment
   variable system wide by using the ’New’ button in the lower
   ’System variables’ area.
```

B.4 JBoss 3.2.1 and Tomcat 4.1.24 - Setup

```
*) Download the JBoss-3.2.1+Tomcat-4.1.24 software bundle from
   http://www.jboss.org/downloads
*) Unpack the archive to 'C:\Java\jboss-3.2.1_tomcat-4.1.24'
*) Add the MySQL-JDBC Connector to JBoss by copying the file
   <install-dir>\mysql-connector-java-3.0.6-stable-bin.jar to the
   C:\Java\jboss-3.2.1_tomcat-4.1.24\server\default\lib - directory.
*) Copy the predefined datasource definition file (named
   'mysql-ds.xml') from the <install-dir> to directory:
   C:\Java\jboss-3.2.1_tomcat-4.1.24\server\default\deploy
   The mysql-ds.xml file must look like as follows:

    <datasources>
      <local-tx-datasource>
        <jndi-name>MySqlDS</jndi-name>
        <connection-url>
           jdbc:mysql://localhost/TMA-TEST
        </connection-url>
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <user-name>tmadbuser</user-name>
        <password>tmadbuser</password>
      </local-tx-datasource>
      <local-tx-datasource>
        <jndi-name>MySqlDSUM</jndi-name>
        <connection-url>
          jdbc:mysql://localhost/usermanagement
        </connection-url>
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <user-name>tmadbuser</user-name>
        <password>tmadbuser</password>
      </local-tx-datasource>
    </datasources>

*) Edit file:
   C:\Java\jboss-3.2.1_tomcat-4.1.24\server\default\
   conf\standardjbosscmp-jdbc.xml
   and add the following lines inside the  <defaults>..</defaults>
   tag:

      <datasource>java:/MySqlDS</datasource>
      <datasource-mapping>mySQL</datasource-mapping>
      <create-table>false</create-table>
      <remove-table>false</remove-table>
      <entity-command name="mysql-get-generated-keys"/>

   Uncomment everything else inside the defaults environment, by
   enclose the original entries with the <!-- and --> characters.
*) Edit file:
   C:\Java\jboss-3.2.1_tomcat-4.1.24\server\default\
   conf\jboss-service.xml
   Find the line containing:
     <attribute name="RecursiveSearch">False</attribute>
   and change it to:
     <attribute name="RecursiveSearch">True</attribute>
   hint: JBoss expects a special deployment order for correct
   initialisation of the Java Messaging Service (JMS), which
   can not be predicted at boot time.
   This setting forces JBoss to initialise the JMS recursively
   until it succeeded and is therefore independent of the
   deployment order.
*) Copy the files, located in the <install-dir>:
    tmaejb.jar
    tmaAlgorithms.jar
    tmaAlgorithmAdapter.jar
    RGcalibration.jar
   into the
   C:\Java\jboss-3.2.1_tomcat-4.1.24\server\default\deploy
   directory.
*) Create a directory named 'tmaweb.war' in the deploy directory:
   (C:\Java\jboss-3.2.1_tomcat-4.1.24\server\default\
   deploy\tmaweb.war)
*) Unpack the contents of the <install-dir>\tmaweb.war file
   with your favoured zip-tool to this directory.
*) Edit file:
   C:\Java\jboss-3.2.1_tomcat-4.1.24\server\default\
   deploy\tmaweb.war\Server.properties
   and change the IP address in line:
   java.naming.provider.url=jnp://ww.xx.yy.zz:1099 to the IP
   address of the server machine, you are currently installing by
   replacing ww.xx.yy.zz with your IP. Hint: The local IP address
   can be determined by opening a command shell (Start -> Run: cmd)
   and by typing: 'ipconfig'.
*) Edit file:
   C:\Java\jboss-3.2.1_tomcat-4.1.24\server\default\
   deploy\tmaweb.war\WEB-INF\UserManagement.properties
   and set the following variables:

       protocol=HTTP
       http_url=localhost
       http_port=15080

*) Edit file:
   C:\Java\jboss-3.2.1_tomcat-4.1.24\bin\run.bat
   Find the line containing:
   'set JAVA_OPTS=%JAVA_OPTS%' and change it to:
   'set JAVA_OPTS=%JAVA_OPTS% -Xms128m -Xmx512m'
*) Start the application server by running:
   C:\Java\jboss-3.2.1_tomcat-4.1.24\bin\run.bat
*) see next section, to see how the JBoss application server is
   installed as a windows service.
```

Install JBoss as a Windows Service

```
*) Copy file <install-dir>\lcp.bat to
   C:\Java\jboss-3.2.1_tomcat-4.1.24\bin
*) Copy files
    <install-dir>\InstallJBossService.cmd and
    <install-dir>\JBoss.exe to
   C:\Java\jboss-3.2.1_tomcat-4.1.24\
*) Open a command shell (Start -> Run: cmd) and change
   to directory C:\Java\jboss-3.2.1_tomcat-4.1.24\ by
   typing 'cd C:\Java\jboss-3.2.1_tomcat-4.1.24\' and execute
   the following command:
   'InstallJBossServive.cmd "JBoss32-TMA-DB" default'
   (note: type in the double quotes!). You can now start and stop
   the application server by using the Services Snap-In from the
   Windows Administrator Toolbox.
   The service is named JBoss32-TMA-DB.
*) After user management integration the TMA-DB application is
   available with a web browser at the following URL:
   http://localhost:8080/tmaweb/index.html
```

B.5 User management integration

```
*) From now it is assumed, to have the files:
    - UserManagementEJB.jar
    - Web.war
   provided by Dieter Zeller available.
*) Download the JBoss-3.0.5+Tomcat-4.0.6 software bundle from
   http://www.jboss.org/downloads
*) Unpack the archive to 'C:\Java\jboss-3.0.5_tomcat-4.0.6'
*) Add the MySQL-JDBC Connector to JBoss by copying the file
   <install-dir>\mysql-connector-java-3.0.6-stable-bin.jar to the
   C:\Java\jboss-3.0.5_tomcat-4.0.6\server\default\lib - directory.
*) Copy file:
   C:\Java\jboss-3.0.5_tomcat-4.0.6\docs\examples\
   \jca\mysql-service.xml
   to C:\Java\jboss-3.0.5_tomcat-4.0.6\server\default\deploy\
*) Open file:
   C:\Java\jboss-3.0.5_tomcat-4.0.6\server\default\
   \deploy\mysql-service.xml
   find section:

     <attribute name="JndiName">MySqlDS</attribute>
     <attribute name="ManagedConnectionFactoryProperties">
       <properties>
         <config-property name="ConnectionURL"
            type="java.lang.String">jdbc:mysql://dell:3306/jbossdb
         </config-property>
         <config-property name="DriverClass"
            type="java.lang.String">org.gjt.mm.mysql.Driver
         </config-property>
         <config-property name="UserName" type="java.lang.String">
         </config-property>
         <config-property name="Password" type="java.lang.String">
         </config-property>
       </properties>
     </attribute>

    and change it to:

     <attribute name="JndiName">MySqlDS</attribute>
     <attribute name="ManagedConnectionFactoryProperties">
       <properties>
         <config-property name="ConnectionURL"
           type="java.lang.String">
             jdbc:mysql://localhost:3306/usermanagement
         </config-property>
         <config-property name="DriverClass"
           type="java.lang.String">
             com.mysql.jdbc.Driver</config-property>
         <config-property name="UserName" type="java.lang.String">
            testtma
         </config-property>
         <config-property name="Password" type="java.lang.String">
            testtma
         </config-property>
       </properties>
     </attribute>
*) Open file:
   C:\Java\jboss-3.0.5_tomcat-4.0.6\server\default\
   \conf\standardjbosscmp-jdbc.xml
   and change the parameters 'datasource' and 'datasource-mapping'
   in the 'defaults' section to:

  <defaults>
      <datasource>java:/MySqlDS</datasource>
      <datasource-mapping>mySQL</datasource-mapping>
      <<... leave other parameters unchanged! ...>>
  </defaults>

*) Copy files 'UserManagementEJB.jar' and 'Web.war' into:
   C:\Java\jboss-3.0.5_tomcat-4.0.6\server\default\deploy\
*) start JBoss for user management by running:
   C:\Java\jboss-3.0.5_tomcat-4.0.6\bin\run.bat
*) After the start process has completed you can access the
   user management configuration tool with a web browser
   at the following location:
   http://localhost:15080/Web/index.html
   You can log in as administrator with username=admin and
   password=admin (change this after first login)
*) Please refer to paragraph ’Install JBoss as a Windows Service’
   in previous section (’JBoss 3.2.1 and Tomcat 4.1.24 - Setup’)
   in order to run  JBoss-3.0.5 for the user management
   as a Windows service.
```

B.6 Integration of additional analysis algorithms

```
*) Copy the jar-file, containing the analysis algorithm ’xyz’
   (i.e., the file xyz.jar contains a session bean that
    implements an algorithm and this session bean has the
    JNDI-name ’xyz’)
   into the directory:
   C:\Java\jboss-3.2.1_tomcat-4.1.24\server\default\deploy
*) Log into the TMA web application and choose menu item
   ’Algorithms->Add algorithm’
*) Enter the JNDI name of the algorithm (i.e. ’xyz’) into
   the provided input field and click the button ’submit’.
```

Bibliography (Biomedical science)

[1] Liu C.L., Prapong W., Natkunam Y., Alizadeh A., Montgomery K., Gilks B., and van de Rijn M. Software tools for high-throughput analysis and archiving of immunohistochemistry staining data obtained with tissue microarrays. *American Journal of Pathology*, 161(5):1557–1565, 2002.

[2] M. Diehn, A. Alizadeh, and P.O. Brown. α-methylacyl coenzyme a racemase as a tissue biomarker for prostate cancer. *American Medical Association*, 287:1662–90, 2002.

[3] O.P. Kallioniemi, U. Wagner, J. Kononen, and G. Sauter. Tissue microarray technology for high-throughput molecular profiling of cancer. *Human Molecular Genetics*, 10(7):657–622, 2001.

[4] J. Kononen, L. Bubendorf, A. Kallioniemi, et al. Tissue microarrays for high-throughput molecular profiling of tumor specimens. *Nature Medicine*, 4:844–847, 1998.

[5] A. Nocito, O.P. Kallioniemi J. Kononen, and G. Sauter. Tissue microarrays (tmas) for high-throughput molecular pathology research. *International Union Against Cancer*, 94:1–5, 2001.

[6] Shaknovich R., Celestine A., Yang L., and Cattoretti G. Novel relational database for tissue microarray analysis. *Arch Pathol Lab Med.*, 127:492–494, 2003.

[7] WHO - World Health Organization. *The International Statistical Classification of Diseases and Related Health Problems*.
last visited on 2003-09-12,
`http://www.who.int/whosis/icd10/`.

Bibliography (Computer science)

[8] R. Adatia, F. Arni, and K. Gabhart. *Professional EJB*. Wrox Press, 2001.

[9] K. Avedal, D. Ayers, and T. Briggs. *JSP Professionell*. MITP-Verlag, 2001.

[10] S. Bodoff. Java Servlet Technology. Technical report, Sun Microsystems, 2003.
last visited on 2003-09-12,
`http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html`.

[11] Date C.J. and Darwen H. *A Guide to SQL Standard*. Addison-Wesley Pub Co, 4 edition, 1997.

[12] D. Coward. Java Servlet Specification. Technical report, Sun Microsystems, 2003.
last visited on 2003-09-12,
`http://jcp.org/aboutJava/communityprocess/first/jsr154/index3.html`.

[13] L. G. DeMichiel. Enterprise JavaBeans Technology - Specifications. Technical report, Sun Microsystems, 2001.
last visited on 2003-09-12,
`http://java.sun.com/products/ejb/docs.html`.

[14] Codd E.F. Derivability, redundancy, and consistency of relations stored in large data banks. Research Report RJ599, IBM, August 1969.

[15] Codd E.F. *The Relational Model for Database Management Version 2*. Addison-Wesley, 1991.

[16] O. Gdalevich. Introduction to sql.
last visited on 2003-09-12,
`http://www.vbip.com/books/1861001800/chapter_contents.asp`.

[17] D. Green. Enterprise javabeansquery language. Technical report, Sun Microsystems, 2002.
last visited on 2003-09-12,
`http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBQL.html`.

[18] JBoss Group. JBoss Group - Home Page.
last visited on 2003-09-23,
`http://www.jboss.org`.

[19] K. Haase. Java message service tutorial. Technical report, Sun Microsystems, 2002.
last visited on 2003-09-12,
`http://java.sun.com/products/jms/tutorial/index.html`.

[20] M. Hall. *Servelts and Java Server Pages*. Prentice-Hall PTR, 2000.

[21] G. Hamilton, R. Cattell, and M. Fisher. *JDBC Database Access with Java: A Tutorial and Annotated Reference*. Addison-Wesley, 1997.

[22] IBM. Servlets. Technical report, IBM, 2001.
last visited on 2003-09-12,
`http://www-4.ibm.com/software/webservers/appserv/doc/v30/ae/web/doc/whatis/icsrvlet.html`.

[23] InnoDB.
last visited on 2003-09-22,
`http://www.innodb.com/features.html`.

[24] JSPTags.com. TagLibLinks. Technical report, JSPTags.com, 2000.
last visited on 2003-09-12,
`http://jsptags.com`.

[25] P. Litwin. Fundamentals of relational database design.
last visited on 2003-09-12,
`http://ken.slctech.org/PROG67/Fund%20of%20database%20designKJ.pdf`.

[26] Sun Microsystem. Designing enterprise applications with the j2ee platform. Technical report, Sun Microsystems, 2003.
last visited on 2003-09-12,

http://java.sun.com/blueprints/guidelines/designing_enterprise_applications/index.html.

[27] Sun Microsystem. Java documentation of classes and interfaces of the Servlet und JSP technology. Technical report, Sun Microsystem, 2003.
last visited on 2003-09-12,
http://java.sun.com/products/servlet/2.3/javadoc.

[28] MySQL.
last visited on 2003-09-22,
http://www.mysql.com/.

[29] Oracle Corporation.
last visited on 2003-09-22,
http://www.oracle.com/.

[30] F. Pascal. *SQL and Relational Basics*. John Wiley & Sons, 1990.

[31] Elke Pauritsch. Mathematische Modellbildung bei mehrfachgefärbten Gewebeproben in der Immunhistochemie. Master's thesis, TU-Graz, 2003. In german language.

[32] E. Pelegri-Llopart. Java Server Page Specification. Technical report, Sun Microsystems, 2003.
last visited on 2003-09-12,
http://jcp.org/aboutJava/communityprocess/first/jsr053/index.html.

[33] The Jakarta Project. TagLibs. Technical report, The Jakarta Project, 2001.
last visited on 2003-09-12,
http://jakarta.apache.org/taglibs/index.html.

[34] The Jakarta Project. Internationalisation. Technical report, The Jakarta Project, 2003.
last visited on 2003-09-12,
http://jakarta.apache.org/struts/userGuide/building_view.html#i18n.

[35] The Jakarta Project. The Struts User Guide. Technical report, The Jakarta Project, 2003.
last visited on 2003-09-12,
http://jakarta.apache.org/struts/userGuide/index.html.

[36] Monson-Haefel R. *Enterprise JavaBeans*. O'Reilly, 3 edition, 2001.

[37] E Roman. *Mastering Enterprise JavaBeans*. Wiley, 1999.

[38] Sun. The JavaBeans Specification. Technical report, Sun Microsystem Inc., 1997. 2003-09-20, `http://java.sun.com/products/javabeans/docs/beans.101.pdf`.

[39] Sun. Java 2 Platform, Enterprise Edition (J2EE) - Specification. Technical report, Sun Microsystem Inc., 2003. 2003-10-24, `http://java.sun.com/j2ee/download.html#platformspec`.

[40] Martina Uray. Ein mathematisches Modell zur automatischen Stanzentypisierung in der Immunohistochemie (TMA-Analyse). Master's thesis, TU-Graz, 2003. In german language.

[41] Dieter Zeller. Design and development of a usermanagementsystem for molecular biology databases. Master's thesis, TU-Graz, 2003.

Glossary (Biomedical science)

F

FISH	Flourescence *in situ* Hybridization: A process which vividly paints chromosomes or portions of chromosomes with fluorescent molecules. This technique is useful for identifying chromosomal abnormalities and gene mapping, p. 3.

H

HE	Hematoxylin and Eosin - Staining: A method of dyeing tissue samples for examination under a microscope. Hematoxylin and eosin staining is the most commonly used staining technique for tissue study. Also known as H&E staining, p. 2.

I

ICD-O	International Classification of Disease for Oncology: The Second Edition of the International Classification of Diseases for Oncology. A further classification of the ICD-9 designed for use specifically for cancer, p. 39.
IHC	Immunohistochemistry: Technique that uses antibodies to identify and mark antigens expressed by cells in tissues, p. 3.

M

mRNA	messenger RNA: RNA that serves as a template for *protein* synthesis. Each set of three bases, called codons, specifies a certain protein in the sequence of amino acids that comprise the protein. The sequence of a strand of mRNA is based on the sequence of a complementary strand of DNA, p. 3.

R

RNA	Ribonucleic Acid: A sequence of *nucleotides* found in the cytoplasm of cells; it plays an important role in *protein* synthesis and other chemical activities of the cell. There are several classes of RNA molecules, including messenger RNA (mRNA), transfer RNA (tRNA), ribosomal RNA (rRNA), and other small RNAs, each serving a different purpose. Gene expression or transcriptional profiling examines mRNA levels.

S

Section	After the receipient paraffin block has been produced, it is sliced into up to 200 sections using a microtome, each about 2-4 μm of width.

Glossary (Computer science)

A

API	Application Programming Interface: An API is an Interface which is used for accessing an application or a service from a program, p. 36.
Application Server	An application server is a server program in a computer in a distributed network that provides the business logic for an application program. The application server is frequently viewed as part of a three-tier application.

D

DBMS	Database Management System: A general term for software that manages a computer database or group of databases, p. 35.
DCL	Data Control Language: The set of data control statements within the SQL language.
DDL	Data Definition Language: A compaq language that describes the record and file structures of a database.
DML	Data Manipulation Language: The set of data-manipulation statements within the SQL language.

E

EJB	Enterprise Java Beans: The Enterprise JavaBeans specification allows software components written in Java to interact with one another in a distributed environment.

I

i18n	internationalization: often called ”i18n” because 18 is the number of letters in between the ”i” and the ”n”. i18n is the process of designing an application so that it can be adapted to various languages and regions without engineering changes, p. 28.

J

JDBC	often cited as Java Database Connectivity, but JDBC is a registered trademark of Sun Microsystems, p. 36.

O

ODBC	Open DataBase Connectivity: An API provided from Microsoft Inc. for accessing relational databases, p. 37.

S

SQL	Structured Query Language: A high level language used in database programming, p. 35.

U

URD	User Requirement Document, p. 56.
URI	Uniform Resource Identifier: A generic term to describe the methods by which objects (web pages, files) on the web are referred to, p. 27.