Keep it Simple and You Will Finish What You Start

—a case study of the NeuroGenerator database project

Christian Engström

TRITA-NA-Eyynn
Abstract

System development includes many aspects. The standpoint in this masters project is that too much focus on the technical aspects is a great risk for software projects. This report is about the design and the development of a database of 3D brain images, called NeuroGenerator. In this thesis, two alternative database solutions have been compared. The results describe how they differ in architectures, resource requirements, and suitability for the task at hand, i.e. fulfilling the requirements in the NeuroGenerator agreement. The results show that good knowledge of the agreement together with a keep it simple approach enhances the chance for overall success.

Håll det enkelt så blir du färdig med vad du påbörjar

En fallstudie av databasprojektet NeuroGenerator

Sammanfattning

Preface

The master's project is the final part of the Master of Computer Science and Engineering program at the Royal Institute of Technology, Stockholm. In the master's project, the student is supposed to apply the theoretical knowledge gained during the previous years and report the work in a scientific manner. That will also be the case in my master's project. However, I will mostly make use of my experiences from my part time job parallel to school, where I have worked with software development for the last two years. Thus, this thesis will not be very theoretical, instead it will discuss practical issues technicians are faced with but not taught how to handle in school. The master's project was performed at Center for Parallel Computers (PDC), a partner in the NeuroGenerator project [NeuroGenerator, European Commission-LIFE program], [Roland et al., 2001], from October 2002 to March 2003. Supervisors were Gert Svensson, associate director at PDC, and Kjell Lindqvist, at the Department of Numerical Analysis and Computer Science. Examiner from the same department was Professor Stefan Arnborg.
# Contents

1 About the Master's Project
   1.1 Background ............................................. 1
   1.2 Goal of Master's Project ............................ 2
   1.3 Limitations ........................................... 2
   1.4 Outline of Thesis ................................. 2

2 System Prerequisites
   2.1 Presentation of NeuroGenerator .................. 3
      2.1.1 Motivation ........................................ 3
      2.1.2 Objectives ....................................... 4
      2.1.3 System functionality and Usage .............. 4
   2.2 Requirements on the Database System ........... 5
      2.2.1 Modeling Requirements ....................... 5
      2.2.2 Query Requirements .......................... 6
      2.2.3 Production Requirements on the Central system 7
      2.2.4 Production Requirements on the Portable NeuroGenerator Products .............. 8
   2.3 Summary ............................................ 10

3 The Solutions
   3.1 My Choice ........................................... 11
      3.1.1 Architecture ..................................... 11
      3.1.2 Motivation of Choice ........................... 12
      3.1.3 Development Steps .............................. 13
      3.1.4 Resources Spent ................................ 13
      3.1.5 Appraisal of the Solution .................... 14
   3.2 The NeuroGenerator Choice ....................... 14
      3.2.1 Architecture ..................................... 14
      3.2.2 Motivation of Choice ........................... 15
      3.2.3 System Components ............................ 15
      3.2.4 Resources Spent ................................ 18
      3.2.5 Appraisal of the Solution .................... 20
   3.3 Integrated Control of Image Processing Chains ........................................... 22
   3.4 Summary ............................................ 22
4 Results
4.1 Master’s Project Impact ........................................... 24
4.2 Performance ............................................................ 24
4.3 Other Aspects ........................................................... 26
4.4 Summary ................................................................. 27

5 Conclusions
5.1 My Conclusion about the NeuroGenerator Project .............. 28
5.2 My Conclusion ........................................................... 28
5.3 End of the Multidatabase ............................................ 29

Bibliography

A Fundamental Database Theory
A.1 Database System Concepts and Architectures .................... 33
  A.1.1 Basics concepts .................................................. 33
  A.1.2 The Relational Data Model ..................................... 34
  A.1.3 The Object-Oriented Data Model .............................. 36
  A.1.4 The Object-Relational Data Model .......................... 38
  A.1.5 The Multidatabase Architecture .............................. 39
A.2 Summary ................................................................. 43
Chapter 1

About the Master’s Project

1.1 Background

In software development projects the goals might not be perfectly obvious. But even if the goals might be vague or poorly formulated, they still should be identifiable in the agreements and specifications made up by the involved parties. Technicians often prefer generality over simplicity, general solutions that can handle future changes in conditions and demands. These general solutions usually demand bigger efforts than less general ones. Sometimes the efforts are too big to be handled within the scope of a project and may risk the overall success. My supervisor Gert Svensson, project technical manager, and the project manager Professor Per Roland at Karolinska Institute felt that this was the case in the NeuroGenerator project [NeuroGenerator], [European Commission-LIFE program], [Roland et al., 2001]. The group of database experts in the project had designed a very general system, which also required the development of many software components, some rather complicated. The finish line was getting closer and there were still many doubts about the solution. Would they be be finished with the solution in time, and how would the system work in production? The database group was still confident and could always present a work-around to any problem. The group was not easy to argue with and would always have the final word. I, who had worked as a programmer for one and a half year in the project, had become more and more critical to their solution due to a long list of practical problems. Therefore Gert Svensson and Per Roland proposed that I, as my master’s project, would: design, implement, and evaluate a less complicated solution, still fulfilling the requirements in the agreement[Agreement]. A solution which also could be used as a backup plan if necessary.

The NeuroGenerator development work has involved many design issues and many choices have been made, often with focus on generality. The NeuroGenerator agreement only prescribes the requirements and tasks at a conceptual level and other choices could have been made that also satisfied the requirements.
1.2 Goal of Master’s Project

The goal of this master’s project is to compare a *keep it simple* focused solution for the NeuroGenerator system with one with focus on *generality*. To do this, it is necessary to clarify the goals and present and implement a system prototype with minimal complexity and compare it with the system solution originally chosen in the NeuroGenerator project. The comparisons will be made with respect to performance, functionality, generality, resources spent, and production aspects.

1.3 Limitations

The NeuroGenerator system is not only a database system. It also includes fully automatic image processing chains, driven by the contents of the database. This thesis will not discuss these chains. Instead it will focus on different database system solutions that are able to handle requirements on modeling of unprocessed and processed image data and queries over non-homogeneous data sets.

1.4 Outline of Thesis

In section 2.1, the NeuroGenerator project will be presented together with motives and objectives. To get a clear view of the primary goals with the NeuroGenerator system, section 2.2, offers an investigation of requirements and tasks to be handled by the system. Section 3.1 presents and discusses my system solution and section 3.2, the solution chosen by the database experts in the NeuroGenerator project. Results and comparison aspects are presented in chapter 4. Finally, chapter 5 summarizes the thesis.

To readers not familiar with the different types of Database Management Systems (DBMSs) and database architectures, fundamental concepts in database theory are offered in appendix A.
Chapter 2

System Prerequisites

2.1 Presentation of NeuroGenerator

NeuroGenerator is a database system under development for analyzing functional brain images [NeuroGenerator], [Roland et al., 2001], [European Commission-LIFE program]. The system is meant to be a research tool for the neuro-imaging community in the EU. The NeuroGenerator project is a collaboration between Karolinska Institute, KTH, Uppsala University, Forwiss, and Active Knowledge supported by the European Commission under the LIFE program (Quality of Life and management of Living Resources). The partners are active in different research fields and contribute to the project with knowledge in Neuro-Science, Computer Science, Database Theory, and Image Processing.

The following three subsections recapitulate what is written in the NeuroGenerator system documents [Roland et al., 2002], [Technical Annex]. Knowledge about the motives and objectives behind the NeuroGenerator project is essential to understand the functional needs and system solutions chosen to fulfill these.

2.1.1 Motivation

Functional mapping of the brain is a rapidly evolving area of research. Mappings based on neuroimaging with positron emission tomography (PET) and functional magnetic resonance imaging (fMRI) are powerful methods to investigate the function of the human brain. Neuroimaging facilities represent a considerable European investment. Almost one hundred groups are active in investigating the functions of the human brain with these methods. The dominant research methodology today consists of measuring the level of activity in different regions when subjects perform different types of tasks. Neuroimaging with PET and fMRI results in three-dimensional images of brain activity. To interpret these data, a set of image processing techniques are required, involving computational steps such as segmentation of the brain, transformation of the data to standard anatomical format, and characterization of regions of locally increased or decreased activity. Due to poor signal to noise ratio in the brain activation images, the task of analyzing these data
is non-trivial, and methodologies such as pooling data over subjects and fitting of statistical models to the data have been developed. Usually, the analysis of brain activation images is performed at the level of individual experiments. In this way, the experimenter may draw conclusions about the influence of the factors that were present in his specific experiment and obtain information about the brain regions that were activated by these. Investigating the functional contribution of even a single part of the human cerebral cortex is a task which is most often too big for a single group of researchers. It takes many experiments to reveal the functional contributions of the single part of the cerebral cortex and sometimes the results may still not converge to a common denominator. Another problem is that different research groups use different software for image analysis and different statistical software to evaluate the significance of the measurements. This leads to varying formats of the end-results. This inhomogeneity makes comparisons and detections inconsistencies and inconsistencies among research groups difficult and in some cases impossible. To obtain a broader picture of the vast number of different areas in the brain and their function, it is evident that access to a large number of experiments will be required, and that these data sets have to be analyzed by a common methodology.

2.1.2 Objectives
The NeuroGenerator project addresses the problems mentioned above by building a database centered analysis environment containing both libraries of analysis modules and a gradually growing homogeneous database of brain activation data from different experiments and different modalities. Creating a homogeneous database is a prerequisite for one main objective of the NeuroGenerator project, namely to provide an infrastructure, by which researchers can benefit from the advantages that can be obtained by analyzing multiple experiments in a joint manner. The NeuroGenerator project also has a number of neuroscientific objectives which should be facilitated by this infrastructure. Examples of such objectives are; to statistically test the consistency of results across research groups, and to statistically test the reproducibility of functional activations, both yet unresolved issues in the research field. It will also be possible to query whether a particular part of the brain is often co-activated or correlated with measures of activations elsewhere in the brain and whole functional networks can be revealed. We will, in section 2.2.2, return to more specific queries that NeuroGenerator must be able to handle.

2.1.3 System functionality and Usage
As the name implies, NeuroGenerator is a database generator, which enables dissemination of homogeneous database products to all EU research laboratories. The central system receives data (meta-data) describing conditions under which an experiment is done, together with the brain images resulting from the experiment in raw camera specific format. Submitting experiments to the NeuroGenerator is performed by a submission interface. This is a stand-alone application that is distributed to the different brain-research labs. The central system will act as the NeuroGenerator
factory and will contain the gradually growing database. The database system has, as the foundation, a large database containing the raw untreated data from PET and fMRI scanners together with meta-data about the experiments. The raw data are mathematically treated with image analysis software and statistical software in a single line of processing. This produces homogeneous data in a single statistical format and a single anatomical format. The central system will thus contain the raw untreated image-data, result image-data produced by the processing chain, and the experiment meta-data.

NeuroGenerator users are researchers, and their view of the system is as a homogeneous result database. Three different usage scenarios are specified in the technical annex [Technical Annex] to the agreement [Agreement] and can be described as:

A. Direct search and retrieval on the central server using a standard Java capable Web browser plus retrieval applets.

B. Extraction of a set of images and meta-data to the local file system for individual, database independent work, using a standard Web browser.

C. Download or receive a CD with a portable database product. Install and operate it on a local computer with no network required.

The portable database product consists of: the database management systems used in the central server acting as the factory, a NeuroGenerator specific user interface with query, analysis, and visualization tools and a user specified selection of the contents in the central database generator. Processing chains are not included in the portable database product.

2.2 Requirements on the Database System

The requirements presented in this section are of two different types. One type consists of the modeling and query requirements specified in the system document [Roland et al., 2002], and the agreement [Technical Annex]. The other type of requirements concern the system working in production. The production requirements are partly specified in the agreement and partly stated by the project managers: Per Roland, and Gert Svensson.

2.2.1 Modeling Requirements

The NeuroGenerator is primarily a database of brain images. The images are however only interesting if there exists a mapping between each image and a description. It is the connection between images and a rich description that makes the NeuroGenerator valuable. The data sets need to be modeled in the system so that relevant mappings are present.
2.2.2 Query Requirements

The meta-data and the image-data represents a non-homogeneous data set. The query language must therefore handle queries over the underlying data sources. Typical types of queries when using NeuroGenerator for research on the human brain are:

- Grouping queries
- Volume of interest (VOI) queries
- Correlation queries
- Consistency reproducibility queries
- Meta data queries
- Cluster thresholding

To understand the examples of such queries, we need to understand the characteristics of functional brain images. Processed functional brain images typically consist of a small number of clusters, distributed sparsely over a large image space. Processed functional brain images are also referred to as cluster images. A cluster image is an image of brain activation and can contain one or more statistically significant activation areas. A cluster image contains either zero or non-zero valued voxels (3D image elements), i.e. a statistically significant activation area is represented as a connected group of non-zero valued voxels. The value of the cluster images are exclusively related to the voxels within the activation areas, or more specifically the location and the volume of the areas. Voxels outside the activation areas are of no explicit interest for the purpose of the NeuroGenerator project. Some query examples from the query types listed above are:

- Grouping query, “Cluster population map”: Find a cluster image as the sum of all cluster images over some subset of the database.
- Volume of interest query, “VOI cluster search”: Given a user defined volume of interest, find all statistically significant activation areas that intersect this volume of interest.
- Correlation query: Given a set of functional experiments, compute which regions in space are simultaneously activated by these experiments. Can be extended with a degree measure.
- Consistency reproducibility query: Given a region in space, compute the regions in space that are co-activated with this region for a set of functional experiments. Can be extended with degree measure.
- Meta data query, “Keyword search 1”: Given a set of keywords from the keyword hierarchy, select all experiments that contain a given combination of keywords.
• Meta data query, “Keyword search 2”: Given a set of keywords from the keyword hierarchy, compute all statistically significant activation areas that are co-activated in experiments where these keywords are present.

• Cluster thresholding, “Cluster volume thresholding”: Given a set of statistically significant activation areas over some subset of the database, select only those areas that satisfy certain conditions on the volume of the cluster.

These queries require a query language like the Structured Query Language (SQL), the Object Query Language (OQL), or something equally worthy to operate on experiment meta-data. The language must also in some way be extended with functionality operating on images at voxel level. Two specific image primitives can be identified in the queries found in the system document [Roland et al., 2002], also evident in the above examples. One computes the degree of overlap between two different cluster images and the other computes an image as the sum/mean of a set of cluster images. Other image primitives and needs might be revealed when NeuroGenerator is used for research by its users.

2.2.3 Production Requirements on the Central system

Stability and Recovery

The back-end system, the database containing data for long term storage, is expected by the project managers to substantial up-time, therefore it is crucial that the system is stable with no primary memory leaks or other erroneous behavior that only can be remedied by rebooting the system. Recovery facilities are also expected, so that an operating system failure does not lead to data losses or inconsistencies in the database.

Large Data Sets

Experiments submitted to NeuroGenerator usually have the size of 300-500 MB for PET and 2-4 GB for fMRI. These data sets mostly consist of raw image-data and only a small fraction represents the meta-data about the experiments. It was decided within the project that the raw image data shall reside on disk media and not as BLOBS in the database, which only stores file references to the raw image-data. Stored directly in the database are the meta-data, and as BLOBS, the image-data resulting from the processing chains.

Since the meta-data and result image-data only represent about one percent of the original submitted data set, the real demand is on the disk-media storage capacity and not so much on the database system. Of course it is expected that the database system itself should not limit the amount of data possible to store in the future.
Security and Authentication
As the central system is located at Center for Parallel Computers (PDC), it must follow PDC’s security policy for user authentication, namely Kerberos [Kerberos] encrypted user logins. The system should also include security for the data stored in the database; external users should not be able to query or alter specified parts of the database. It is necessary to make certain parts of the database invisible to external users, because some information should be held confidential, e.g. non-published experiments, anatomical images with face intact, and such information that might reveal the identity of the subjects participating in the experiments.

Multiuser Environment
The database products chosen must support concurrency control to handle multiple users of the system. The concurrent users should be handled by separate threads that can access the data in parallel.

Multi-processor Support
The central database system is running on one or more computer nodes at PDC’s IBM SP super computer. Some computer node contain four parallel processors. It is desired to fully utilize this environment to get the best performance possible. That means that the database products in the system preferably shall be able to execute distributed on different computer-nodes. Or if executed on one computer-node at least utilize all four processors on that computer when handling concurrent users.

Support
Support is essential for the database system when it enters the production phase. The concept of support is not specified in detail within the project, it might be sufficient with documentation only. Still support must be available to an administrator for all products used in the system.

The production requirements on the central system have been identified during discussions with the project leaders. These requirements do not represent anything out of the ordinary, and are met by any larger DBMS vendor.

2.2.4 Production Requirements on the Portable NeuroGenerator Products
This section presents requirements that influence the choice of system solution more strongly. As mentioned previously in user scenario C, section 2.1.3, the portable NeuroGenerator products are clones of the central database generator, raw data and processing chains excluded. The dissemination of these products is one of the primary objectives within NeuroGenerator and it is surprising that the following
requirements are not better specified within the project. In the agreement only the following can be found:

Packaging of DBMS for HDP installation at user’s premises
Besides the data sets, the DBMS system has to be packageable for local processing at clients sites. Therefore, a service for easy DBMS packaging and installation at user site has to be prepared.

In the agreement “HDP” stands for Homogeneous Database Product. The actual requirements come from how the project managers interpret the agreement. Discussions with the project managers, Per Roland and Gert Svensson, have resulted in the requirements presented below.

**Easy Installation**

Data sets and DBMS products in the central system must be packageable for local processing at user sites. The chosen DBMS products in the central system must be possible to include into a service for easy unpacking and installation of the complete system at user premises.

**Multiple Platform Support**

The portable NeuroGenerator products must be operating on the platforms most commonly used by the research institutions across Europe. A survey performed within the project has shown that UNIX-like platform environments on Sun and PC hardware are the most commonly used. Examples of such are: Solaris on Sun, Linux and FreeBSD on PCs. Only UNIX on Sun computers is a requirement in the technical annex [Technical Annex] but the project hopes to attract more interest if multiple platforms are supported.

**Work With Limited Resources**

The portable NeuroGenerator products must operate on a work station. A limited resource to keep in mind is especially the size of primary working memory. Primary working memory can only be extended to a certain limit depending on number of memory slots on the work stations motherboard. Normal size for today’s work stations is however between 128 and 512 MB.

**License Issues**

For all software components within a portable NeuroGenerator product the license issue must be solved. The complete system must be allowed to be freely distributed for research usage.
Support

Installing and operating portable NeuroGenerator products requires at least the same support as administrating the central database server located at PDC.

2.3 Summary

Most of the requirements presented in this section are rudimentary for any database system. Only two requirement aspects really influence the choice of system solution. First, it must be possible to use conditions on images at voxel-level when querying the database. Second, the central system must be exportable to a stand-alone database product.
Chapter 3

The Solutions

From the previous chapter, we have a picture of NeuroGenerator and what is demanded of it. This chapter will present my choice of solution and the choice of the NeuroGenerator database expert group. The chapter will contain concepts and terminology from the database theory. Readers not familiar with these should visit appendix A.

3.1 My Choice

My choice is primarily focused on handling the requirement of making the NeuroGenerator system portable. That focus also fits well with the keep it simple approach, important to this master’s project. To get the simplest sufficient solution, I have chosen a single open source object-relational DBMS, see appendix A.1.4, called PostgreSQL [PostgreSQL].

3.1.1 Architecture

The architecture consist of the object-relational PostgreSQL DBMS [PostgreSQL] extended with NeuroGenerator specific User Defined Functions (UDFs) and files for storing the actual image-data, all depicted in Figure 3.1. Thus, the PostgreSQL DBMS contains experiment meta-data and paths to image files on disk. PostgreSQL uses the standard SQL language, well known to many users. But as it is an object-relational DBMS, the SQL language can be extended with UDFs. In PostgreSQL, a UDF can execute functions written in C/C++. In this case the UDFs make use of a library of functions developed at the Karolinska Institute. The library contains highly optimized image primitives implemented in the C programming language.

In PostgreSQL it is also possible to create any kind of user defined object classes/types, see appendix A.1.3. Defining an object type representing the image type used in NeuroGenerator would make it possible to store the images as BLOBS in the DBMS. Then all the information could be contained within the DBMS and no explicit usage of the file system would be necessary. However, having the images
available as files may have other advantages to the users of the portable NeuroGenerator products. For example, making it easier to perform database independent image analysis with their own tools.

3.1.2 Motivation of Choice

Both the Technical Annex [Technical Annex] and the project manager, Per Roland, explicitly required the system to be portable. Minimizing the number of components was therefore essential. The fewer components used in the system the easier it is to make it run properly on many platforms. Often fewer components and fewer layers means better performance.

PostgreSQL was chosen because it is an open source product and supports according to [PostgreSQL, Supported Platforms] over thirty different platforms. It provides fast functionality for exporting and importing the complete contents of the database. The DBMS is free for any users, which means that the project has no license costs at all. It is also very easy to install since all the source code is included. The installation process consists of the three steps known in the whole UNIX community: ./configure, make, and make install, which compiles and installs the DBMS optimally at any supported platform. The PostgreSQL DBMS is very well documented [PostgreSQL] and offers many recommendations for administration and performance enhancement.

An object-relational DBMS was necessary to handle the query requirements presented in section 2.2. Queries combining conditions on meta-data as well as on images at voxel level are available thanks to the UDFs.

Another reason for this solution is the query language. PostgreSQL supports the SQL92 standard. The SQL language is widespread and well documented, making it easy to NeuroGenerator users who want to learn to make ad hoc queries to the database.
<table>
<thead>
<tr>
<th>Component</th>
<th>Task</th>
<th>Man hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL</td>
<td>Installation, setup and evaluation</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Implementation of schema</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Submission interface connection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>to PostgreSQL with JDBC</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Definition of UDFs and C-library</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Transfer of workflow results</td>
<td></td>
</tr>
<tr>
<td></td>
<td>to PostgreSQL</td>
<td>40</td>
</tr>
<tr>
<td>Image primitives</td>
<td>Definition of image file format</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>for the images in PostgreSQL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Implementation of image file format and image</td>
<td>240</td>
</tr>
<tr>
<td></td>
<td>primitives</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3.2.** Presentation of man hours.

### 3.1.3 Development Steps

The work tasks involved with this solution were:

- Installation, setup, and initial performance evaluation.
- Implementation of the database schema, see appendix A.1.1, as SQL-definitions.
- Building a C-library with image primitives, i.e. compile and link the image primitives developed at Karolinska Institute into a shared dynamic library. A library from which PostgreSQL loads functions used in the UDFs dynamically at runtime.
- PostgreSQL definition of the UDFs supporting the query types presented in section 2.2.2.

### 3.1.4 Resources Spent

My solution did not bring any license costs or any other investments, the only resources spent were the man hours used for implementing the system. The man hours are compiled in Figure 3.2
3.1.5 Appraisal of the Solution

It is Gert Svensson’s, Per Roland’s, and my opinion that this solution meets all the requirements discussed in section 2.2. It handles the query and modeling requirements and fits well to both the central system and to the portable NeuroGenerator products. The query performance using the UDFs was also unexpectedly good as will be shown in chapter 4.

There is one obvious limitation with this solution. The UDFs are very specific; every time a new UDF is needed it propagates to the C-library where a new image primitive has to be developed. In the prototype developed in this master’s project, only two UDFs were integrated into PostgreSQL, one computing the overlap, and one computing an image as the mean of a set of images. There were however about twenty more primitives available at Karolinska Institute, but they were not necessary to meet the requirements in the specification [Roland et al., 2002]. They can easily be integrated into PostgreSQL if they are of interest.

3.2 The NeuroGenerator Choice

This choice of solution was primarily based on the modeling and query requirements presented in section 2.2. To get the most general solution, the database group choose an object-oriented multidatabase system, see appendix A.1.5.

3.2.1 Architecture

Due to the inhomogeneous data sets represented by the image-data and experiment meta-data, the solution separates these data sets into two different DBMSs. One relational, see appendix A.1.2, DBMS containing the descriptive meta-data and one raster DBMS, RasDaMan [RasDaMan], containing the multidimensional image-data. To seamlessly integrate these two DBMSs, a third object-oriented DBMS, see appendix A.1.3, was introduced. This third DBMS, Amos II [AMOS II], is the central part of the system as it represents the global data model, provides the global access language (AMOSQL), and acts as the user interface to the multidatabase system. AMOS II is thus responsible for the query processing shown in Figure A.3. In NeuroGenerator, Amos II was referred to as the mediator server. In RasDaMan the result images are represented at a conceptual level and a general image query language called RasQL is provided. RasDaMan actually uses yet another relational DBMS as a back-end for storing the actual voxel values of the images. The raw unprocessed image data and intermediate results were stored as files on the file system, only file references were stored in the database.

The solution thus introduces four different components that possess database functionality, using mediator technology that provides transparency to upper layers. This architecture is depicted in Figure 3.3.
3.2.2 Motivation of Choice

As the database group in the NeuroGenerator project wanted the most flexible solution, the multidatabase architecture was the obvious choice. They stated that it enables the development of an application programming interface that is independent of the underlying image and meta-data storages, i.e. making it possible to change the underlying DBMSs without propagation to user applications.

The group stated the necessity of using RasDaMan. RasDaMan should be used as it provides fast access to raster data, e.g. images, and has a rich set of image primitives. The NeuroGenerator would have a general tool for handling all types of queries over images. Since it is not possible to store any descriptive meta-data mapped to the images in RasDaMan, the multidatabase solution was also a necessity. Then AMOS II would provide the mapping between the meta-data in the relational DBMS and the images in RasDaMan.

The third argument was that an object-oriented multidatabase also makes it possible to create a domain-dependent high level query language, see appendix A.1.3, accessing both image and meta-data. The high level queries could be expressed using neuroscientific concepts by defining object types/classes and methods using neuroscience terminology in the global schema in the mediator server.

3.2.3 System Components

In this section the numbered components in Figure 3.1 are presented together with the work tasks they involve.
Component 1 - The AMOS II DBMS

The AMOS II mediator DBMS is developed as a research tool by a research group at Uppsala University supervised by Professor Tore Risch. The research group is also a partner in the NeuroGenerator project. The object-oriented AMOS II is a very light-weight DBMS and supports an idea that Risch in [Risch, 2003] describes as:

The wrapper-mediator approach divides the functionality of a data integration system into two kinds of subsystems. The wrappers provide access to the data in the data sources using a common data model (CDM), and a common query language. The mediators provide coherent views of the data in the repositories by performing semantic reconciliation of the CDM data representation provided by the wrappers.

The AMOS II is extensible with user defined functions which can execute local/global translation software, by Risch referred to as wrappers, written in Java or C. It also permits high-level abstractions and queries. The DBMS keeps the complete contents of the database in the primary working memory to provide high computational performance.

In order to integrate data sources with different representation formats, it is necessary to translate the data from each data source into the data abstraction of the AMOS II data model. This leads to the development of one wrapper for each data source. The Amos II must also be extended with heterogeneous query optimization software for good performance of queries over the underlying data sources. Tasks involved with AMOS II were:

- Initial work such as installation, setup, studies of manuals, and studies of AMOSQL specification.

- Implementation of the global schema, see appendix A.1.1, A.1.3.

- Implementation of insert-methods used by the submission interface when it connects to the database for population of Experiments. Calling predefined methods encapsulating database manipulations are up to a hundred times faster than sending the commands as text strings to the AMOS II. Population time for an experiment using inserts-commands as text strings was between five and fifteen minutes, depending on experiment size. The insert-methods reduced the population time to about 20 to 90 seconds.

- Implementation of ASCII format off-load function. As AMOS II works in primary memory, every thing in the database is gone if the DBMS is shutdown. This function made it possible to dump the contents of the database as AMOSQL commands to a text file. Before it was only possible to make a memory dump of the database when a backup was wanted. The memory dump backup facility was not suitable for the export of the database contents since the dumps were dependent on platform and AMOS II version.
Component 2 - The RasDaMan DBMS

RasDaMan is a domain-independent DBMS which supports multidimensional arrays of any size and dimension, and over freely definable cell types. Its purpose is modeling and storage of sampled natural phenomena. Brain activation becomes spatio-temporal array data of three dimensions once it is sampled by a PET or fMRI camera and quantized for storage and manipulation in a computer system. Thus, data in a representation well suited for RasDaMan.

In RasDaMan everything is stored in arrays. Each array element, referred to as cell, is positioned in space through its coordinates. A cell can contain a single value, e.g. an intensity value in case of grey-scale images, or a composite value, e.g. the integer triples for the red, green, and blue components of a color image. All cells share the same structure which is referred to as the array cell type. The database developer defines both the spatial extent, i.e. a lower and upper bound in each dimension, and cell type in the array type definition.

The arrays are grouped into collections. All elements in a collection share the same array type definition. Collections form the basis for array handling, just as tables do in relational database technology. The RasDaMan query language includes many matrix operators and allow queries to work on subparts of images. Working with subparts of images reduces the response time when searching the database for brain activation in a certain area. Work involved with RasDaMan was:

- Installation and setup. The decided platform for the central NeuroGenerator system was AIX Unix on PDC’s IBM SP super-computer called Strindberg. The RasDaMan DBMS was not ported to this platform. After much effort it was still not possible to link the object files resulting from compiling the source code. Instead RasDaMan was setup on a PC using Linux. An other complicating issue was the RasDaMan license scheme which demanded online contact with the license server in Germany at all time, otherwise RasDaMan would block. The license was linked to a computer with a certain Internet address. Changing the location of the DBMS required a new license.

- Design and implementation of local RasDaMan schema, i.e. the array type definition and collection definitions.

Component 3 - A Relational DBMS for the Meta-Data

To store the meta-data about the experiments the multidatabase solution required a generic relational DBMS. Between the commercial DB2 and the non-commercial FireBird, FireBird were the strongest candidate within the NeuroGenerator project. Tasks involved were:

- Installation, setup, and administration.

- Implementation of schema, i.e. translation of the meta-data parts of global schema defined in AMOSQL to SQL. This multidatabase solution was quite
contrary to the typical multidatabase described in appendix A.1.5, where the
global schema has no influence over the schemas in the underlying DBMSs.

Component 4 - A RasDaMan Compatible Relational DBMS

The RasDaMan DBMS needs a relational DBMS for the actual storage of the mul-
tidimensional data. Internally and invisible to the RasDaMan user, arrays are de-
composed into smaller units, called tiles, which are maintained by this DBMS. Only
ORACLE and DB2 DBMSs are supported by RasDaMan. An ORACLE license was
too expensive which left the choice of DB2. Work involved with this component was:

• Installation and setup on AIX. This work was not all easy, many interactive
choices had to be made during installation which required much reading in the
extensive DB2 documentation. Many times the error messages produced by
the DBMS were difficult to interpret. For example, much effort was put on
one that demanded the support from IBM for translation. It revealed that the
DBMS could not be installed in a directory located on the network file system,
it had to be installed on a local disk.

• When RasDaMan was not possible to install on AIX, this DB2 also needed
re-installation and setup on a Linux station. The work went smoothly with
all the practice on AIX.

Components 5 and 6 - Wrappers

The wrappers are what actually glues the multidatabase together. They offer query
translation mechanisms for transparent access to mediated data from the AMOS II
object-oriented query language. The development work was non-trivial and included
intermediate schema meta-class constructs, specification of mapping rules, syntax
analysis with tree-parsing techniques, and many intricate transformation steps using
various layers. The details are not interesting in the context of this thesis. Compared
with the multidatabase described in appendix A.1.5, this work was easier when the
developers had control of the complete system. The wrappers were implemented in
the Java programming language and for all communication between the DBMSs the
JDBC-API was used.

None of the wrappers were existing before the NeuroGenerator project. When
the extent of the development work became apparent, it was decided to strongly
limit the functionality in the wrapper between AMOS II and RasDaMan. Ad hoc
queries to RasDaMan should not be supported, only a set of predefined image queries
and update commands should be executable through the wrapper.

3.2.4 Resources Spent

This section presents an estimation of the resources spent in the development of the
solution. Many values are rough estimates resulting from interviews with project
<table>
<thead>
<tr>
<th>Component</th>
<th>Task</th>
<th>Man hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. AMOS II</td>
<td>Installation, setup and testing</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>Implementation of schema</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>Implementation of insert functions</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Implementation of ASCII off-load function</td>
<td>160</td>
</tr>
<tr>
<td>2, 4. RasDaMan, DB2</td>
<td>Installation, setup and debugging</td>
<td>480</td>
</tr>
<tr>
<td></td>
<td>Implementation of schema</td>
<td>40</td>
</tr>
<tr>
<td>3. FireBird</td>
<td>Installation and setup</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Translation of schema, AMOSQL to SQL</td>
<td>80</td>
</tr>
<tr>
<td>5. RasDaMan-Wrapper</td>
<td>Overall development</td>
<td>1280</td>
</tr>
<tr>
<td>6. FireBird-Wrapper</td>
<td>Overall development</td>
<td>960</td>
</tr>
</tbody>
</table>

Figure 3.4. Presentation of man hours.

participants. Interviews were necessary when the project progress report [Progress report] did not provide high enough resolution and the cost statements were unaccessible. Still, values are in the correct range and should give a valid picture.

Each component in the multidatabase solution did in some way consume resources, the dominant part was the man hours spent on the implementation work. These are compiled in Figure 3.4.

Software Licenses

This solution includes two components that do not have cost free licenses, RasDaMan and DB2. The project used an evaluation license for DB2 during the development work which did not bring any costs. IBM also provides after individual application, a cost free DB2 license for research usage. RasDaMan brought an initial cost of 20 000 Euros for a license permitting the usage in the central system. An additional 5000 Euros was also to be paid for each portable NeuroGenerator product disseminated to the European research labs.
3.2.5 Appraisal of the Solution

The query requirements and the modeling requirements are met by this solution. Descriptions can be mapped to the individual images and a large set of image primitives is available for queries. The production aspects and the portability-aspect are however neglected.

The solution used in the central system has many problems due to the fact that AMOS II is a research tool and not a commercial DBMS product (or well-tested open source product). Certain limitations worth to point out are:

1. The DBMS uses a non-standard query language. The language lacks nested selects statements in queries and many aggregate functions.

2. The DBMS is not well documented. Only two web pages are available [AMOS II Concepts], [Amos II User’s Manual].

3. The AMOS II does not follow the JDBC standard for communication with user applications implemented in Java. To communicate, the user application must execute an embedded AMOS II within it self, just for sake of protocol.

4. The DBMS has primitive backup facilities, the contents of the database can only be stored as a dump of the primary memory. The dumps are not compatible between different platforms and different AMOS II versions. Late in the project an ASCII-text dump facility was added.

5. The DBMS resides completely in primary working memory, which limits the storage capacity.

6. The AMOS II lacks security for both data and user authentication. It is not possible to set permissions on object classes/types or attributes, the complete database is public. The database is also open to the Internet for communication with the LISP programming language. Unauthorized users with LISP-knowledge may influence both data and methods in the database.

7. The DBMS is not multi threaded, concurrent users block each other which prolongs the response time.

To handle the points 3-7, Tore Risch, a member of the database group, suggested a new architecture, see Figure 3.5. Nothing should be stored permanently in AMOS II. A new AMOS II containing just an empty schema should be started for each user connecting to the central server, AMOS II should only be used as a cache. In this way AMOS II is relieved from the responsibilities such as: security, multi-user support, storage capacity, and backup etc. and relies instead on the underlying DBMSs to handle these tasks. This new solution not only further complicates the system administration and the development work. It also puts an upper bound on the number of concurrent users to about 250. This is due to that each AMOS II is using at least 7 MB primary memory in the central server computer. Another
question is, who or what is starting a new AMOS II for each connecting user? Yet another layer seems to be necessary to handle the initial client contact.

With the new architecture the modeling requirements, the query requirements, and the production requirements on the central system in chapter 2.2 are met. However, the production requirements on the portable NeuroGenerator products were still foreseen by the database group. Not only was it harder to create a portable NeuroGenerator product supporting multiple platforms when the utilized DBMSs are not open source products. It was not even possible to create a product for off-line usage. First, the RasDaMan license only permits usage of the DBMS as long it is connected to the RasDaMan license server in Germany. Second, the DB2 license does not permit the DBMS to be included into the NeuroGenerator product which should be distributed cost free. Third, both DB2 and RasDaMan demands interactive inputs regarding the platform during the installation process, prohibiting them from being included into a service for easy installation. Other issues with the solution are porting all the components to various platforms, and export/import of the complete contents of the system, schemas as well as the data. Issues, that are hard to solve in practice.

When the functionality in the wrapper between AMOS II and RasDaMan was restricted, this solution has the same limitation as mine. Each time a new image primitive needs to be added to the AMOSQL, an update of the wrapper is needed. That is of course much less work than what is necessary in my solution. Adding a
new RasQL query to the wrapper is much easier than adding a new image primitive function to the library.

In this appraisal, it is also important to mention that at the time of writing this thesis, February 2003, this solution has not proved its concept. The components in the multidatabase have never been tested together. No global query, depicted in Figure A.3, simultaneously accessing both underlying data sources has been working.

3.3 Integrated Control of Image Processing Chains

The processing chains consist of a set of image processing modules which are applied to the raw image-data and finally produces the statistical result images. A control system, referred to as the workflow. It sets up the in- and out-data to each module and executes the modules in a specific order. The workflow is thus a queue manager which uses the information in the database.

The database group decided that the most elegant solution would be to integrate the queue manager, the workflow, into the AMOS II. Thus, implement the complete workflow in AMOSQL and execute it within the DBMS. Then the workflow would gain direct access to all information and also traceability for the image processing would be provided. AMOS II’s active rules should be used to keep track of the processing chain events, e.g. the starts and terminations of the modules. In that way the multidatabase would contain the submitted data, the result data, and detailed information of the steps leading from one state to an other.

Many aspects with this idea were very attractive, others were not. Now the NeuroGenerator became extremely dependent on AMOS II. In practice AMOS II did not live up to the expectations. The active rules, often called triggers in other DBMSs, were not active, they were passive and had to be polled indefinitely in a loop. This work-around, i.e. to explicitly check the rules, significantly reduced the query performance. Another big problem was that AMOS II displayed severe internal memory leaks during workflow execution due to bugs in the logging facilities. The database image could grow up to 150 MB during the processing of one single fMRI experiment, without inserting any new information.

This led to that the workflow still using AMOS II had to be used as a stand-alone application. Results had to be transferred to the central database. With my solution the workflow is used in the same manner.

3.4 Summary

This chapter has presented two alternative system solutions for the NeuroGenerator. Mine, consisting of the single object-relational DBMS PostgreSQL, and the database group’s object-oriented multidatabase. My solution is simple and yet fulfills the requirements in the agreement. The other solution was more complicated and did not meet a primary aspect of the agreement. The NeuroGenerator system has to be portable and the multidatabase solution was not suited to handle this requirement.
The multidatabase solution could in theory provide a wider functionality than the PostgreSQL solution. In practice, it had to be reduced which lead to the two solutions ending up with almost the same functionality.
Chapter 4

Results

This chapter presents results from performance tests and compares other aspects of the two alternatives. Not only the results from comparisons are presented, also the impact of this master’s project on the NeuroGenerator project is discussed.

4.1 Master’s Project Impact

In the initial work of this master’s project I searched for a suitable DBMS. A small performance test with PostgreSQL was done. A few tables and an UDF were defined. The UDF used a highly optimized overlap-function using technique originating from the Commodore 64 era. Forty cluster images and some dummy meta-data were inserted. I formulated a query which combined all images against each other, computed the overlap for each combination, and put some conditions on the meta-data.

The result was very good, the computation of 1600 overlaps was done in less than ten seconds. Earlier test with RasDaMan had compared about 30 images in ten seconds. Of course I had cheated a bit. I had pre-converted the images from using 16 bits per voxel to one, i.e. to a binary format. It can be done with no significant information loss in the cluster images, see section 2.2.2. The function I used was also specialized for this format. It, however, proved the concept and a full scale investigation of the performance and the good of RasDaMan was initiated. The investigation made use of my prototype in the performance test. The investigation report [Naeslund and Selander, 2002] concluded that the good of RasDaMan could not compensate for the relatively poor performance and high cost it brought to the NeuroGenerator system. The Figure 4.1 origin from that report displays the performance in the two DBMSs. As the result RasDaMan was dropped and the UDFs developed for my solution was to be used in AMOS II.

4.2 Performance

The PostgreSQL solution also incorporates a representation of the brain activation images, which within the project was referred to as the *collage representation*. In
Figure 4.1. Visualization of performance test between RasDaMan and PostgreSQL. Figure origins from the RasDaMan Evaluation report [Naeslund and Selander, 2002].

the RasDaMan evaluation report [Naeslund and Selander, 2002], Mikael Naeslund describes the collage representation as follows:

Given the characteristics of the functional brain images, it is intuitive to represent the images logically as collages. The basic idea of such a collage representation is that only the voxels within identified clusters are explicitly stored as voxel values connected to coordinates. Furthermore, the voxels within clusters are stored collectively for each cluster in binary objects that are directly and individually accessible. For each binary object in the collage, meta data such as bounding box, voxel count, volume, variance etc. are stored.

With this technique, overlap search was even faster than what is shown in Figure 4.1. Bounding box values can be used to narrow the search set before any actual voxels are compared.

There is at the time of writing, February 2003, no way to directly compare the query performance between the two solutions. The multidatabase has never been able to handle a global query with the use of RasDaMan and the UDFs developed for the PostgreSQL solution have still not been incorporated. A comparison regarding image queries is no longer very interesting since AMOS II was going to use the same image primitives as PostgreSQL. A qualified guess is that the PostgreSQL solution still is significantly faster since the other solution uses two different DBMSs; AMOS
II, FireBird, and a wrapper between them implemented in Java, a language not
known for speed.

It is still possible to compare query and update performance regarding the meta-
data parts of the database. Tests can be performed using AMOS II as a primary
memory database with no usage of the wrapper and the FireBird DBMS. Population
of the same set of experiments into the two different DBMSs shows that insertions are
about fifty percent faster in the PostgreSQL DBMS, which had no predefined insert
methods and was writing everything to disk quite contrary to the AMOS II. Query
response time was tested by having the same set of experiments in the databases
and by extracting similar information from each of them. Again, PostgreSQL was
on average thirty percent faster than AMOS II.

When the task specific test showed that PostgreSQL individually outperformed
both AMOS II and RasDaMan, my conclusion is that the multidatabase solution
would have been much slower than the PostgreSQL solution.

4.3 Other Aspects

Comparisons can be made of other aspects. Development costs for the two
solutions differ. It is apparent from the Figures 3.2 and 3.4 that the multidatabase
solution consumed 2800 more man hours than the PostgreSQL solution for the actual
implementation. The many meeting hours spent on the multidatabase solution,
discussing the design, problems, work-arounds, etc. are however not visible in the
comparison. The multidatabase alternative also included a 20 000 Euro license cost.

Administrating the solutions in the production state can also be expected to
differ. The multidatabase must at least be distributed over two computer sites using
different platforms, RasDaMan and DB2 located on a Linux PC-station and AMOS
II and FireBird on a IBM AIX machine. The multidatabase administration therefore
require more resources for hardware, platform support, license handling, etc. Also
more debug-reporting and trouble-shooting can be expected as more components
involved usually gives a bigger risk that an error will occur. Substantially more
documentation is also needed to be produced for the multidatabase alternative.

Another aspect is dependencies, both presented alternatives leads to that the
NeuroGenerator system becomes dependent on a specific DBMS. The multidatabase
depend largely on the AMOS II and the other on PostgreSQL. This means that the
system-developers have to implement some software components very specific for a
certain DBMS. For the PostgreSQL solution, the case is not so bad. PostgreSQL
supports the SQL92 language standard and the ODBC, and JDBC standards for
communication with external applications. The syntax of UDF-definitions is how-
ever specific for the PostgreSQL DBMS. For the multidatabase, the case is worse.
The AMOS II has its own language and does not support the ODBC and JDBC
standards. The DBMS has also a specific syntax for UDF-definitions. Replacing
AMOS II also leads to large updates in the wrappers. Thus, a DBMS migration is
therefore more complicated with the multidatabase alternative.
The primary aspect is however how well suited the solutions are for the Neuro-Generator application. As mentioned earlier, NeuroGenerator is a database generator. The agreement [Technical Annex] states that “The HDPs are portable and will disseminated to all EU members” (HDP - Homogeneous Database Product). That is one of the primary objectives with the project. From the chapter 3, we know that the multidatabase is not suited for this requirement. Therefore the PostgreSQL solution also wins in this aspect.

4.4 Summary

In this chapter results from performance test and comparisons of other aspects have shown that the simpler solution is the best alternative. With the PostgreSQL solution overlap-search is over 50 times faster, development costs have only been a fraction of the costs for the multidatabase alternative. The PostgreSQL DBMS is known to work well in production and does not bring any license costs. Features such as open source, very good documentation, and fast dump and restore facilities also make the PostgreSQL DBMS well suited for the portable database product.
Chapter 5

Conclusions

5.1 My Conclusion about the NeuroGenerator Project

The multidatabase solution has without doubt complicated the development of NeuroGenerator. It has seriously delayed other work tasks in the project which where dependent on the database. It is apparent in the agreement that large specified work packages are delayed over a year and half, with overall inefficiency and resource waste for the project as the result.

It is unfortunate that an evaluation like this master’s project was not performed in the beginning of the project. To initially evaluate the simplest possible alternative would not have cost much and would in this case have saved large resources.

5.2 My Conclusion

My conclusion is that poor specifications and non-pragmatic technicians is a dangerous combination. The higher technical skills the involved technicians possess, the higher is often their strive for general solutions. This behavior together with loose specifications imposes a risk, because there is a big chance that something too complicated is started that will not be finished in time.

It happens again and again, and my opinion is that the technicians are the responsible ones. I had the same experience in a software development project-course at KTH, were we chose a very general solution, worked ourself to death, ended up with seriously restricting the solution in order to handle the deadline. Almost at the end of this master’s project, I read an article [Jensfelt, 2003] about a software project in the construction business. Five years and 113 million Swedish Kronor was spent on developing a general document platform for construction projects. The system was again too complicated to implement and the project resulted in a CD with documentation.

One thing is obvious; we, the technicians appear ridiculous and get a bad reputation if we CANNOT FINISH WHAT WE START.
5.3 End of the Multidatabase

In the end of December 2002, Per Roland and Gert Svensson agreed that the PostgreSQL solution had proved itself. With the PostgreSQL solution the project had a chance to reach its goals. The AMOS II was dropped and the PostgreSQL solution was chosen as the main track for the project.

The NeuroGenerator system will have its first web-release, URL: http://www.neurogenerator.org, in the end of March 2003. Then the PostgreSQL solution will incorporate about thirty image primitives which can be utilized through the user interface.
Bibliography


Appendix A

Fundamental Database Theory

A.1 Database System Concepts and Architectures

A database system solution can make use of different types of DataBase Management Systems (DBMSs) or hierarchies of different DBMSs. Several criteria can be used to classify DBMSs and to understand these criteria it is necessary to introduce fundamental concepts in database theory. Primary criteria in [Elmasri and Navathe, 1999] are the data model, the number of users, and the number of sites over which the database is distributed. This appendix explains the basic concepts and the difference between the data models and architectures used in this thesis.

A.1.1 Basics concepts

A Database

“A database is an integrated collection of persistent data, representing the information of interest for the various programs that compose the computerized information system of an organization” definition proposed in [Atzeni et al., 1993]. Alternatively in [Elmasri and Navathe, 1999]: “A database is a collection of data”.

In a database, data are separated from the programs that use them: data are described autonomously, and programs make use of these descriptions. Different programs may access and modify the same database and share common data. Inconsistencies and redundancies among the representations of the same data in different programs are thus reduced.

A Database Management System

A database management system (DBMS) is a collection of programs that enables users to create and maintain a database, [Elmasri and Navathe, 1999]. The DBMS is hence a general-purpose software system that facilitates the processes of defining, constructing and manipulating databases for various applications. It provides common and controlled means for accessing and modifying data, along with integrated services which should support security, integrity, and reliability. A fundamental
aspect of a DBMS is the data model used for the high-level description of data. This data model should be as independent as possible of the physical representation of data. The independence guarantees that modifications made to the physical representation do not affect the high-level description and, as a consequence, the programs using the DBMS.

**A Database Schema and State**

In any DBMS it is important to distinguish between the description of the database and the database itself. The description of a database is called a database schema, which is specified during database design and is not expected to change frequently.

The actual data in a database may however change frequently. The data in the database at a particular moment in time is called a database state. Every time a record is inserted or deleted, or the value of a data item in a record is changed, the database changes its state.

The distinction between database schema and database state is very important. When a new database is defined, only the database schema is specified to the DBMS. At that point, the corresponding database state is the empty state.

**DBMS Languages**

There are mainly two types of languages a database designer/user will make use of in a DBMS. One is the data definition language (DDL) which is used for constructing the conceptual schema definitions, as well as the internal schemas for the database and any mappings between the two levels. Second, the data manipulation language (DML) for retrieval, insertion, deletion, and modification of the data. In current DBMSs the two types of languages are usually not considered distinct, rather, a comprehensive integrated language is used, e.g. SQL, OQL.

**A.1.2 The Relational Data Model**

The relational data model has been the most commonly used over the last twenty or so years. The model uses the concept of a mathematical relation as the single structure to organize data, which somewhat looks like a table of values. A table row represents the basic *information element* in the relational data model, each row in a table represents a collection of related data values.

The data model has its theoretical basis in set theory and first order predicate logic. Major reasons for its popularity are the high degree of logical independence and that the model successfully couples a precise mathematical definition with a useful representation based on tables. Both an informal and a formal definition of a relational database is provided in [Atzeni et al., 1993], the informal reads as the following:

A relational database is represented as a collection of tables. Each table is assigned a unique name in the database. A row in a table represents
Experiment | Exp_id | Study_name | Name_of_lab              | Date          
---|--------|------------|--------------------------|---------------
    | 7      | Tactile match | Karolinska Institute    | 20-Feb-2001   
    | 3      | tD3D       | Karolinska Institute    | 6-Jun-2002    

Subject | Subject_id | Age | Sex | Exp_id |
---|------------|-----|-----|--------
    | KI4711     | 23  | male| 7      |
    | 2192       | 41  | female| 3    |
    | KI4712     | 43  | female| 7    |
    | KI4713     | 34  | male| 7      |
    | 2191       | 37  | male| 3      |
    | 2193       | 31  | male| 3      |
    | KI4711     | 23  | male| 3      |

**Figure A.1.** The tabular representation of a database

a relationship among sets of values. Column headings contain distinct names, and for each column there is a set of possible values, called the domain.

Also the main characteristics of a relation represented by means of a table are stated in [Atzeni et al., 1993]. In each table:

- The values of each column are homogeneous: the values of an attribute belong to the same domain (integers, strings, etc.)
- The rows are different with respect to one another: a relation is a set, and therefore it never contains identical tuples.
- The order of the columns is irrelevant, since they are always identified by name and not by position.
- The order of the rows is irrelevant, since they are identified by content and not by position.

The first item represents the domain constraint in a relational database. This restricts each attribute A in a table to be an atomic value from the domain dom(A). Item number two expresses the key constraint, that no two rows can have the same combination of values for all their attributes. Usually, there are many sub-sets of the attributes in a relation with the property that no two rows should have the same combination of values for these attributes. A sub-set with this property which is also minimal is called a key. There might be more than one key in a relation and one have to be chosen as the primary key.

Figure A.1 shows the tabular representation of some information about brain experiments and the participating subjects. Each row in the Experiment-table represents one unique brain experiment and each row in the Subject-table, a unique participating subject. The occurrence of the attribute “Exp_id” in both tables makes the tables joinable. That is, zero or more Subject-table rows are related to
one Experiment-table row. A key in Experiment-table could consist of the attribute “Exp_id”, and in the Subject-table the key could consist of the “Subject_id” together with the “Exp_id” attribute. The “Exp_id” attribute needs to be included in order to represent the one-to-many relationship. In the relational database theory an attribute like the “Exp_id” in the Subject-table is called a foreign key. Worth noticing, is that the model in this example with the proposed keys, does not allow storage of subjects not participating in any experiment.

In a relational DBMS the database schema, i.e. the description of the database, consists of the tables and the constraints (domain and referential) defined by the database designers.

### A.1.3 The Object-Oriented Data Model

The relational data model has been quite successful in developing the database technology required for many traditional business database applications. However it has some shortcomings when developing more complex database applications— for example, databases for scientific experiments and multimedia databases (images, audio, video, etc.). These newer applications have requirements for more complex data types and query formulations than what is available in traditional database systems. DBMS based on the object-oriented data model have been proposed to meet these needs, when their key feature is the power they give the designer to specify both the structure of complex objects and the operations that can be applied to these objects.

The object-oriented data model has its origin in object oriented programming languages, which themselves have their roots in the SIMULA language proposed in the late 1960s. Today, the model is also applied in the areas of databases. The object-oriented data model lacks a comprehensive mathematical foundation. Instead the model is based on the concept of a computational entity, called an object, which encapsulates an information element as well as the operations acting on it. We say that an object typically has two components: state, its encapsulated values and behavior, and the specified operations also commonly called methods. An object is a real system entity stored in memory or on disk.

A large number of differences exists in current object-oriented data models. Despite this and the lack of formalism, Bukhres and Elmagarmid [Bukhres and Elmagarmid, 1995] states that there exists certain accepted concepts concerning these models which can be grouped into a core model. Their presentation of the core model concepts include the following:

- **Objects and identity:** Each real-world entity is modeled by an object. Each object is associated with a unique identifier that makes the object distinguishable from other objects. In particular, object-oriented DBMSs provide objects with persistent and immutable identifiers. An objects identifier does not change even if the object modifies its status.
• Complex objects: Each object has a set of instance attributes; the value of an attribute can be an object or a set of objects. The set of attributes of an object represents the object status.

• Encapsulation: An object status is accessed or modified by sending messages to the object to invoke the corresponding methods. The set of messages that can be sent to an object is the object interface.

• Classes: Objects sharing the same structure and behavior are grouped into classes. A class represents a template for a set of similar objects. Each object is an instance of some class.

• Inheritance: A class can be defined as a specialization of one or more classes. A class defined as specialization is called a subclass and inherits attributes, messages, and methods from its superclass(es).

The first two items makes one of the main goal with object-oriented databases possible, namely to maintain a direct correspondence between real-world and database objects. This is possible because of the unique object identifier (OID) and an object structure of arbitrary complexity, which enables an object to contain all of the necessary information that describes the real-world object. In contrast, in relational database systems, information about a complex object is often scattered over many relations, leading to loss of the direct correspondence.

Other useful concepts in the object-oriented data model are inheritance and operator polymorphism. Inheritance permits specification of new classes that inherit much of their structure and operations from previously defined classes. Specification of object classes can proceed systematically. This makes it easier to develop the data types of a system incrementally, and reuse existing class definitions when creating new classes of objects. Operator (method) polymorphism refers to an method’s ability to be applied to different types of objects; in such a situation, a method name may refer to several distinct implementations, depending on the type of objects it is applied to. This feature is also called operator (method) overloading. For example, a method to calculate the area of a geometric object may differ in its implementation, depending on whether the object is of type triangle, circle, or rectangle. Object-oriented DBMSs with query languages supporting method overloading have so called domain dependent query languages. This means that the results from two database queries using the same set of methods but applied to different object types will most certainly differ. Such a feature does not exist in traditional relational DBMSs.

One important issue in object-oriented DBMSs is how to represent relationships among objects. Many alternatives exists but the most practical solution is chosen in the ODMG 2.0 standard, which explicitly represents binary relationships via a pair of inverse references—that is, by placing the OIDs of related objects within the objects themselves, and maintaining the referential integrity.

In an object-oriented DBMS the database schema consists of the class definitions and the definition of the methods that can be applied to each class of objects.
A.1.4 The Object-Relational Data Model

The relational model have proved to work well and its popularity is also helped by a very robust infrastructure in terms of the commercial DBMSs that have been designed to support it. The object-relational data model is a trend that combine the best features of the object data model and languages into the relational data model so that it can be extended to deal with the challenging applications of today. Common features that the relational DBMSs are enhanced with are:

- Support for additional or extensible data types.
- Support for user-defined routines (functions).
- Implicit notation of inheritance.
- Application programming interface for new types and functions.

An explanation of why object-relational DBMSs have emerged is given in [Elmasri and Navathe, 1999]. They classifies DBMS applications according to two dimensions or axes: (1) complexity of data along one axes, and (2) complexity of querying along the other. The classification scheme is depicted in Figure A.2. Traditional relational DBMSs belongs to the upper left quadrant. Although they support complex ad hoc queries and updates (as well as transaction processing), they can deal only with simple data that can be modeled as a set of rows in a table. Many object databases
fall in the lower right quadrant, since they concentrate on managing complex data but have somewhat limited querying capabilities. In order to move into the upper right quadrant to support both complex data and querying, relational DBMSs have been incorporating more complex data objects while object-oriented DBMSs have been incorporating more complex querying.

In an object-relational DBMS the database schema consists of the table-definitions, the class-definitions of complex data types, and the definitions of any user-defined methods that can be applied to the data types in the database.

### A.1.5 The Multidatabase Architecture

The relational, object-oriented, and object-relational DBMSs discussed above are most often so called centralized DBMSs or in some cases homogeneous distributed DBMSs. A DBMS is centralized if the data is stored at a single computer site. A centralized DBMS can support multiple users, but the DBMS and the database themselves reside totally at a single computer site. A homogeneous distributed DBMS can have the actual database and DBMS software distributed over many sites, connected by a computer network. In homogeneous distributed DBMS only one type of data model is used and it is the same DBMS software that is executed on the multiple sites.

A recent trend is to develop software to provide a global interface to several autonomous preexisting databases stored under heterogeneous DBMSs. This leads to a federated DBMS, also called a multidatabase system. Users can access multiple remote databases with a single query. The multidatabase software automatically performs the data model and access language transformations between the global query and the local databases. Rather than provide transformations between every combination of local data models and access language, multidatabases define a global data model and a global access language. The participating DBMSs are loosely coupled and have a degree of local autonomy. They may also use different data models. However, the object data model is currently the most popular choice for the global data model in multidatabases. The object-oriented paradigm is particularly adept at capturing the semantics of other data models and representing it in ways that are easy to integrate. Also, the global interface of a multidatabase must present large amounts of varying information in a form that makes the information easy to access and manipulate. The abstraction capability of object-oriented systems and the ability to apply functions very naturally to the appropriate objects make the object-oriented paradigm a powerful tool at the user interface as well.

A multidatabase system might be the best general solution for an advanced application. However, the development work will include many challenging issues. The major issues are discussed in [Bukhres and Elmagarmid, 1995] and are recapitulated in the following subsections.
The Multidatabase Schema

There is a range of solutions when constructing the database schema in a multi-database system. At one end, some multidatabases create and maintain a full global schema to describe all information available across the global system. At the other end, no global schema is maintained, and global queries must deal with multiple data sources and the transformations between them. In the latter category, the global access language has many constructs and operators to simplify and enhance the power of the user interface. In between these extremes, there are a variety of architectural possibilities that maintain partial global schemas and/or provide enhanced global language functions for dealing specifically with multidatabase issues.

Global schemas are often difficult to create and maintain. Global designers must understand heterogeneous local database structures and naming conventions and integrate large numbers of these into some cohesive whole –without being able to change anything locally. Creating a global schema is very knowledge-intensive work. It is also difficult to automate global schema maintenance because of the depth of knowledge required to judge the impact of local changes.

Multidatabase systems that rely on global access language functions rather than a global schema to integrate local schemas, usually extend the base language with operators and constructs for dealing with multiple data sources and the transformations between them. By not creating or maintaining a global schema, these systems eliminate most of the problems associated with global schemas. However, they sacrifice the logically centralized database image by exposing users to the idea that there are multiple data sources in the system (loss of some transparency).

Local Autonomy

Local autonomy is a virtue of multidatabases because it allows global access to added to existing DBMSs with minimal impact on local processing. The negative part is that autonomy causes problems for the global system because of the lack of global control.

Local/Global Transformations

Multidatabase software must translate between the local and global data model and access language. Mapping between data models is a difficult problem, but today there exists a variety of tools and a wealth of experience in this area. Mapping access language functions is sometimes more difficult because the level and amount of functions may differ between the local and global systems. If the local system has more functions, the global system just never uses the extra features. However, if the global language offers more functions than the local one, the translation software must make up the difference.
Global query Optimization

The essence of global query processing is similar in all multidatabases, see Figure A.3. Global query processing can optimize the path only from the global user interface to the local DBMS interface. The actual data is under the control of the local DBMS query optimizer. The distributed nature of multidatabases also complicates query optimization. The global optimizer must consider the performance of the communications paths and the power of potential nodes for local processing and subsequent result processing. Optimal decisions require complete, accurate, and instantaneous information about the state of the distributed system. The global optimizer should structure global query parsing, decomposition into subqueries, and subsequent recombination of subquery results so that communications costs are minimized and database processing is done at high performance nodes.

In spite of the many performance reducing steps, multidatabases can benefit from the parallelism inherent when subqueries are processed concurrently at local nodes. Another level of optimization is possible in the local/global translation software. Each local DBMS will have its access path optimized for a certain style of query. The local/global software can translate global queries to maximize the use of local optimizations.

Global Concurrency Control

Concurrency control is the coordination of concurrent processes that operate on shared data and may potentially interfere with each other. Centralized DBMSs typically use a lock on a data item to indicate that one process is updating the value and that other processes must wait until the update completes before reading the new value. The traditional concept of a transaction is a sequence of reads and writes that is short-lived and atomic. Atomic means either all the actions complete successfully or they all fail. This does not translate well to the multidatabase environment. One problem is that multidatabase transactions can be long-lived and may not need to be strictly atomic, they may reference multiple remote nodes, must go through multiple layers of translation, and must wait for local transactions to complete at the target nodes. So multidatabase transactions are inherently longer than local ones. Also, the types of applications of multidatabases tends to use longer-lived transactions. Another problem in multidatabase concurrency control is that local DBMSs do not provide local concurrency control information at the user interface, and the local DBMSs may use different concurrency control mechanisms. Still, any global transaction may have subqueries that interact with local transactions using multiple control techniques.

A variety of solutions have been proposed for multidatabase concurrency control, this is a focus of much current research. The earliest solution was to restrict multidatabase systems to be read only. Most concurrency control difficulties deal with coordinating updates and such restriction eliminates these problems.
Figure A.3. Multidatabase query processing.
A.2 Summary

This appendix has presented some database concepts and some different architectures. The most important concepts are the different data models:

- The relational data model, with its strict mathematical foundation together with a useful table representation.

- The object-oriented data model, in spite of the lack of mathematical foundation very useful and popular with its powerful means to model real-world objects.

- The object-relational data model, where the best features from the object-oriented data model have been added to the relational. A object-relational DBMS user sees the database as a set of tables, however, columns might have object-values and user-defined functions are available.

Also some architectures have been discussed; the centralized, the homogeneous distributed, and the multidatabase architecture. The multidatabase systems can offer the most general solutions, but also have many challenging problems in the design and implementation process that must be solved. The most important issues are; construct of the multidatabase schema, the autonomy of local DBMSs, local/global transformations, global query optimization, and global concurrency control.