KUNGL
TEKNISKA
HÖGSKOLAN

Evolutionary Development of Brain Imaging Meta-analysis Systems

Jesper Fredriksson

Stockholm 2002

Licentiate Thesis
Royal Institute of Technology
Department of Numerical Analysis and Computer Science

ISBN 91-7283-320-3
TRITA-NA-0215
ISSN 0348-2952
ISRN KTH/NA/R-02/15-SE

KTH Reprocentral, Stockholm 2002

Abstract

We discuss the development of a special class of complex systems, brain imaging meta-analysis systems and the role of databases in brain research. We further present the design and development of a subsystem, the workflow management system, following an evolutionary process with systematized reuse of database concepts to provide for cost-efficient development. The development effort includes a formal description and a graphical language for workflows.

ISBN 91-7283-320-3 • TRITA-NA-0215 • ISSN 0348-2952 • ISRN KTH/NA/R-02/15-SE

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis addresses the problem of constructing complex systems to meet the demands of:

- *Reuse of design, components and subsystems*
- *Specification of an evolutionary development process*
- *Providing functionality to manage complexity*

More specifically, we focus on constructing systems for management and analysis of brain imaging experiments. The 3D raster volumes typically delivered by brain imaging devices are large, approximately half a million voxels of real valued intensity values per scan. One brain imaging study may consist of more than 1000 scans of brains and it is estimated that 1500 new studies are performed each year. Moreover, the analysis of data is rather complex, as described in section 2.3.2 and also heterogeneously performed at different laboratories *and* techniques are still evolving.

As the number of published studies steadily increase, the need to be able to compare results from related studies becomes more and more pressing. But how can one compare results obtained from different laboratories using different experimental designs, imaging devices, analysis methods and file formats? A natural solution is to structure and collect them in a database, *in a format as close to the source as possible.*

The homogeneously processed raster data, together with careful experiment and processing descriptions will open up new possibilities for brain imaging research in allowing for meaningful *meta-analysis* i.e. inter-experimental analysis of data. We will here show how to pave the way for a new approach to studying the human brain, by a piecemeal growth of an environment successively more capable of aiding the single researcher in formulating new hypotheses about how the human brain functions.

1.1 Overview of the Thesis

Chapter 2 will introduce the reader to the field of brain imaging: basic facts about the brain, how functional experiments are designed and analyzed, and also how databases are used for meta-analysis of brain imaging studies. The next chapter deals with strategies for development of complex systems, such as brain imaging meta-analysis systems and aims at combining an evolutionary viewpoint on software engineering with reuse-oriented concepts, especially *mediator technology*. Chapters 4 and 6 deal with the theory and implementation of a *workflow management system* that is responsible for automating processing of experimental data and storing information about how the process was conducted in order to be able to compare results after the analysis. Chapter 5 introduces some tools used during developement of the meta-analysis system, most prominently the mediator system AMOS II, but also a rasterdata management system called RasDaMan and an elegant framework for development of drawing applications with semantical properties, named JHotDraw, used to create a graphical language for the workflow management system.

1.2 Brain Research Projects and Background

The work described in this thesis has been performed during three different projects.

- *ECHBD:* The European Computerized Human Brain Database aims at collecting functional and anatomical brain imaging data in processed format [RZ96] [Fre99] [FRS99].
- *BINS:* The Brain Imaging Neuroinformatics System collects and processes raw functional imaging data [FS01] [FSR01].
- *NeuroGenerator:* The NeuroGenerator project aims at collecting, processing and distributing processed databases of functional brain imaging studies [R$^+$02b] [FS01] [FSR01].

Compared to the, in the brain imaging community, well known BrainMap database [LFDM94], consisting of centre-of-gravity coordinates for activations reported in literature, the ECHBD system contains full raster volumes of result images and visual query functionality allowing a more quantitative approach to analysis of result images.

However, the heterogeneity of result data obtained at different laboratories prompted the construction of a database for raw data, to allow for a uniform

processing of all data sets and make end results more comparable. The fMRIDC database [H+01] also collects raw data, but has not yet advertised any processing enviroment for obtaining homogeneous result data and at present merely allows scientists to request raw data to be sent on CDs.

The BINS and NeuroGenerator systems are presently collecting raw data from partners in Europe and Japan and will offer Internet access to homogeneously processed result data as well as allow partners and contributors to download and/or request via mail whole databases of processed data and software. The name NeuroGenerator indicates a database *generator*, with access to super-computer facilities at the centre for parallel computers (PDC) at the Royal Institute of Technology in Stockholm.

In the sequence of architectures as presented in Fig. 3.12 we are currently moving from architecture (c) to (d), by interfacing the RasDaMan system described in section 5.2 to the AMOS II mediator system.

The graphical user interface to the workflow system described in section 6.5 is in a very early stage and the interfacing to the workflow database (WF-Server) is rather straightforward but will provide increased interaction with the workflow and clear and lucid feedback about the status of the workflow.

1.3 Scientific Contribution

The scientific contributions of this thesis is in methodology for development of complex systems for meta-analysis of brain imaging data. The focus is on a strictly organized reuse of concepts and components, described in chapter 3 building on [FS01] [FSR01], in which my contribution is roughly half and a third, respectively. The methodolgy is exemplified in chapters 4 and 6 by the development of a workflow management system for brain imaging meta-analysis, where the rapid development process resulting from reuse of database concepts such as object-orientation, active rules and transaction management led to a cost-effective development. The complexity of the task of construction of the meta-analysis system for brain imaging data needs a strictly defined methodology for its successful outcome, as does all system construction of complex systems. The main contribution of this thesis is in the systematization of reuse and evolutionary development to form a methodology applicable in a decentralized organization.

Properly extended and modified, it should also prove valuable and productive in a broder context than the development of systems for research, such as information system development, or any type of system that is complex enough, being constructed in organizations that are to some degree decentralized.

These results were obtained, first during the work on the ECHBD system, presented in [Fre99] and [FRS99] of which my contribution was 100% and roughly a third, respectively, and then during the work on the BINS and NeuroGenerator systems, accounted for in [FS01] [FSR01] and also in [RSL+01] where my contribution is roughly 10%

Other contributions, not previously described, are in the development of a workflow system for research purposes, where earlier projects such as the workflow system described in [A+98] building on the ZOO environment [ILGP96] and the system described in [C+94b] have pointed out the value of reuse of database concepts for scientific workflows, an approach that is carried over to this system. However, they fail to mention the special requirements of a workflow management system for *research purposes*. Obviously, the ability to trace the *lineage* of processsed data is of fundamental importance if any subsequent comparisons of data are to be made, something pointed out and *formalized* in the Petri Net framework of section 4.4. Other requirements are pointed out in section 4.4.3 and the implementations are described in chapter 6.

Chapter 2

Databases for Brain Research

This chapter will start by introducing some basic terminology and known facts about the brain. We then proceed to discuss the role of databases in brain research and introduce the important concept of *meta-analysis*, the key purpose of using large scale databases in brain research.

2.1 The Human Brain - Concepts, Terminology and Basic Facts

Introductions to the field of brain research can be found in [KSJ99] (neuroscience in general), [FFF+97] (brain imaging), [Rol93] (neurobiological foundations of brain imaging). The physiological background to brain imaging is outlined in chapter 2 of [Led01]. An ambitious attempt at systematizing neural science, focusing on general principles for neuronal systems can be found in [She98], while [CS96] deals with computational aspects of the brain.

The human brain is an organ with average volume 1400cm^3 and weight $1300 - 1400$g. Its surface, the *cerebral cortex*, appears wrinkled with bulges called *gyri* and furrows called *sulci*. It is believed that this $1.5 - 3.5$ mm thick, folded sheet of highly packed neurons is responsible for much of the computational processes the mind is capable of, and it is also the part of the brain that is more developed in humans than in other species. There are about 10^{11} neurons in the central nervous system (the brain, the cerebellum and the brain stem), of which the brain consists of approximately 2×10^{10} neurons and each neuron makes on average about 10000 synapses (connections) onto other neurons.

To describe locations in the brain, we will use the following terminology (Fig 2.1):

- *rostral* or *anterior:* forwards, towards the face

- *caudal* or *posterior:* backwards, towards the neck
- *dorsal* or *superior:* upwards
- *ventral* or *inferior:* downwards

For displaying images in 2D, we may want to section the brain, in one of three possible ways, by fixing one of the three (Cartesian) coordinates according to Fig. 2.1:

- *coronal:* fix the z coordinate
- *horizontal:* fix the y coordinate
- *sagittal:* fix the x coordinate

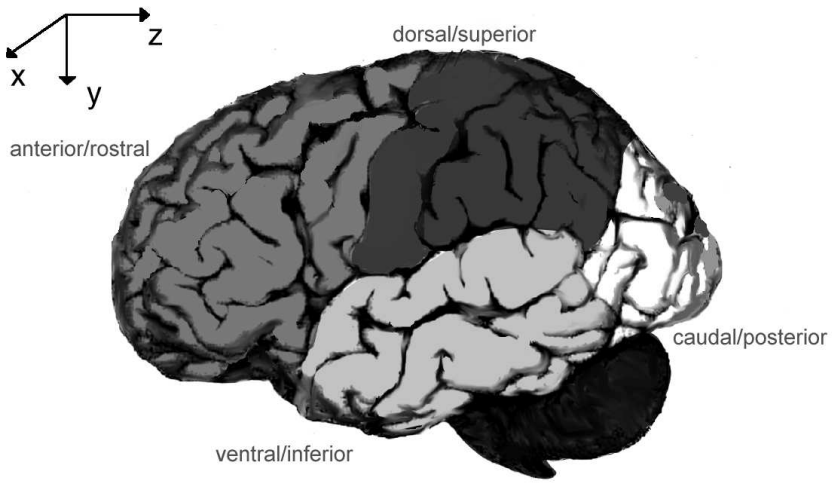


Figure 2.1. Directions and the three slicing planes

Further, the brain is divided into two morphologically relatively symmetrical halves, *hemispheres*, connected in the mid-sagittal plane through the *corpus callosum* (Fig. 2.2).

Some of the prominent landmarks and cortical areas are depicted in Fig. 2.3. First of all, the brain is divided into *lobes*: *frontal*, *parietal*, *temporal* (left and right) and *occipital*. The *central sulcus* marks the border between the frontal and parietal lobes. The gyrus posterior to the central sulcus, the *postcentral gyrus*, is the primary sensory cortex, while the gyrus anterior to the central sulcus, the *precentral gyrus*, is the primary motor cortex. In the

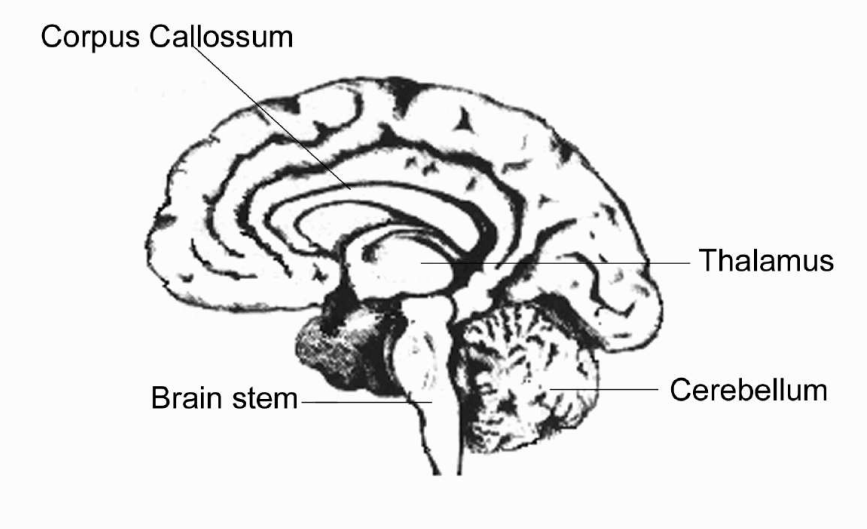


Figure 2.2. A sagittal slice at the midline of the brain, where the corpus callosum connects the two hemispheres

occipital lobe one finds the visual cortex and inferior to the occipital lobe is the *cerebellum*.

All of these areas and landmarks, except three, are anatomically defined. The motor, sensory and visual cortex are believed to be conglomerates of functionally homogeneous areas, *cortical fields*. Other examples of cortical fields are located within the olfactory cortex, processing smell, or the in the hippocampus, believed to play an important role in the forming of new memories. Damage to the hippocampus may lead to so-called *anterograde amnesia*,[1] where the ability to form new memories is lost.

Other than the cortical field hypothesis, there has been postulated a number of potential governing hypotheses regarding the way the brain represents information.

An example is the so-called "Grandmother cell hypothesis", stating that each object perceived by the brain is retained in one specific neuron (thus, for example, your grandmother would be kept in a single neuron). Despite the obvious problems of this theory, perhaps most notably the number of cells needed for long-term storage, there are signs that this type of coding is actually employed in certain regions of the brain. What can be said with certainty about this problem is that it is likely that different coding mechanisms are

[1] which anybody who has seen the film *Memento*(2000) by Christopher Nolan will recognize

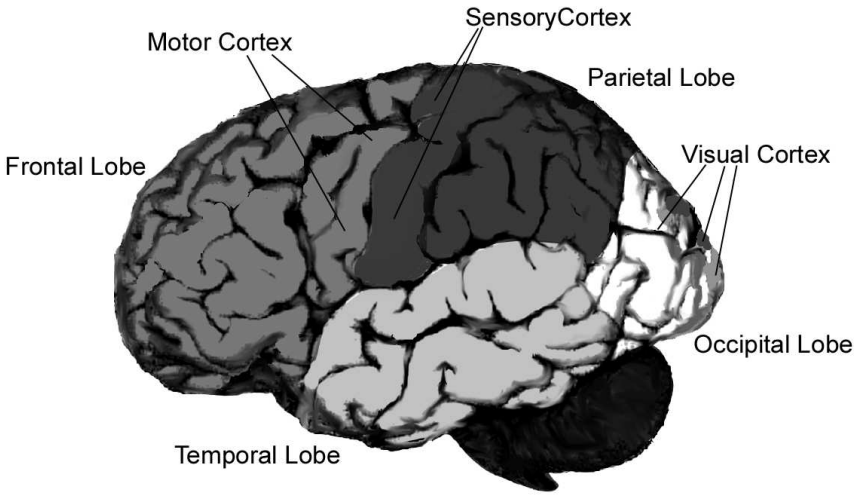


Figure 2.3. The cortical lobes and some functional areas

employed in different parts of the brain. Examples of this can be found in section 2.1.1.

2.1.1 The Brain as a Computational Network

Two closely related problems in brain research is how the brain represents and processes information. That representation and processing are closely related can be seen from a simple example from the visual cortex:

Consider the visual cortex, sub-divided into a number of areas [FvE91] such as V1, V2, etc. The classic Hubel and Wiesel studies [HW68] showed that in primate primary visual cortex (a.k.a. V1), neurons tend to respond strongly to specifically oriented edges in visual input. Further, neurons responding to similar orientations tend to be spatially close, organized in vertical columns. This representation is often referred to as *orientation preference* of V1 cells. Other examples of cerebral representation are topographical mapping of somatosensory areas (sensory input from neighbouring points on the skin tend to give responses in neighbouring neurons in somatosensory cortex) [DKH+02], frequency coding for hearing and the combinatorial odor mapping of the olfactory system [MS00] believed to be the basis for smell processing in mammals.

The orientation preference coding in V1 is just one example of several specialized regions for processing of visual input. It is believed that vision is implemented in the brain as several processing streams, together forming an integral conception of the surrounding world in the mind. Felleman and

van Essen demonstrate such processing streams in [FvE91], where the general idea is that visual input enters through the retina, is sent to a sort of "relay station", the thalamus, and then through different routes in the purposefully sub-divided visual cortex (e.g. V1, V2,...). Regions appearing later in the processing streams generally perform more complex and specialized functions, such as segregation of overlapping shapes [LAG+02]. Examples of visual processing streams are depicted in Fig. 2.4. This conforms to the neural network concept of a feed-forward net, which is a rather over-simplified model of the extremely complex recurrent visual processing in the brain.

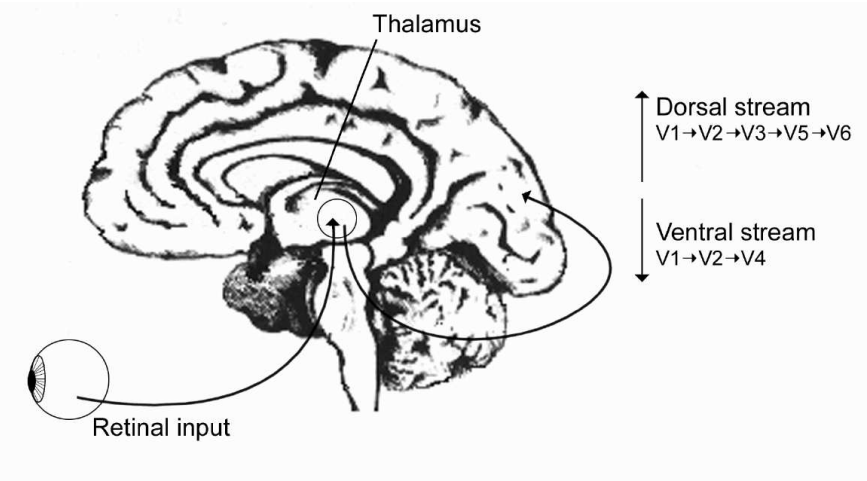


Figure 2.4. Global networks: the so-called "what" (ventral) and "where" (dorsal) visual processing streams

From the visual processing streams described above, it would seem natural to think of neural processing as occurring within distributed networks. Our visual conception of the surrounding world may thus be formed by interconnecting several specialized neural sub-systems, or cortical fields. We refer to these *functional networks* as *global networks* to emphasize that their constituents are also networks, *local networks*. This idea is put forth in [She98], where the editor stresses the importance of integrating knowledge from the many levels of neuroscientific work being done.

The local networks, then, consist of *neurons*, *axons* and *dendrites*. A simplified and crude description of the function of a neuron is that it sums up input from incoming dendrites and, if the input exceeds a threshold, emits a so-called "spike", through the outgoing axon. An interface from axon to dendrite is called a *synapse*, where pre-synaptic electrical signals are converted to chemical molecules which diffuse to a post-synaptic site where depolarization or hyperpolarization occur, which will have either an *excitatory* or an

inhibitory effect on the target neuron. In reality, synapses are extremely complicated relay junctions, where among other things, long-term memories may be formed through repeated stimulation, and thereby strengthening, of this connection [SK91].

Local networks are formed by combining excitatory and inhibitory synaptic actions in structures that to some degree can be approximated by simple boolean circuits, see Fig. 2.5. In [She98] it is shown how local circuits similar to that in Fig. 2.5 implement neural functions like direction selectivity in the retina and rhythm generation in the motor cortex.

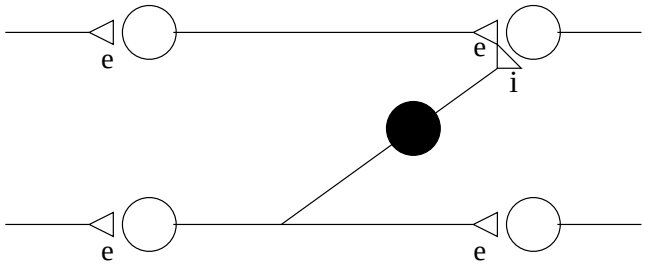


Figure 2.5. A simple local circuit. Note the effect of the inhibitory interneuron (i), regulating excitation in the upper right excitatory (e) neuron.

In essence, then, we have found that the brain is made up of networks on several levels[2]. As we will see next, contemporary brain imaging techniques are too coarse in spatial resolution to be used for mapping local networks, but may reveal features of the global networks.

2.2 Studying Brain Function Using Imaging Techniques

Brain imaging research focuses on improving our understanding of brain function, by measuring changes in physiological variables during experiments where the subject performs a set of pre-specified tasks.

We focus on the function of the *normal* (healthy) human brain, which is often studied using one of the following broadly defined methods:

- *Invasive methods:* These methods are quantitative and generally more accurate than e.g. PET and fMRI, but can only be performed on animals and the transfer of results is often non-trivial. Examples are measurements on electric current and optical imaging.

[2] we could go on to describe intra-cellular mechanisms, or synaptic function, but that is not the focus here

- *Studying lesion/impairment effects:* Either a study of an impaired human brain, or of a lesioned (in general *non-human*) brain, utilizing a contrapositive proof strategy. Lesion studies on animals require transfer of results from non-human to human brain.

- *Non-invasive imaging techniques:* Directly observing changes of a brain activation dependent variable in human or non-human brains during pre-specified external conditions. A typical deficiency of this class of methods are inherent limitations in precision of contemporary imaging techniques.

- *Meta-analysis:* Can be defined as inter-experimental analysis of data, for example testing reproducibility of activations across similar studies.

The data dealt with in the BINS and NeuroGenerator projects are mainly the following:

- *Cytoarchitecture:* microstructural data from post-mortem brains containing quantitative measurements of the density of nerve cells.

- *Anatomical data:* a 3D MRI [SB92] scan of each subject's brain

- *Functional data:*

  - Positron Emission Tomography (PET): a method by which positron emitting radioactive isotopes are incorporated into tracer molecules which are injected into the blood stream. The PET scanner then measures the concentration of the tracer molecules reaching the brain. For a more comprehensive description, see [Rol93]. With PET, one can measure e.g. the regional cerebral blood flow (rCBF), which, since the depolarisation of neurons is related to the rCBF, can be used to locate neuronal activity changes in the brain.
  - functional Magnetic Resonance Imaging (fMRI): fMRI is a method by which the radio signals from nuclear spins are recorded at Larmor frequency. By changing the magnetic field gradients around the brain, one can get 3D images of the spin signals from protons and other nuclei. When neurons are working hard, the deoxyhemoglobin concentration decreases locally and thus reduces disturbances of the proton signals. The MR signal is stronger from such regions. For a full description of the principles of fMRI, see [MB99].

In the case of fMRI and PET, there is currently a spatial resolution of approximately one millimeter. 1mm^3 cortical tissue contains approximately

44000 neurons. The restrictions this limitation of resolution puts on brain imaging research are of course severe. For example, we will not be able to observe the inner workings of local networks as described in section 2.1.1 with these methods.

Other than the spatial resolution of imaging methods, there are a number of difficult problems for brain imaging resarch to deal with, some of which may be dealt with using databases:

- *Inter-individual structural variability:* To be able to compare results obtained from different brains, a transformation to a *standard anatomical format* is applied for measurements on each brain, see section 2.3.2. Thus, inter-individual structural variance should be small compared to the voxel-size used for subsequent statistical analysis. The inter-individual structural variance can rather easily be estimated using a database of anatomical data.

- *Neural representation:* There are many indications [Rol02] supporting a *cortical field hypothesis*, that neighbouring neurons tend to work together and thus become "active" simultaneously. But what if this is not true in general? How could one ever disprove the cortical field hypothesis? It is difficult to envisage for example a statistical test which could be used to assess this hypothesis.

- *Inter-individual homogeneity of functional organization:* Even if brains appear similar in structure, it may still be the case that some of their *functions* are differently implemented [Str02]. It is not impossible to imagine functions of the brain that could be encoded radically differently in different humans, such as the ability to perform arithmetic etc.

- *Bloodflow (PET) or blood oxygenation (fMRI) as indicator of brain activity:* In chapter 2 of [Led01], it is argued that adherence between the commonly used activation indicator variables for PET and fMRI and *actual* brain activation is stable and rather stable, respectively. However, in the case of fMRI, the chain between measured variable and activation is rather lenghty and some concerns about linearity, assumed in subsequent statistical analysis, can be raised.

2.3 Experimental Design and Data Processing in Brain Imaging

This section will give a brief description of the structure of imaging data introduced in section 2.2. We will focus on PET experiments because the analysis

methods have a higher degree of maturity. One of the main differences between PET and fMRI studies are that measurements in fMRI are collected much faster and thus we have also a temporal dimension to address in the analysis. More specifically, the assumption that images obtained at two different, but neighboring points of time are independent, is not valid anymore.

2.3.1 Baseline PET Experimental Designs

The principle behind baseline experimental designs is depicted in Fig. 2.6 and summarized in table 2.3.1. The activity in the brain is measured under a relatively long period of time (approx. 1 minute) and then averaged to account for effects from variability of brain activity and intensity of radioactivity. After a pause, during which the brain returns to rest state, a new image is registered, while a new condition is imposed on the brain.

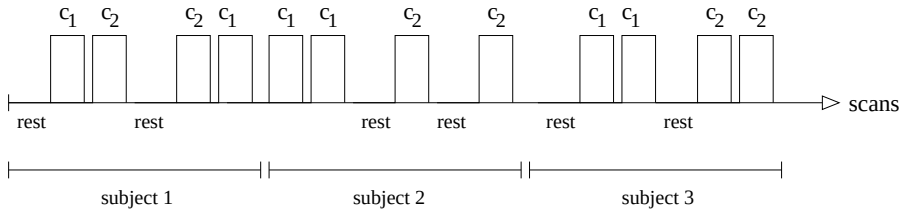


Figure 2.6. An example of a baseline experimental setup for PET imaging, with $I = 3$, $J = 3$, $K = 2$

An experiment consists of I *subjects*, subject to J experimental *conditions*, which are each presented to the subject and repeated K times (*repetitions*). Additionally, each subject is scanned once by an MR scanner to obtain an anatomical image. During a baseline setup of experimental conditions, as in Fig. 2.6 one commonly measures change in bloodflow (as described in section 2.2) which results in $N = IJK$ functional images. Below, we also consider the I anatomical images and one *standard brain*.

One now needs to decide upon a set of *explanatory variables*, $V = \{v_i, v_2, \ldots, v_M\}$, as in classical hypothesis testing. Explanatory variables can be a partitioning of conditions, subjects, or some measured covariate (such as response times), or any function thereof. In the experiment of Fig. 2.6 we decide to simply use conditions as explanatory variables, together with the mean of activation, μ. Thus: $V = \{\mu, c_1, c_2, \text{rest}\}$.

To summarize the layout of the experiment, we now relate N images to our M explanatory variables in a *design matrix*. The design matrix X corresponding to the experiment of Fig. 2.6 will, with the aforementioned explanatory variables, look like:

Experiment	"ExperimentName"
Description	"..."
Subjects	s_1, s_2,...,s_I
Conditions	c_1, c_2, ..., c_J
Repetitions	1, ..., K

I anatomical images
$N = IJK$ functional images
One *design matrix* (**N**x**M**) relates N functional images to M explanatory variables.
Finally, n *contrast vectors* (**M**x**1**) describe *null-hypotheses* to be assessed (see below)

Table 2.1. Summary of experimental design and statistical modelling

$$X = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

The usual method of analysis is to employ, for each voxel value y, a linear model [Gra76] as suggested in [FFF+97]:

$$\mathbf{y} = X\beta + \mathbf{e}$$

We estimate β by the method of least squares:

$$\hat{\beta} = (X^T X)^- X^T \mathbf{Y}$$

We here employ a generalized inverse of a matrix A, denoted A^-, for which it holds that $AA^-A = A$. We may now wish to test a hypothesis, $\mathcal{H}_0$, involving the *contrast* between explanatory variables, defined by a *contrast vector*, $\mathbf{c}$.

$$\mathcal{H}_0 : \mathbf{c}^T\hat{\beta} - \mathbf{c}^T\beta = 0$$

For example, to test the difference between condition 1 and rest, we define:

$$\mathbf{c} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ -1 \end{pmatrix}$$

The hypothesis $\mathcal{H}_0$ can be assessed if we assume $\mathbf{e} \in \mathcal{N}(0, \sigma)$ by comparing

$$\frac{\mathbf{c}^T\hat{\beta} - \mathbf{c}^T\beta}{\sqrt{\hat{\sigma}^2\mathbf{c}^T(X^TX)^-\mathbf{c}}}, \text{ where } \hat{\sigma}^2 = \frac{\mathbf{e}^T\mathbf{e}}{N - \text{rank}(X)}$$

with a Student-t distribution [Gra76]. Depending on the outcome of the comparison, we either accept or reject the hypothesis. The key assumption thus made is that all measurements on a voxel can be regarded as uncorrelated with common variance, a simplification that is somewhat justified by the relatively long intervals between measurements.

Another problem, subsequent to the univariate hypothesis testing described above, is how to set the threshold for rejection of the null-hypothesis.

A common threshold used is to, for each voxel, reject the null-hypothesis at a 0.05 significance level, i.e. the probability of falsely rejecting $\mathcal{H}_0$ is at most 5%. But, if testing e.g. 500000 voxels, we will on average falsely reject $\mathcal{H}_0$ for as many as 25000 voxels.

Moreover, the voxels can not be regarded as uncorrelated, for many reasons, and the standard *Bonferroni* correction of the significance level to 0.05/500000 will most certainly be too restrictive. One such cause of correlations is that, according to the cortical field hypothesis, the intensity values of neighboring voxels *within* a cortical field should be correlated within a scan where this cortical field is involved in the neural computation.

To adjust for these correlations we may choose to follow [Led00], [FFF+97] or [GLN01], a process described below in text and figures as "clustering".

2.3.2 Image Processing: From Raw Image Data to Statistical Cluster Images

The images scanned according to section 2.3.1 are raw, unprocessed data[3]. We may now wish to statistically analyze the data in the manner outlined in section 2.3.1, a process containing several steps, Fig. 2.7.

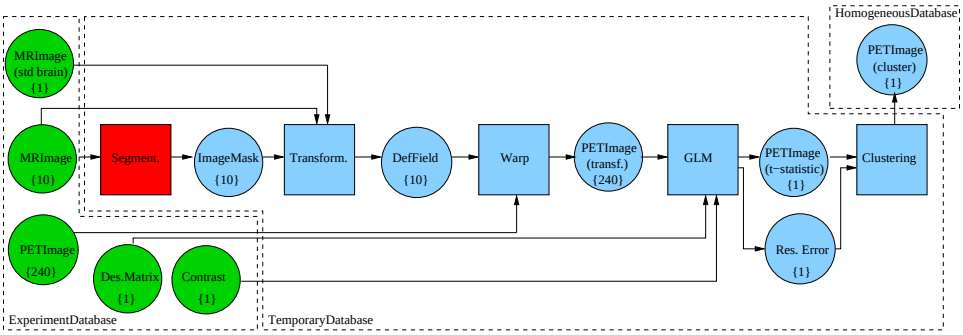


Figure 2.7. The PET processing chain for a single experiment with 10 subjects, 3 conditions and 8 repetitions. Numbers in parentheses are number of instances

The very much hypothetical example of Fig. 2.7 is made up of ten subjects, three conditions and eight repetitions of each condition. That is, each subject is scanned $3 \times 8 = 24$ times in the PET scanner and the anatomy of the brain is scanned once in an MR scanner. In total for this experiment, we get $10 \times 24 = 240$ PET images and 10 MR images. We only test the hypothesis specified by one contrast. Below, we will briefly review a typical statistical analysis of this experiment, focusing on conceptual stages of the analysis, referring to the data flow in Fig. 2.7 as a *processing chain*.

- *Segmentation:* The scanned anatomical images contain portions depicting soft tissue and bone, which we do not want to influence subsequent steps in the processing chain. A step where these portions are removed from the image can be either manual or based on automatic image analysis. The automatic algorithm described in [UL02] used in the NeuroGenerator/BINS projects takes approximately 45 minutes to complete using a single node of the parallel computer, for each anatomical image.

- *Transformation:* To be able to compare images from different subjects, we compute a transformation from each segmented anatomical image into the *standard brain*, a *median* brain chosen from a population of

[3]Except from a reconstruction stage performed within the scanner

young males. A number of different image processing algorithms exist for this purpose. A typical requirement on this transformation is that it should preserve topological relations in the image. We currently utilize the AIR package [WCM92] for this purpose, which takes roughly 5 minutes per segmented anatomical image to complete.

- *Warping:* This stage of the processing chain simply applies the previously computed transformation to each *functional* image. Completes in rouoghly half a minute per functional image.

- *General Linear Model (GLM):* This is the procedure described in section 2.3.1, to perform a voxel-wise least-squares estimation and subsequent hypothesis testing. The end-result is an image of uni-variate Student t-distributed voxels. This step is performed once per experiment and the simple matrix operations takes less than a minute to perform.

- *Clustering:* To account for correlations within the image, a method of thresholding voxel values is employed. The basis for this algorithm is either theoretical results obtained from random field theory [Adl91] as in [FFF+97], or based on distribution-free tests [Hol96], Monte-Carlo simulations [Led00] or the false-discovery rate [BH95] [GLN01]. In the BINS/NeuroGenerator projects we have initially chosen to use the computationally intensive Monte-Carlo method, running 4500 simulations, lasting for seven hours. If several contrasts are specified, the results of simulations can be reused.

These are the main conceptual steps of the processing chain. When actually executed, some further steps are neccessary. More details on executing the processing chain and its input and output data can be found in section 6.1.

2.4 Databases for Brain Imaging Meta-analysis

Given the rather complex experimental designs and analysis depicted in previous sections, how can the brain imaging community be helped by databases? Which problems can be addressed by using databases, and which can not?

2.4.1 Problem Solving Environments

One of the main ideas behind BINS, NeuroGenerator and, before that, ECHBD is data sharing between research groups *and* also the sharing of a common piece

of software for accessing the data. This, in turn, leads to a form of distributed collaboration between groups and, hopefully, higher productivity.

The data-centric system for brain imaging research we thus seek shares a number of characteristics with *Problem Solving Environments* (PSEs) [WRL+00] [THF+97] [PMHJ98] [GHR94] [RB96].

In [WRL+00], key aspects of a PSE are identified as:

- *intelligence:* to ensure the PSE is easy to use and computationally efficient
- *collaborative tools:* to allow effective collaborative work on complex problems
- *visualization:* to support data analysis and navigation, as well as provide visual support for runtime monitoring and steering of applications.

For users working with brain image meta-analysis it is important to be able to:

- *Access large sets of well-structured raw data:* The data is structured in a common format with descriptive data, organized according to a database schema.
- *Access tools to process data into a homogeneous statistical format:* This is the processing chain described in section 2.3.2. The fact that all data are homogeneously processed will make end-results comparable to a much higher degree than if processing had occurred at different laboratories, using different implementations of various algorithms.
- *Monitor processing of data:* The workflow system described in chapter 6 enables monitoring of data processing as well as automatic parallelization on a supercomputer.
- *Visualize image data and corresponding descriptive data.*
- *Add own algorithms to the present set of state-of-the-art processing tools.*
- *Access tools for post-processing:* The ECHBD project [RZ96] [FRS99] [Fre99] for example, offers visual search and boolean operators on cluster images and cytoarchitectural maps. Since all users have access to the same functionality and the same data, all post-processing results can be reproduced at all user sites.

- *Contribute to the set of open-source post-processing tools:* The NeuroGenerator desktop, currently under development, will offer an open architecture [SF02] to encourage users to write their own applications for the processed data.

To facilitate development of these solutions we have in this work used the following key technologies:

- *Object-oriented database technology:* To model the complex data structures *and* computational processes on data, object-orientation is needed. Because efficient management of large sets of data is important, we use the data-centric approach of database technology.
- *Mediator technology:* As we will see in section 3.2.2, the mediator approach provides an efficient evolutionary path for system architecture when data sets grow and system requirements change.
- *Active database technology:* The management of image processing can benefit from a state-transition paradigm as offered by for example Event-Condition-Action (ECA) rules, as will be presented in chapter 6.

The experience gathered from development of the NeuroGenerator/BINS environment shows that re-use of known concepts from database technology as listed above can be utilized in a system development process characterized by rapid prototyping and iterative system refinement. Section 3.4.1 will display the evolutionary process from a use-case perspective, while chapter 6 will describe the development of a workflow system that manages the previously defined processing chain.

2.4.2 Meta-analysis and Data Mining

Meta-analysis can be defined as analysis across sets of results originating from more than one experimental study. It is generally associated with testing a hypothesis (e.g. consistency of activation patterns), while data mining on the other hand can be used to perform more assumption-free searches for patterns, such as finding hidden variables in studies by use of correlations.

Early attempts at meta-analysis within the field of brain imaging research were often made by carefully reviewing relevant literature, grouping studies based on their experimental design (conditions used, scanners involved and contrasts computed). A direct comparison of published results often show inconsistencies in findings. Reasons for this can be several:

- *Different scanner types and/or settings*

- *Different image processing algorithms and/or implementations*
- *Different parameter settings in image processing algorithms*
- *Different standard brains*
- *Hidden variables in experimental designs, not documented in literature*
- *Systematic inter-subject differences in morphological structure and functional brain activation*
- *Incomplete understanding of dependency between scanner-measured neural variable and neural activity*

The BINS and NeuroGenerator projects aim to overcome many of these issues by carefully documenting experimental designs and using consistent methods for processing image data.

Examples of meta-analysis and data mining on brain images are appearing in the literature, for example in schizophrenia studies [W+96] [HM92] [W+00] [ZPHJ00], reproducibility of brain activation [CSAK99] [PLGL00] [NH02] or connectivity [S+01].

2.4.3 Brain Imaging Meta-analysis Database Systems

Over the past decade there have been numerous attempts at creating databases for management of brain imaging experiment [LFDM94] [RZ96] [C+94a] [LDBo] [HDJM+00] [A+96] [D+97] [CKKE97] [MDH99] [H+01] [RSL+01]. A detailed description of all projects and databases is clearly beyond the scope of this thesis. This section will instead highlight some important features of database systems for brain imaging meta-analysis.

The most basic need suggesting the use of databases is that of comparing results across laboratories, a goal which can to some degree be fulfilled by a simple file-sharing utility. However, the ultimate goal will always be to obtain a better understanding of the function of the brain. To support this goal, the database should offer more than purely experimental image data, so that the user of the system may obtain more information from the collection of experimental results than each experiment in isolation. To be able to extract this information from a database available of experimental results, the database system should offer tools for:

- *Data access:* How should users access the database? For smaller datasets, such as those basing activation data on centre of gravity coordinates, bandwidth is not a problem, but when accessing several full 3D volumes over the Internet, bandwidth restrictions may cause a problem.

Mirror-sites and distributed systems may help to overcome this problem by introducing redundancy, perhaps in the form of multicast sessions [FJL+97]. In the NeuroGenerator project, raw data suppliers will be offered whole processed databases for download or sent by CD, DVD, tape etc.

Further issues in this category are which types of data to allow access to: e.g. *processed* or *raw* data, and which types of imaging devices to support for in the database. Two types have been mentioned here so far: PET and fMRI, but other types exist (EEG, MEG, Spect, ...). Raw data collections are magnitudes of order larger than the corresponding processed data and will put higher demands on a system for managing and distributing them.

- *Searching the space of experimental data:* The body of conducted experiments is huge, estimated to consist of roughly 10000 subjects and 100 TB of data and also *growing* with 1500 new imaging studies per year [OfHBM01]. Obviously, the experiments will have to be carefully catalogued, indexed and made searchable, both with respect to descriptive data, *meta-data*, and the content of images.

  One of the earliest established databases, BrainMap [LFDM94], provides activation data in the form of centre-of-gravity coordinates, searchable in so-called Talairach space, and linked to published papers. The ECHBD system [RZ96] [Fre99] [FRS99] provides full rasterdata volumes in the form of thresholded statistical images, linked with structured descriptions of the experiments with pointers to literature.

- *Meta-analysis, data mining and modelling:* To support meta-analysis, the data should be as homogeneous as possible with respect to acquisition and pre-processing. One may however argue that total homogeneity of data is impossible and the best we can settle for is to provide traceability of processed data, a *lineage*, as defined in section 4.4.3. To allow for advanced usages of the database, the system needs to be powerful in expression and also *extensible*, because one will not know in advance what functionality eventually will be required. An extensible query language meets many of the requirements, but may also impose high cognitive demands on users.

- *Exchange formats:* There will be many co-existing databases, of which, some will specialize in the types of data they provide or what functionality they offer. Others will connect to or even be integrated with other databases. It may thus be valuable to agree upon an exchange format for data between databases, using XML or other meta-modelling languages.

The ability to "check out" an experiment available from a database in an exchangeable transfer format will certainly appeal to many users.

- *Advanced visualization:* Most contemporary imaging techniques deliver 3D scans of the brain and in some cases (e.g. fMRI) even time-series of 3D scans. The database system should aid the researcher in learning about patterns in the multi-dimensional activation space, such as the global networks (section 2.1.1). In the Genesis projects, efforts were made to utilize virtual reality for visualization [LDBo].

In additions to these issues, there are more mundane problems to be tackled like *data ownership*, *submission procedures*, *formats of data*, data security and guaranteeing the privacy of experimental subjects. Also a critical issue, both mundane and advanced, is quality control of data, which will involve both manual inspection, submission procedures and data mining efforts [NH02].

Yet another issue is the integration of multi-disciplinary data [GLM00], such as genome data, imaging data and direct recordings from cellular and molecular structures, as well as how to manage a comparison with non-human correlates [S$^+$01].

Chapter 3

Evolutionary Strategies for System Development

A contemporary textbook on software engineering [Som01] states that there are four different *software process models*:

- *Waterfall model:* The classical step-wise development process from specification to operational system. Its inflexible nature and long startup time makes it rather risky to use in practise.
- *Evolutionary development:* The focus is on rapid prototyping and early user involvement. Either based on successive growth of the system, or on a succession of throw-away prototypes. Basing development on successive growth is attractive, but the evolutionary growth does not necessarily produce the best end-result, especially for larger projects.
- *Formal systems development:* Development based on formal requirements specification.
- *Reuse oriented development:* Formally not so different from the waterfall model, but focus is on finding COTS ("Commercial-Off-The-Shelf") components and/or develop own components. May lead to compromises when implementing requirements.

Disregarding the formal systems development option for now, either evolutionary- or reuse-oriented development seem to be the most attractive options. Below, we combine the two approaches into a controlled evolutionary process, using middleware, specifically mediator technology to provide an invariant interface between different evolutionary stages. The control part of the process is designed so that end-results do not become unnecessarily complex, to make sure the resulting process is low-cost and low-risk.

We will promote the use of this reuse-oriented evolutionary system development process for architectural analysis of large and complex systems by way of an example; the construction of a brain imaging meta-analysis system.

3.1 Evolutionary Software Process Models

The process of construction of large software systems can be divided into three different but closely related classes of activities:

- *Project Management:* The organizational and administrative tasks that control other activities.
- *Architectural Design:* The high-level design of the system: requirements specification and use-case analysis, leading to, for example, decisions about type of software reuse (sections 3.2.2 and 3.2).
- *Software Development:* Design, development and integration of components of the system.

In [Tha02] basically the same view is expressed, but from a more organizational point of view. A software process model basically belongs to the project management level (but remember that activity on different levels are closely related) and focuses on how to "...determine the order of the stages involved in software development and evolution and to establish the transition criteria from one stage to the next." [Boe88].

That this activity is important is evident from the oft-cited "software's chronic crisis" [Gib94]: many software development projects fail to meet the schedule or do not deliver a fully satisfactory end-product. It may seem that if an instance of a software process model is reasonable and if its stages are followed, the end-product will be satisfactory. Either things are not that simple, or software process models in general are not good enough.

An evolution of process models is sketched by Boehm in [Boe88], starting with the naive *code-and-fix* model, all too commonly resulting in "spaghetti code", followed by the waterfall model, criticized for being too document-driven. Its successor, the evolutionary development model, is generally suitable for vaguely defined systems, where the prototyping stages help identify requirements, but may lead to an evolutionary "cul-de-sac". Boehm refers to the formal systems development model as the *transform model*, and argues that its application domain may prove to be very limited, a common concern about this model[1], and introduces the *risk-driven spiral model* as a mixture of a waterfall and an evolutionary model, but based on a *risk analysis* of each

[1]...but today the pendulum may have swung

stage.

More recent research tends to focus on tailor-making and parameterizing process models for specific project instances, specifying process model patterns or pattern systems for process models [BFG93] [GKF00], automating portions of development [G+96] or returning to a more formal approach [All97] [LKA+95]. Also of great interest is to find a framework within which one can compare, simulate and model development processes [DI01] [BW01].
The process model specified in section 3.4 aims at development of especially complex systems for loosely co-operating sub-projects according to a combined evolutionary and reuse-oriented paradigm.

3.2 Evolutionary Development Using Component-based and Mediator Technology

This section and the next will discuss two different views on reuse-oriented development. Reuse in software development can occur at many levels. Here, we focus on:

- *Design reuse:* Guidelines for structuring solutions, such as *patterns* or *pattern systems.*
- *Code reuse:* Actual reuse of class-libraries and components.
- *Systems reuse:* To build a system from reuse of other sub-systems, such as a database manager.

The type of reuse we will discuss in this section belongs to the second and third level, i.e. components and systems reuse by use of brokers and/or mediators, while section 3.2.2 will focus on the first and second levels.

The component object model and mediator database systems provide basically the same functionality: a homogeneous view over a heterogeneous set of components. We briefly define the two concepts and note some of their differences before concentrating on uses of mediator technology in evolutionary system development.

3.2.1 Component Object Models and Mediator Technology

Component object models like CORBA [Vin97] and COM/DCOM/COM+ focus on achieving (distributed) object interoperability by adding a communication layer that all requests go through. This layer is in CORBA terminology

called the ORB, or *Object Request Broker*. Redrawn from [Vin97], Fig. 3.1 captures the idea behind CORBA object communication and the role of the ORB.

The main idea is to support reuse of compiled objects through a message-passing architecture for inter-process communication.

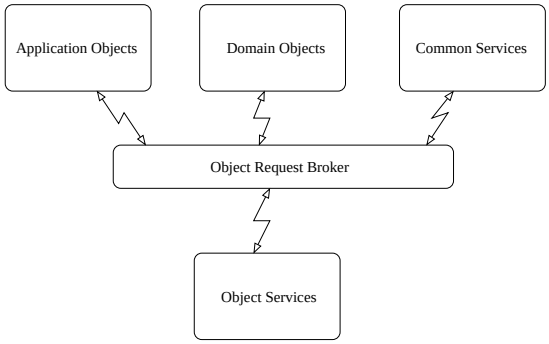


Figure 3.1. The CORBA architecture

All communication between objects will thus go through the ORB, costing some overhead in communication, but on the positive side, change in one component's interface will not require changing other referring components.

Comparing this approach with the common communication pattern *mediator*, we note that mediator "is-a" broker [LLM98]. In the words of [GHJV95] "A mediator serves as an intermediary that keeps objects in a group from referring to each other explicitly". This is important because of the reuse possibilities it enables: since components do not refer explicitly to each other, change in one of the N components does not implicate change in any of the other $N-1$ components in contrast to the worst-case, when all components refer to each other and all $N-1$ components will have to be updated.

Use of the mediator pattern is the standard way of achieving interoperability between non-homogeneous distributed database systems. Fig. 3.2 describes the typical architecture of a *wrapper/mediator* system [Wie92] for multidatabases. Relevant parts of a query issued to the mediator is sent to each wrapper and there rewritten according to pre-specified rules. Partial results are sent back to the mediator which combines them and presents the end result in a unified way to applications.

Comparing the mediator of database technology with the broker concept of distributed systems, the basic goal is the same: creating a homogeneous view over heterogeneous components. There is, however, a major difference in the granularity of message-passing objects. In the distributed system world, all communicating objects go through the ORB, while database mediators only

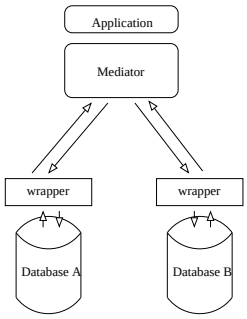


Figure 3.2. The wrapper/mediator approach

have to mitigate between heterogeneous *datasources*, such as whole database systems.

Nevertheless, the mediator approach allows distributed databases to work on different operative systems, using different database management systems, different data models (relational, object-oriented, ...), different database schemas, and so on.

As we will see from examples later on, the databases may be e.g. specialized raster data management systems [FSR01], ordinary filesystems, XML documents [LRK00] or standard relational database management systems [FR97].

Assume that we have a system for cataloging brain imaging experiments, stored in a relational database that can perform basic queries on descriptive data for each experiment and point to catalogs on disk where image data is stored, as in Fig. 3.3.

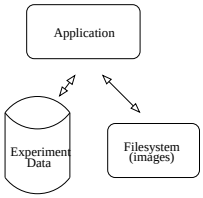


Figure 3.3. A simple database of experiments

We now face the task of creating a new system that will make us able to query both descriptive data *and* image data simultaneously, for example to phrase the query Q = "Give me all images that have a pixel value greater than x and where all subjects involved were left-handed".

Can we reuse the old system, and, if so, how do we add the image capabilities needed? Yes, by the use of a mediator, according to Fig. 3.4.

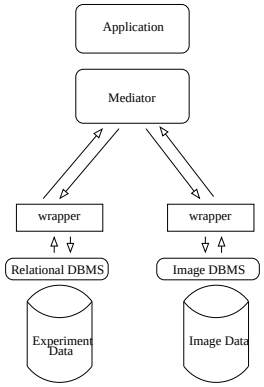


Figure 3.4. An image database system

This is the "don't scrap it - wrap it" approach [RS97] for reuse of legacy systems. We will be able to reuse the old system, but with added total system complexity compared to a top-down solution. On the other hand, this complexity is visible to neither end-users nor application programmers!

3.2.2 Mediator Technology in Evolutionary Development

According to section 3.2, mediator technology provides interoperability between databases.

Let us go back to the example from the previous section of an image database system, only now we do *not* already have a (legacy) system for cataloging experimental descriptions, but we still want to develop a system capable of answering the query Q previously defined.

Is it still a good idea to first build a system like that in Fig. 3.3 and then move to the system of Fig. 3.4?

Normally, the answer would be "no", because of the added complexity of the system. A better solution would be to extend the database with a new type [SM95], capable of modelling images, by the use of type-extensions (called *datablades* in Informix, *cartridges* in Oracle). However, these type-extensions are generally not easy to implement and usually require a deep knowledge of the base DBMS to be extended.

On the other hand, by using a pre-existing specialized DBMS such as the *RasDaMan* DBMS (see section 5.2) for managing image data, we will benefit from reuse of:

- *Data types:* The relevant data types are already present in the specialized DBMS and can be modelled in the data model of the enclosing mediator

system, delegating all the operations to the specialized image DBMS subsystem.

- *Query language constructs:* When extending the query language of the mediator system with primitives for image operations, the constructs offered by the image DBMS can be reused.

- *Optimization techniques:* In the RasDaMan system, a special storage model (see section 5.2.1) provides optimization of subimage retrieval - the system fetches from disk only those parts of the image that are requested. Other query optimization techniques such as query transformation, and indexing structures of the image DBMS can also be reused.

The specialized image DBMS will probably not be good at modelling ordinary relational or object-oriented data, and the end-solution will by necessity involve an integration of two separate data sources. In the ECHBD project [Fre99] [FRS99], the responsibility of system integration was put on the application programmer, leading to a solution with poor future extensibility.

An alternative solution is to store the images as BLOBs (Binary Large OBjects), with no semantics attached, and interpret the data in the application layer. However, this does not allow us to utilize query optimization of the image part of the query, which in some cases would lead to severe restrictions on the queries we are able to submit to the system.

So, what are the positive effects of using a solution such as that in Fig. 3.4? As stated above, we reuse all the strengths of the pre-existing specialized DBMS instead of having to re-develop all domain technology already available. The wrapper-mediator architecture also allows an evolutionary development process:

- *level one:* The bare-bones system with no image functionality, corresponding to Fig. 3.3.

- *level two:* Full functionality - the image DBMS manages image data.

- *level three:* Full functionality and good performance - the indexing and optimization techniques of the sub-system are utilized.

Thus, as a side-effect of this developmental strategy, we can first test concepts by rapidly prototyping the system of Fig. 3.3 and then gradually increment functionality and performance, thus employing an evolutionary approach to the system development.

3.3 Reuse-oriented Development using Patterns and Frameworks

The object-oriented movement of the eighties is generally considered a big step forward in reuse-oriented development. This notion of progress is tightly linked to the advent of *inheritance*, i.e. hierarchical specialization of behaviour. It can be said that object-orientation helps developers in thinking about and structuring many real-world applications, a very important advance in software engineering.

In [Kre92], software reuse is said to involve the following concepts:

- *abstracting*
- *selecting*
- *specializing*
- *integrating*

Of these general concepts, all were attainable before the advent of object-orientation, although, most notably, a more powerful specialization mechanism was provided for through inheritance. It can be argued, however, that *composition* can provide the same functionality as inheritance; in practice at least this author tends to use composition more frequently for specialization of behaviour.

Pattern technology draws upon ideas from object-orientation to describe frequently occurring problem-solution pairs, the classical "Smalltalk"-example being that of a *model-view-controller* pattern for creating (graphical) user interfaces [KP88] [GR83]. In [BMR+96] patterns are hierarchically classified into *architectural patterns* (e.g. model-view-controller), *design patterns* (e.g. singleton) and *idioms* (language-specific low-level patterns).

Below, we give two perspectives on system development using patterns, the first being the concretization of the design reuse of patterns into actual code reuse in the form of *frameworks*, the second being organization of *pattern systems*.

3.3.1 Patterns and Frameworks

The invention of *patterns* is frequently attributed to the architect Christopher Alexander [AIS+77] [Ale79] who realized that in architecture there exist recurring situations that share a problem - solution couple.

This observation proved to be as true in software engineering as it is in architecture, and the ideas of Cristopher Alexander commingled effortlessly

into the object-orientation movement in the eighties. The Smalltalk language incorporated these ideas, for example in its extensive use of the *model-view-controller* pattern [GR83] [KP88]. By providing a supplementary layer of abstraction to the ideas already in swing at that time, pattern technology could be argued to constitute the very pinnacle of object-orientation.

In the nineties the pattern concept had rooted firmly in software engineering and patterns were identified, formalized, classified [GHJV93] [Zim94] and nicely cataloged [GHJV95] [BMR+96].

A *framework* can be described as in [Joh92]: "...a reusable design of a program or a part of a program expressed as a set of classes". For example, the JHotDraw framework, described in section 5.3, provides an object-oriented framework for GUI development in Java for applications involving drawing and graphical languages. It uses several known patterns, such as *model-view-controller*, *composite*, *factory*, *state*, *strategy* and *prototype* (see [GHJV95] for a description of the patterns). The use of patterns in frameworks can be viewed as a concretization of the *design reuse* of patterns to actual *code reuse* by incorporating design concepts in class libraries.

The components of a framework can be partitioned into two categories [Pre95]:

- *Hot spots:* Customizable features, such as presentation of objects on screen in a drawing application framework, that are predicted to vary widely between uses of the framework. Increasing the number of hot spots allows for a more flexible reuse of the framework.

- *Frozen spots:* The fixed features that mark the domain of the framework, such as the model-view-controller pattern of a GUI development framework. Finding the correct frozen spots ensures a more sound reuse of the framework and is pedagogically important for communicating the domain area to application programmers.

Another difficult issue in developing frameworks is balancing *white-box* (inheritance) against *black-box* reuse (composition) [FS97], where, again, the former provides a more flexible reuse while the latter makes the framework simpler to use.

The idea of basing the architecture of a framework on patterns has two clear advantages:

- *Patterns offer effective and flexible reuse of code:* The high abstraction level of patterns enables expeditious development, while selection and specialization of existing patterns provide necessary flexibility.

- *The theory behind the framework is easily accessible:* As argued in [Joh92], documentation in the form of patterns will effortlessly amalgamate the

theory of using the framework with already known concepts from pattern technology.

In 6.5, the design of a workflow monitoring tool, using the JHotDraw framework, is described.

3.3.2 Pattern Systems and System Development

The explosion of work published on patterns during the last decade resulted not only in single patterns and catalogs of patterns for software engineering, but also in thinking in terms of *pattern systems* for solving problems. In [BMR+96] a pattern system is defined as a collection of patterns with descriptions of relations between them. One may regard a pattern system as a language for discourse on domain-specific issues, though pattern systems are, as noted in [BMR+96], rarely *complete*, in the sense of, for example a context-free language.

Examples of pattern systems can be found in various areas, such as software architecture [CS95] [BMR+96], framework development [RJ96] and workflow management [vdABtHK00]. We will here concentrate on patterns for software architecture in the domain of complex information systems, specifically brain imaging meta-analysis systems, but the discussion should be valid for a larger class of applications.

This section presents a selection of architectural patterns and comments on the usage of these for information systems and, more specifically, for building database-centric brain research environments. Experiences from working with system-architectural issues in this field during the projects ECHBD, BINS and NeuroGenerator, related in [Fre99] [FRS99] [FS01] [FSR01] [RSL+01], are characterized below in terms of pattern systems.

In [BMR+96], software engineering patterns are divided into three categories (Fig. 3.5):

- *Architectural patterns:* These patterns affect the overall system architecture, for example the model-view-controller (MVC) pattern in an interactive system. Expresses fundamental structures of whole systems.

- *Design patterns:* Solves recurring problems faced by (application) programmers, such as having one and only one instance of an object, which is solved by the *Singleton* pattern. Design patterns frequently constitute building blocks for architectural patterns.

- *Idioms:* Language-dependent solutions to lower-level problems that application programmers use without bothering about how they are implemented, such as maintaining a count of instances of equivalent objects.

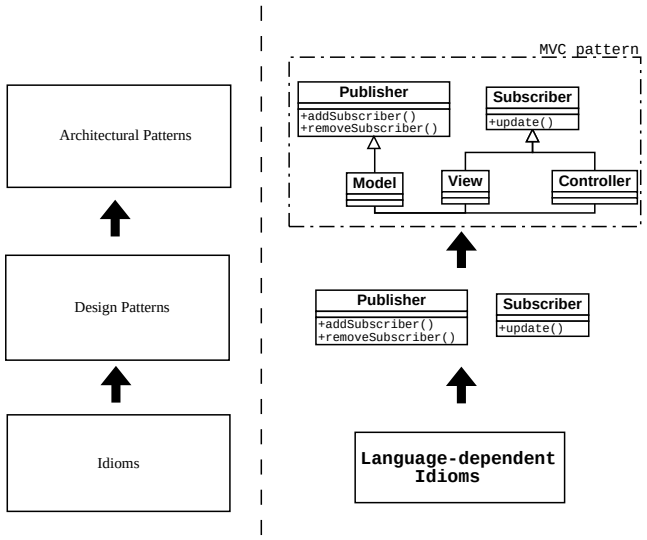


Figure 3.5. The model-view-controller pattern described as a hierarchy of patterns

As can be seen in Fig. 3.5, this hierarchy of patterns help structuring such relatively complex concepts as an MVC interactive system by recursively describing patterns at one level in terms of subordinate patterns. In Fig 3.5 we do not take into account all idiom-level concepts needed for implementation of the design patterns.

Two simple patterns are briefly explained below as a reference for a discussion of development of a *Workflow Management System (WFMS)* (see chapter 4). For more information on particular design patterns, see [GHJV95] or [BMR+96].

- *Composite:* The Composite pattern enables a hierarchical division of a components constituents. Using the object-oriented concept of *overloading*, a composite object can be manipulated in the same way as the simple objects it contains. The operation performed by a composite object is merely a for-loop over the simple objects' operations. In Fig. 3.6 this pattern is described according to [GHJV95], an abstract baseclass *Component* defines the interface common to both subclasses *Simple* and *Composite*. A composite object consists of one or more components, each component either simple or composite. An example usage, later (section 6.3) described more thoroughly, is in the workflow setting. A computational proxy represents a workflow process and can be either simple or composite, i.e. containing other computational proxies.

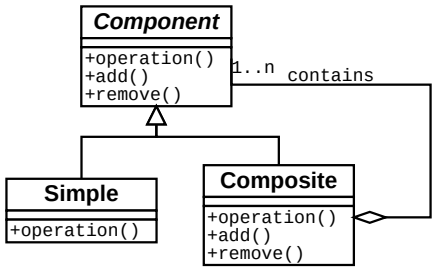


Figure 3.6. The Composite pattern

- *Type-Object:* Type-Object, depicted in Fig. 3.7, is a pattern that enables run-time creation of classes, thus de-coupling instances from types [Joh96]. Each type is an instance which can be created at run-time. This pattern is useful for dynamic and adaptive behaviour of systems. It is used in the workflow management system described in section 6.3 for providing blueprints for types of workflow processes. In combination with the strategy pattern [GHJV95], each type of workflow process may e.g. implement a different scheduling strategy.

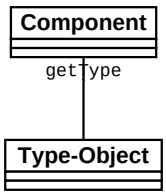


Figure 3.7. The Type-Object pattern

It is probably not meaningful to describe *one* design, in terms of patterns, of a WFMS, but rather discuss designs by the aid of a *system of patterns*, involving for example the Composite and Type-Object patterns. But even so, there are ways to implement a WFMS that cannot be described only through patterns. In section 6, we will present an implementation focusing on reuse of database management concepts such as Event-Condition-Action (ECA) rules. It does however utilize the composite and type-object patterns for modelling workflow processes, so the pattern system is not as clear-cut as in Fig. 3.5.

Similar to Fig. 3.5, a system of patterns/components for development of image meta-analysis systems may be defined as in Fig. 3.8. The three levels depicted in Fig. 3.8 are design patterns, architectural patterns and *components*, of which the latter provides a concretizatization of different subsystems,

frameworks, API:s or systems such as DBMSs. The hierarchical composition of the levels of the pattern system are only partially ordered, as noted above, since the workflow system may be implemented by reuse of DBMS components.

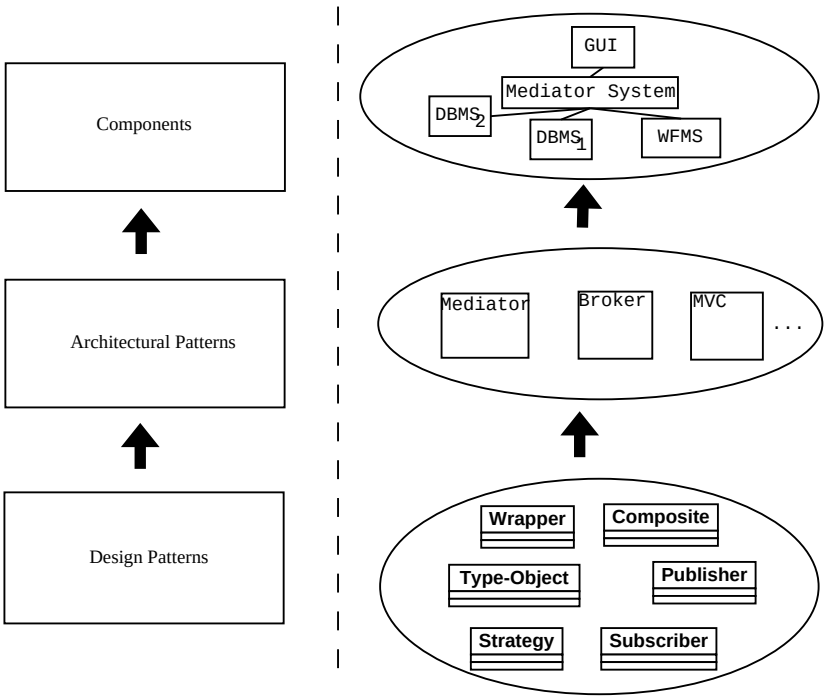


Figure 3.8. A component/pattern system for design of image meta-analysis systems

3.4 An Evolutionary Process for Development of Brain Imaging Meta-analysis Systems

The functional system architecture depicted in Fig. 3.9 can be seen as a requirements specification of the system, from which some initial requirements can be identified as:

- *Raw data storage facility for PET and fMRI experiment data*
- *Processing facility to transform raw data into statistical data*
- *An environment for meta-analysis of experiment data*

With this picture in mind, and the background to brain imaging research from chapter 2, how do we go about building this system?

In this section, we will present a process model sketch applicable to this problem instance. The task of completing the process model to a more general setting will be left as future work.

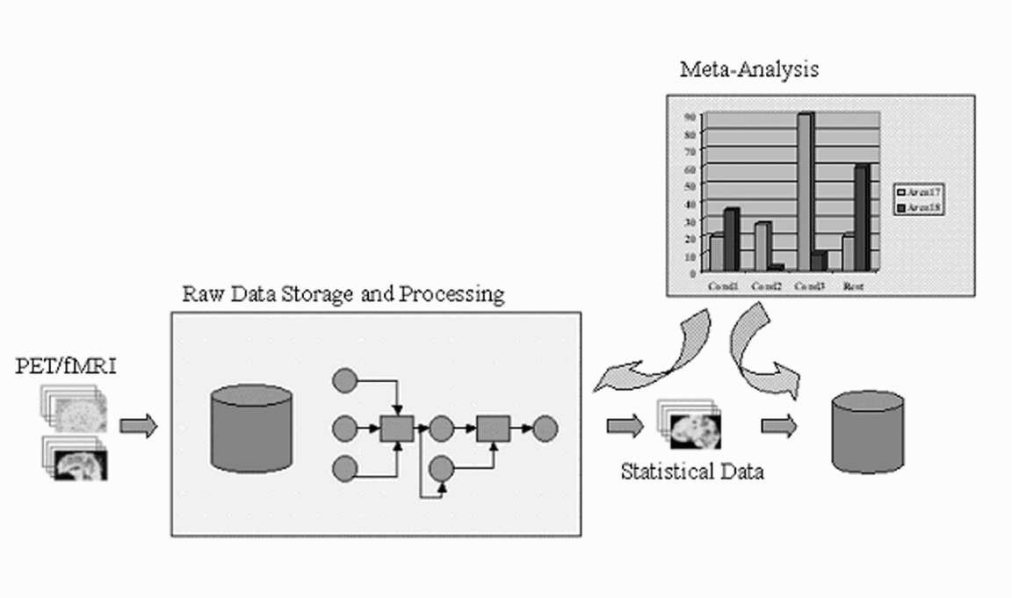


Figure 3.9. The functional system architecture

In this process, we first define a set of pertinent use-cases [JCJv92], successively more demanding in terms of functionality and performance of the system. A subsequent analysis phase is used to identify reuse of three different kinds:

- *Subsystems identification*
- *Database extensions*
- *Architectural patterns*

The database extensions are divided into:

- *explorative level:* some model extension aspects may be explored and resolved entirely on the query language level [OR96]
- *functional level:* by addition of specialized ADTs for specific domains (e g image data, spatial data, matrix data) the core database model may be extended to new application domains, such as Finite Element Analysis [OR96] and Spatial Analysis [Ouk01].

- *performance level:* when the domain model has become linguistically and functionally well integrated, efficient access paths in the form of domain specific indices and optimization rules may be introduced to obtain scalable query evaluation performance [SM95].

In the process we also develop a design on a lower level, in terms of:

- *Schema development*
- *Database partitioning*

The result of each analysis phase is a set of design artifacts that can be transformed into an architecture. By prototyping these architectures, changes will be inferred upon the original sequence of architectures. Also, a structural analysis of the evolutionary path is necessary, to ensure that the sequence of resulting architectural designs is "sound", i.e. not overly complex for the use-cases defined.

This approach may be regarded as a modification of the evolutionary development process for especially large and complex systems. Because the system is so complex, we desire an incremental transfer of knowledge from end-users to system designers. This is incurred by the use-cases and the prototyping and analysis phases.

Another important aspect to consider when choosing a development process is the organizational structure of the project. In [Cop99] it is noted that "The structure of the organization that builds the software is homomorphic to the structure of the software". We argue that the combined reuse-oriented evolutionary process is ideally suited for a project team consisting of loosely co-operating sub-teams, due to the loose coupling of software modules. In the present case, several field specialists with different personal goals (such as writing doctoral theses) work together to compose a system for brain imaging meta-analysis.

Note that this introduction contains a *meta-modelling* phase of the process model, i.e. the definition of what reuse and design artifacts to focus on. For another class of systems, we would probably define different kinds of reuse and design artifacts. The complete process model is depicted as a Petri Net (see section 4.4) in Fig. 3.10.

3.4.1 Use-case-based Evolutionary Architectural Analysis

By modelling typical use cases in steps satisfying increasingly higher demands on performance, scalability and system expressiveness, an evolution path is established for the system.

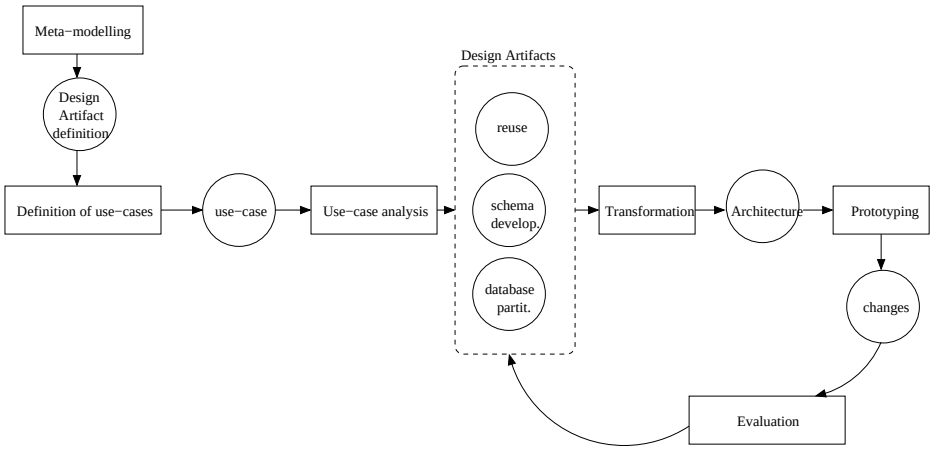


Figure 3.10. The process model depicted as a Petri Net

The use cases can be found in Fig. 3.11 and the corresponding architectural solutions for each step are described in Fig. 3.12. The labels KI and PDC below denote the neuroscientific research laboratory at the Karolinska Institute and the supercomputing center at the Royal Institute of Technology, respectively, located a few kilometers apart and connected by a high-speed data link.

(a) *Local execution of one processing module:* The privileged user browses through databases and selects a process to execute, and its indata

Analysis:

- define basic databases: ExperimentDatabase and ProgramDatabase
- Explorative Level extensibility: Query language provides at least preliminary user interface
- Functional Level extensibility: Extend DBMS with functions for:
  (i) invoking programs on indata images
  (ii) notifying the user when program terminates
- *Schema development:* Define base classes *Provider*, *Experiment*, *Subject*, *Condition*, *Repetition* and *Image* for data, and *PCModule* for programs (or, processing chain modules).

(b) *Whole processing chain execution:* The privileged user designs a processing chain from raw data to some refined format, which is then executed and stored in the database.

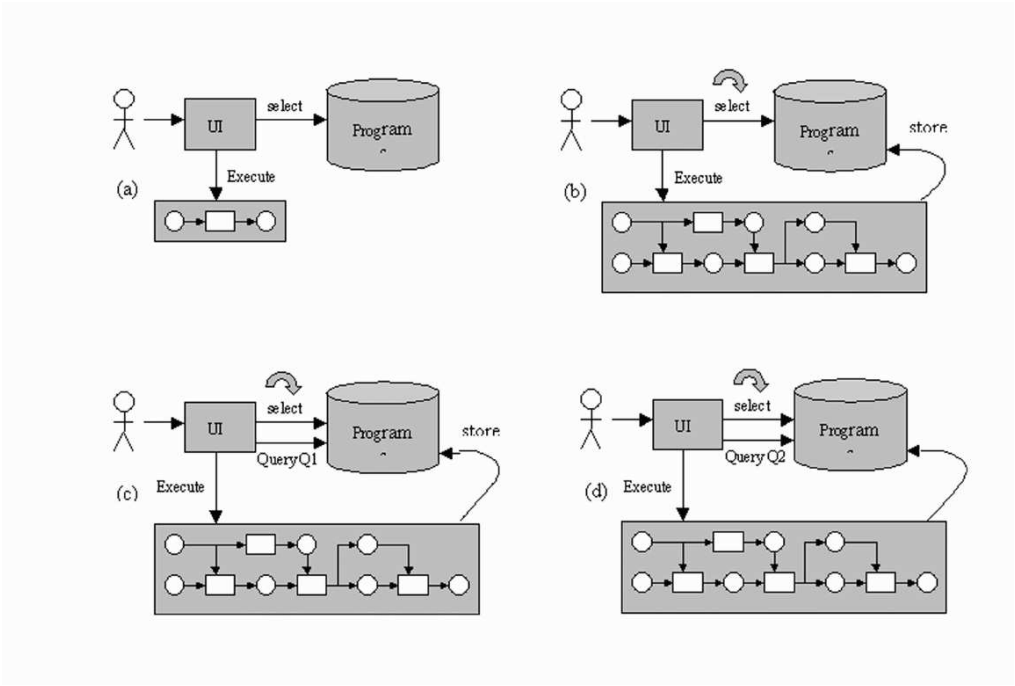


Figure 3.11. A set of increasingly more demanding use cases

Analysis:

- *Functional Level extensibility:* Complex processing creates a requirement for workflow management, which is kept internal to the AMOS II DBMS by use of Event-Condition-Action (ECA) rules. A *computational proxy* [C+94b] is introduced as a new datatype with foreign functions to manipulate execution of its corresponding PCModule. A visual language is desirable to aid users while defining workflows.
- *Databases to be defined:*
  - WorkflowDatabase for workflow data
  - TemporaryDatabase for temporary image objects
  - HomogeneousDatabase for end results
- *Schema development:* Add classes *Analyst*, *Workflow*, *CompProxy* and *DataSource* for workflow management.

(c) *Same as (b), but processing at PDC node is monitored from KI node:* Example query, issued by the user to the HomogeneousDatabase:

Q1 := "Find all images in which the subimage corresponding to the [x0:x1,y0:y1,z0:z1] interval has a voxel value greater than x".

Analysis:

- the shared supercomputing environment with vastly increased computational power and storage capacity needs to be remotely monitored and controlled through the EASY scheduling system.
- apply the mediator design pattern (here: multidatabase nodes KI and PDC)
- *Functional Level extensibility:* define type extensions using foreign functions to perform simple raster data queries.
- schema development: the database schema needs to be mirrored in a relational DB2 database for persistent storage. An ODBC translator provides mediated access to this data. Subimage relations are modelled by the function subimage between *VoxelSet* and *Image*.

(d) Same as (c) but with larger database and more demanding meta-analysis queries Example query, issued by the user to the HOMOGENEOUSDATABASE:

Query Q2 := " Find the number of statistically significant clusters, for each voxel, that contains this voxel".

Analysis:

- add raster data package with secondary memory storage, special storage structures and access paths for better scaling properties, and data compression for faster data transfer; either by adding this to the previously defined (in c) Performance Level extension or by providing mediated access to a separate specialized DBMS
- schema development: define subclass to VoxelSet, *ConnComp*, to be able to represent an arbitrarily shaped region of the brain

3.4.2 Architectural Consequences

A set of architectures resulting from the analyses outlined above is summarized in Fig. 3.12. Schema development is summarized in Figs. 6.2 and 6.3. An issue suggested by these architectures is whether three different DBMSs are really needed, or in general, how can we be sure that the evolutionary design process leads to a sound architecture? Like in biology, evolutionary software

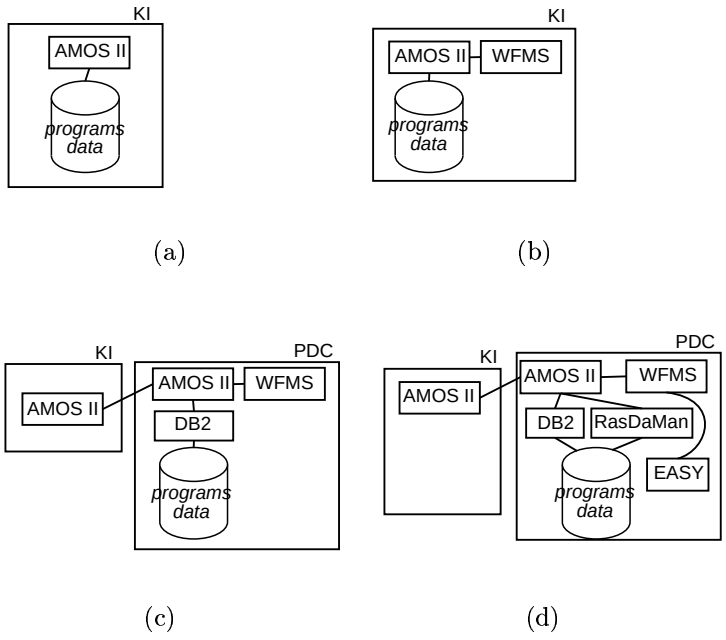


Figure 3.12. A sequence of architectures

design is usually expressed as modification of existing structures, while more thorough redesign is rare.

One might consider two alternative designs, using for database management

- either a single relational DBMS (DB2)
- or a specialized raster data DBMS (RasDaMan) and a relational DBMS (DB2) for metadata

The first of these alternatives is feasible, but content-based retrieval (e.g. query Q1) on this large amount of image data will presumably not be scalable. The second alternative, however, does not provide a common query language thus putting the demanding task of data integration on the application programmer, as in [Fre99] [FRS99].

Thus, the best solution depends largely on the size and complexity of the databases. The BINS project stops at level (c) of Fig. 3.12, while NeuroGenerator builds on results from the previous project and promises more effective solutions on several levels [RSL+01]. In order to realize these solutions, NeuroGenerator has been set up to provide the expertise necessary for managing the resulting design complexity. The label EASY used in Fig. 3.12 denotes the supercomputer job scheduler interface at PDC.

BINS/NG databases may be partitioned into the following categories:

- ExperimentDatabase contains a catalog of experiments, subjects, raw image data, and image metadata.
- ProgramDatabase is a set of (mainly image) processing modules subject to software version control. Programs are stored at the PDC node in a DB2 database as BLOBs.
- WorkflowDatabase manages the workflow control data as shown in Fig. 6.3 Intermediate data are stored in the file system.
- HomogeneousDatabase contains the processed database, consisting of processed images, their pedigrees and other metadata
- TemporaryDatabase is used for view materialization of sub-workflows, optimization of subsequent analyses, and sample tests of process output quality. It is also in itself the target for meta-analysis.

Chapter 4

Workflow Management for Brain Image Meta-analysis

In section 2.3.2, a typical so-called processing chain for statistical analysis of PET image data was presented. Given that the example is a standard procedure for analyzing only *one* PET experiment[1], and still involves some 500 raw, intermediate and end-result images, we will most likely need a tool to manage image data *and* processes on image data.

We will in this chapter introduce the concepts of *workflow management*, WFM, and *workflow management systems*, WFMS, and argue that a WFMS is well suited as a tool to manage processing chains.

In [GHS95], a workflow is defined as "a collection of tasks organized to accomplish some business process". That the restriction to business processes is unnecessary is exemplified in [A+98], taken from soil sciences. Workflow management, WFM, is then the management of this collection of tasks and data. A brief overview of what a system for workflow management, a WFMS, can offer is provided in section 4.1, followed by discussions on how to represent and specify processes (section 4.2), organize transactions (section 4.3) and analyze properties such as *correctness* of workflows (4.4) and also briefly note some special requirements of *scientific* workflows in section 4.4.3

4.1 Motivation

The processing chain in Fig. 2.7 is depicted as a directed bipartite graph, consisting of *data*, *processes on data* and *relations* in the form of input and output data from processes to data.

[1] fMRI experiments are approximately an order of magnitude larger in size

We are going to call the programs implementing processes on data *processing modules*, while the process instances will be referred to as *computational proxies* [C+94b].

For executing a processing chain on a single experiment instance, some Unix shell scripts were developed (see section 6.1), purposefully encapsulating existing software for a processing module, i.e. a single step of the processing chain. The user starts a shell script, waits until it has terminated, and starts the next script in the processing chain. These shell scripts are simple to start and are automatic in the sense that the user does not have to interactively drive the processing of one script, but there are still a number of difficulties for the user to face:

- *No automatic parallelization:* The processing chain is in many cases *trivially parallelizable*, meaning that the execution of one script on N brain images can be performed in parallel on N different nodes, independently of each other. This distribution of tasks on nodes should be taken care of by the system.

- *Keeping track of the order of processing is difficult:* One PET experiment may involve running six or more processing modules hundreds of times in a causally determined order: each processing module needs its input data to have been generated before it can start. The user has to keep track of this order.

- *No automatic error handling:* The scripts implementing processing modules can leave a log-file as a trace of its activities, but if something goes wrong, the user will have to keep track of how to undo the compromised processing and where to re-start processing.

- *Reuse of previous processing chains and intermediary data is difficult:* The user will have to manually keep track of which of previously calculated intermediary data or processing chain structures that can be reused.

- *Monitoring the status of the processing chain is difficult:* To know what steps of the processing chain have been completed and estimate when the whole chain will be completed is, without proper tools, potentially complicated.

The Workflow Management System (WFMS) is implemented in order to automate these aspects of running the processing chains. Basing the architecture of the WFMS on a DBMS has the additional advantage of allowing

querying of all data regarding the status of a processing chain, previously executed processing chains, and so on.

To meet these and other demands on functionality, a WFMS supports some, or all of the following concepts:

- *Automated processing:* An important goal in the NeuroGenerator/BINS projects has been the automatization of processing chains. Thus, each participating processing module has been reengineered to run *unsupervised* [Hal]. For each execution of a processing module, the WFMS will check that the previous executions of processing modules on which this execution depends, are finished - and commence execution. The constraints this puts on the system is implemented and checked by so-called *active rules*, see section 6.3.1.

- *Traceability:* When saving the workflow needed to create an object, its history, or *lineage*, can be traced back to the data and processes that created it, which is especially valuable in scientific workflow management systems, see section 4.4.3.

- *Workflow specification in a graphical language:* To specify dependencies, and thus the order of execution of processing chains, the WFMS will need a language for describing the dependencies and selecting the processing modules and data involved. A natural description to use for this purpose is a form of a *graph*, see section 4.4. To depict this description *graphically* provides intuitive user interaction, see section 6.5.

- *Concurrency and parallelization management:* When executing a processing chain in a parallel computer system, as described in chapter 6, we want to be able to run several processing modules that are not dependent on each other *concurrently*, distributed on the parallel nodes of the system. By specifying dependencies in the workflow, for example by the use of Petri Nets, as in section 4.4, all processing that is not inter-dependent can be performed concurrently. The implementation of the concurrent processing is described in section 6.4.

- *Workflow analysis tools:* When specifying a processing chain, the user is responsible for constructing it in a *correct* way. Some of these correctness criteria can be formalized and automated, as described in section 4.4.

- *Transactional model:* Execution of processing modules may sometimes fail, for reasons such as software bugs or hardware failures. When this occurs, we need a mechanism to take action accordingly, for example to abort execution, erase any partial results that are not guaranteed to be uncompromised, and rerun aborted executions.

- *Different processing constructs:* In [vdABtHK00] several processing constructs are discussed. The processing chains of BINS/NeuroGenerator contain to date not very advanced constructs, for example there is no iterative processing. However, such constructs may become of interest later, for example in a hierarchical processing chain, where workflow management totally replaces scripting languages. Thus, the use of the *Composite* pattern in modelling workflows in section 6.3 seems natural, and furthermore, it is important to be able to extend the workflow model with new processing constructs, such as iteration.

- *Monitoring of processing:* The current status of processing should be possible to query in either textual or graphical form. This is currently possible through the AMOSQL interface described in section 6.4.1, but will benefit from interfacing the WFServer to the GUI described in section 6.5.

- *Reuse of processing structures and intermediate data:* When designing processing chains similar in structure to old processing chains, it should be possible to reuse previously saved structures and edit them. Further, previously calculated intermediate data that costs several hours of processing to recalculate should be possible to reuse, when appropriate.

- *Dynamic (run-time) modification of processing:* It may prove to be useful to dynamically be able to change the course of action in a running processing chain.

- *Adapting to change:* It has been recognized [vdA99] that a serious shortcoming in many contemporary workflow systems is in adapting to change in the form of *ad-hoc change* and *evolutionary change* of workflow processes.

4.2 Modelling Image Processing

In this section we deal with the definition of a *workflow model*, for structuring image processing in an image meta-analysis environment, using for example an object-oriented model or a database schema.

In reviewing commercial WFMSs, Georgakopoulos and Hornick [GHS95] describe a *workflow model* as consisting of the following entities:

- *Workflows:* A partial or total ordering of tasks.

- *Tasks:* A partial or total ordering of operations, descriptions for human actions or other tasks. In our system, we do not yet take into account

human actions, but focus on automatically executing steps of the processing chain. Note the recursive definition in that a task may consist of tasks.

- *Manipulated objects:* In the case of image processing, the majority of manipulated objects are images.
- *Roles:* A human skill or a service provided by some system that is required to perform a task.
- *Agents:* Humans or systems enacting roles.

In [KS94], the related concept of *workflow specification* is described as:

- *Task structure:* A description of the structure for carrying out a task, such as a two-phase commit (2PC) transaction, common in database systems, or a non-transactional structure. We discuss task structures further in section 4.3.
- *Typed inputs and outputs of task types.*
- *Relations between input and output among task types:* Output of one task may provide input data to another task.
- *Preconditions for transitions occuring within one task:* Together with relations among input and output, preconditions specify the ordering of tasks, ie. a task can be performed when its preconditions are satisfied, which may involve the production of another tasks output.

In section 2.3.2 we defined a basic processing chain that transforms raw PET image data into statistical cluster images. Many possible processing chains exist, and in view of the previous definitions of workflow models and workflow specifications, we introduce the following concepts for managing processing chains:

- *Processing module:* A program implementing one step of the processing chain.
- *Computational proxy:* The term is borrowed from [C+94b] and reflects the task carried out by a processing module.
- *Data in initial, intermediary and final form:* Input and output data from processing modules.
- *Relations between processing modules and input and output data:* Two mappings from the set of processing modules to the set of data: *input data* and *output data*.

- *Resource:* Actors involved in running the processing chain tend to be non-human, such as the scheduler of the parallel computer system.

- *Transactional structure:* The course of action to take, for example when an operation performed in a task fails.

When structuring these concepts, we may for example choose to employ either an object-oriented model [A+98] [C+94b] [Man01], or a more formal approach based on *Petri Nets* [vdA94] [vdA98]. Of the object-oriented approaches mentioned above, [C+94b] seems to use the most traditional approach, based on a object persistency environment, while [A+98] uses a clever transformation of some workflow constructs onto the meta-model, the *schema*, of the underlying DBMS and argues that implementation efforts may be reduced by reuse of database concepts.

The Object Management Group (OMG) specifies a WFMS reference architecture [OMG02], critized in [MJ98] for not using the descriptive power of patterns. It was argued that the object-oriented modelling suffered from this, for example in the failure to recognize the ability to use a Type-Object pattern [Joh96]. This line of thought led to the introduction of an extensible WFMS *MicroWorkflow* [Man01], building on the aforementioned design patterns and the architectural pattern *Microkernel* [BMR+96] for adaptive systems. In [Man01], the core functionality of the WFMS is obtained by a combination of design patterns such as Type-Object and Composite with a special pattern language, similar to [vdABtHK00].

In comparison to the OMG reference architecture, Petri Net based models focuses on analysis of workflow, outlined in section 4.4. The strengths of a more formal theory as described in [vdA94] are mainly the unambiguity of workflow specifications and a well-defined theory for analysis and simulation.

In section 6.3, we describe an implementation of a workflow model using object-oriented concepts, long-running transactions (see section 4.3.2) that also resembles a Petri Net model in its graph structure. The analysis techniques of Petri Net models may be used through an external GUI, as described in section 6.5.

A third category of approaches to workflow management is represented in [KS95], where the focus of modelling processes is on a strong *transactional model*, a concept which will be dealt with in the next section.

4.3 Transactional Models

When executing a processing chain, a constituent task may for some reason fail (due to program malfunction, insufficient disk-space, etc.). To handle this and other possible failures in processing, we need to specify appropriate courses of action to take before starting the execution. These courses of action were in section 4.2 referred to as the *task structure*. When specifying task structures, we may be satisfied with a non-transactional structure, or employ models of various degrees of expressibility for the transactional structure. An excellent overview of advanced transactional models can be found in [Elm92].

4.3.1 Transaction Management

In short, the classic ACID[2] properties of ordinary (short-lived) database transactions ensure *execution* and *failure* atomicity.

- *Execution Atomicity:* To maintain execution atomicity of database transactions involves the isolation of transactions and the consistency of the database. Other transactions cannot read intermediate results from a transaction that has not yet committed, because using these intermediate results may produce inconsistency in the database.

- *Failure Atomicity:* Failure atomicity is often desired when dealing with aborted transactions. Either a transaction is fully committed, or not at all. If partial transactions are allowed, then again, we risk the consistency of the database.

To maintain execution and failure atomicity through a transaction guarantees a sufficiently consistent behaviour for many simple transaction types, such as ATM applications. The advantages are rather obvious, the system gurantees that either the whole transaction is completed, or the whole transaction is aborted, with no partial results or pending inconsistencies in the database.

In the case of image processing as described in section 2.3.2 and depicted in the example of Fig. 4.1, this may be implemented as a traditional transaction for each task (depicted as a box in Fig. 4.1). Consider the example of a software bug, rendering output data from a image processing program to be zero-length. Assuming that we can catch the software failure[3], we may remove the compromised results and later try to redo the transaction.

[2] Atomicity, Consistency, Isolation and Durability

[3] e.g. by reading the exit signal from the process

However, this is rather inefficient, since the traditional transactional model thus employed will not operate concurrently on independent transactions, thus, the obvious parallelism present in Fig. 4.1 would not be exploited.

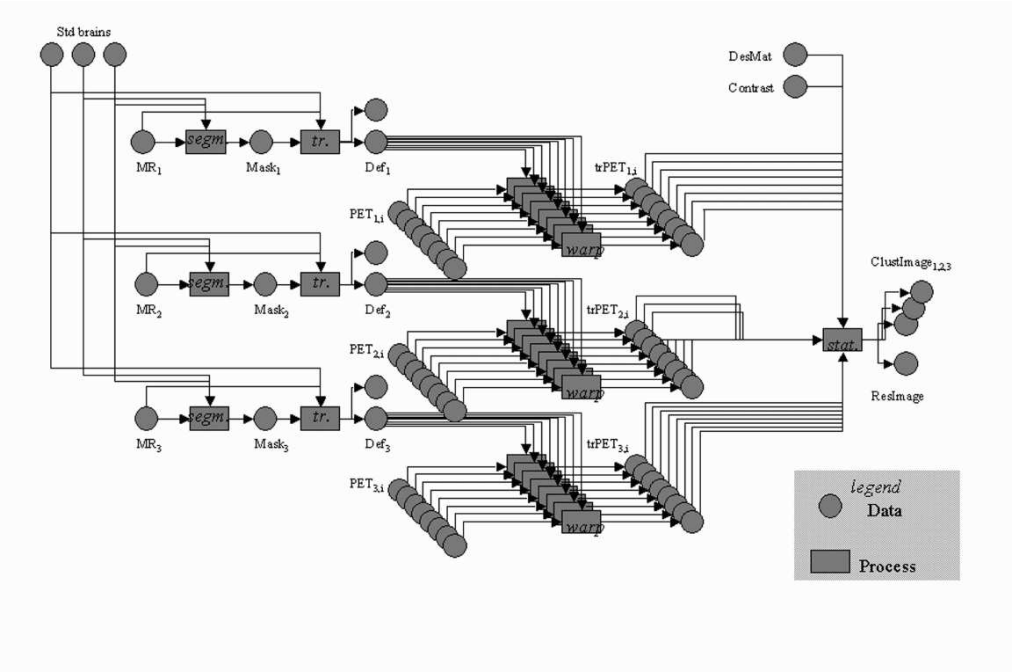


Figure 4.1. Transactional view of workflow. The processing chain is executed on an experiment consisting of three subjects, two conditions and four repetitions.

4.3.2 Long-running Transactions

One of the preconditions for the transactional model described in section 4.3.1 to work well is that the transactions are quick, usually on the order of seconds, maximally. Another precondition is that the task structure can be described by a flat transactional structure. For applications such as image processing, transactions may involve rather computation-demanding operations. PET image processing in the BINS and NeuroGenerator systems are initially based on the processing modules of table 4.1, from [Hal]. Processing ranges from seconds to several hours and, as can be seen from Fig. 4.1, transactions could benefit from some additional structure.

In a *Saga* transactional model [GMS87], forward transactions $\tau_1, \tau_2, \ldots, \tau_m$ are coupled with compensating transactions $\sigma_1, \sigma_2, \ldots, \sigma_m$ so that it is ensured that either the sequence $\tau_1, \tau_2, \ldots, \tau_m$ or $\tau_1, \tau_2, \ldots, \tau_k, \sigma_k, \sigma_{k-1}, \ldots, \sigma_2, \sigma_1$,for

proc. module	duration	input data	output data	parameters
imagemodel	7 hours	set of PET images	cluster image	
		design matrix	t-image	
		contrast vector	σ-image	
debone	45 mins	anat. MR image	mask image	
		segm. std. brain		
		unsegm. std. brain		
air_trans	5 mins	anat. MR image	def. field	
		mask image		
		segm. std. brain		
air_warp	30 secs	PET image	tr. PET image	
		def. field		
		mask image		
im_mask	30 secs	set of PET images	null-val. mask	
subsample	10 secs	PET/MR image	sub. PET/MR	factor
convert	10 secs	Raw scanner image	internal format	

Table 4.1. Summary of processing modules

some k such that $1 \leq k \leq m$ are performed. Further, it is possible to organize sagas hierarchically. A saga may thus contain a *sub-saga*, and recursively a sub-saga may consist of further sub-sagas.

This structuring tool may thus help us managing the relatively complex processing present in the PET processing chain. One may discern three levels of processing:

- *Conceptual level:* This level corresponds to the conceptual description of processing, e.g. the *transformation* to standard anatomical format and the *statistical analysis* in Fig. 2.7.
- *Process level:* The process level corresponds to the processing modules of Fig. 2.7, where each module has a well-specified task to accomplish.
- *Algorithmical level:* Each processing module in Fig. 2.7 and table 4.1 consist in reality of scripted calls to up to 40 different specialized programs. Instead of managing the processing modules, we may detail transactional structure to the level of single program executions, thereby allowing, among other things, smaller atoms of transaction.

The compensating transactions for each forward transaction depicted in Fig. 4.1 are thus capable of undoing the forward transactions, for example

clearing a working directory. If a sub-saga is aborted and its compensations are executed, the parent saga may use *forward recovery*, if possible, to execute its other constituent sub-sagas. If forward recovery is not possible, due to transaction dependence on the aborted sub-saga, *backward recovery* takes place through execution of compensations.

For saga transaction management to be applicable, the task structure should contain largely independent sub-transactions and forward transactions should be compensatable. If sub-transactions are dependent on each other, there is no guarantee that they will be executed in the proper order. A possible saga structure for the example of Fig. 4.1 can be found in section 6.3.

Further extensions of the transactional model can be found in [Elm92] or [KS94] [KS95] [DHL90].

4.4 Petri Net based Analysis of Workflows

In [vdA98] a construct called *WF Net* (or, *Workflow Net*) is constructed from a *Petri Net*. We will follow this approach in section 4.4.2, but start out with the appropriate definitions from graph theory below.

4.4.1 Some Elementary Definitions from Graph Theory

A *graph* $G = (V, E)$ is a set of *vertices*, $V = \{v_1, v_2, \ldots, v_m\}$, and *edges*, $E = \{e_1, e_2, \ldots, e_n\}$, between vertices ($e_i = (v_j \leftrightarrow v_k)$). Such a graph is called *undirected*.

If the graph is *directed*, all edges are associated with a direction, i.e. $e_i = (v_j \rightarrow v_k)$,or $e_i = (v_j \leftarrow v_k)$.

A *path* in a graph is a sequence of vertices $\{v_1, v_2, \ldots, v_m\}$ such that each adjacent pair of vertices in the list are connected by an edge. A *cycle* in a graph is a path $\{v_1, v_2, \ldots, v_m\}$, where $v_m = v_1$, i.e. the path leads back to the first node of the sequence.

Since cycles in graphs often provide extra difficulties in theory and algorithms on graphs, an important concept is the *Directed Acyclic Graph*, DAG. This graph has directed edges and no cycles.

Another common concept, or class of graphs, is the *bipartite graph*. A graph is bipartite if its vertices V can be partitioned into two disjoint subsets V_1 and V_2 such that each edge in E connects a vertex from V_1 to one from V_2 and $V = V_1 \cup V_2$. Examples of the four graphs mentioned so far can be found in Fig. 4.2.

A final useful definition is that of a *strongly connected graph*, in which for a directed graph $G = (V, E)$, each pair of vertices $\{v_i, v_j\}$ has a path from v_i to v_j.

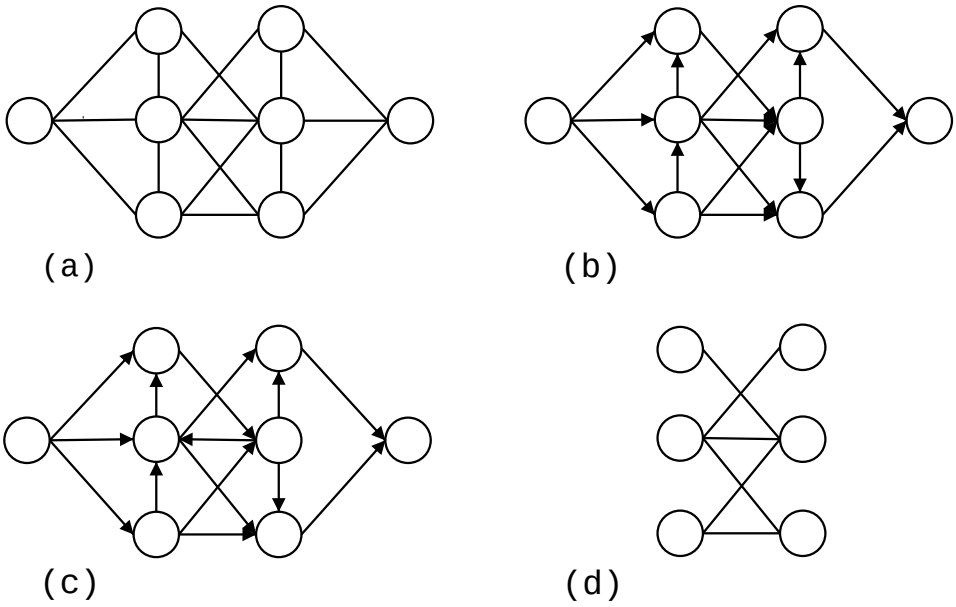


Figure 4.2. (a) Simple undirected graph (b) directed graph (c) directed graph with a cycle (d) bipartite graph

4.4.2 Modelling Workflows as Petri Nets

When modelling a workflow, we need to take into account *processes*, input and output *data*, *states* and *transitions* between states. Obviously, a static graph structure is not sufficient so we extend the graph definition to include states and transitions. As a first step we consider so-called *Petri Nets*.

The theory of Petri Nets is rich and diverse with numerous extensions, since its powerful modelling capabilities have made it a popular tool in many applications. We will only scratch the surface of this theory to provide some more substance to the problem at hand, namely the modelling of processes and workflows of processes. There are interesting results that may be applied to the workflow in BINS, such as verifying "soundness" of workflows and with the proper extension of time, perhaps optimization of execution can be addressed in this general framework. We will in this section follow [vdA98] for obtaining a means to model and analyze workflows for image meta-analysis by the use of Petri Nets.

Definition 4.1. A Petri Net PN is the tuple (P, T, F), where P denotes a set of vertices, or in Petri Net terminology, *places* and T is another set of vertices, called *transitions*. Further, $P \cap T = \emptyset$, and the edges are defined as $F \subseteq (P \times T) \cup (T \times P)$ and $\forall t \in T$, $\exists p \in P$: $(p, t) \in F$.

Petri Nets are commonly visualized using circles for places and squares for transitions with arcs between them in the form of a bipartite graph.

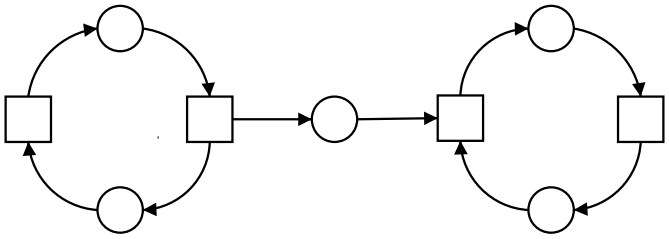


Figure 4.3. An example Petri Net

So far, this is just a directed bipartite graph, where P and T define the two partitions of vertices and F are the directed edges between the partitions. Let us add some more definitions.

Definition 4.2. A *preset* $\bullet v$ of a vertex $v \in P \cup T$ is the set of vertices $\{v' | (v', v) \in F\}$ and the *postset* $v\bullet$ of v is similarily defined as the set $\{v' | (v, v') \in F\}$

We are now ready to define the entity *WorkFlow Net*, or *WF Net*.

Definition 4.3. A Petri Net PN is a WF Net if and only if:

- PN has an input place $i \in P$ for which $\bullet i = \emptyset$ and an output place $o \in P$ for which $o\bullet = \emptyset$
- For PN' $= (P, T \cup t, F')$, with $F' = F \cup \{(o, t), (t, i)\}$, PN' is strongly connected. We will refer to PN' as the *extended WF Net*.

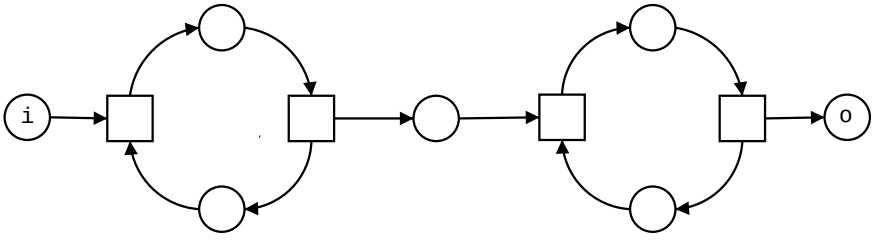


Figure 4.4. A WorkFlow Net

Definition 4.4. We define a *state*, M, by adding zero or more *tokens* to each place $p \in P$. Thus, M is a function:

$$M : P \to \mathbb{N}$$

with the partial ordering:

$$M_1 \leq M_2 \Leftrightarrow M_1(p) \leq M_2(p) \ \ \forall p \in P$$

Let us further shorten the notation for two special cases: We define the initial state I so that $M(i) = 1 \leftrightarrow M = I$ and similarly for the end state O: $M(o) = 1 \leftrightarrow M = O$

The tokens change during execution according to a firing rule:

Definition 4.5. An *enabled* transition may fire according to:

- *Enabled transition:* A transition t is *enabled* iff each input place p of t contains at least one token, $M(p) \geq 1 \ \ \forall p \in \bullet t$
- *Firing transition:* When an enabled transition t *fires*, t consumes a token from each input place of t and produces a token in each output place of t, $M(p) = M(p) - 1 \ \ \forall p \in \bullet t$ and $M(q) = M(q) - 1 \ \ \forall q \in t\bullet$.

Some notation: given a Petri Net (P, T, F) and states $M_1, M_2, \ldots M_n$:

- $M_1 \xrightarrow{t} M_2$: Transition t is enabled in state M_1 and fires, producing state M_2.
- $M_1 \to M_2$: There exists a transition t such that $M_1 \xrightarrow{t} M_2$
- $M_1 \xrightarrow{\tau} M_n$: The sequence of transitions $\tau = t_1 t_2 \ldots t_{n-1}$ results in

$$M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} M_n$$

- $M_1 \xrightarrow{*} M_n$: There exists some $\tau = t_1 t_2 \ldots t_{n-1}$ for which $M_1 \xrightarrow{\tau} M_n$. We say that M_n is *reachable* from M_1.

Further definitions of interest are *liveness* and *boundedness* of Petri Nets:

Definition 4.6. The liveness and boundedness of a Petri Net are important concepts concerning the dynamics of the net:

- *Live:* A Petri Net PN with initial state M_0 is live iff for every reachable state M_i and every transition t, there is a reachable state M_j enabling transition t.

- *Bounded:* A Petri Net PN with initial state M_0 is bounded iff $\exists n \in \mathbb{N}$ such that $\forall p \in P$ and all reachable states M : $M(p) \leq n$.

As a simple illustration of a firing sequence, consider Fig. 4.5, where *segmentation*, *transformation* and *warping* of a functional image (cf. Fig. 4.1) is depicted as the sequence of states $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} M_3$. For a description of the graphical presentation format, we refer to section 6.5.

When considering parallel tasks, for example by adding another PET image to the transformation, we have to keep track of which tokens are "compatible" with other tokens. If MR images mr_1 and mr_2 have been segmented, the resulting mask image mask_1 can only be used for transformation of mr_1. The solution is to add a *color* to each token in order to avoid mixing entities. Aalst indicates how to do this in [vdA98]. Another solution is to model that scenario as in Fig. 4.1, where all transitions are duplicated for each separate processing. This can be compared with the terms *processing module* and *computational proxy*, as introduced in section 4.1.
In [vdA98] it is stated that there are three types of analysis, using Petri Net models of workflow:

- *Validation:* By interactive simulation testing whether the workflow will behave as expected.

- *Verification:* Assessing the *correctness* of the workflow. For example if the number of tokens are mismatched, or if transitions are misplaced, tokens may get lost in the Petri Net.

- *Performance analysis:* We will in chapter 6 look at the interfacing of the workflow system to the supercomputer scheduler. To find an efficient scheduling strategy for workflows, it may be of interest to add the notion of time to Petri Nets [Bow96] and do either simulations or formal analysis.

As an example of correctness verification, we will define the *soundness property* for Workflow Nets.

Definition 4.7. A WF-net $PN = (P, T, F)$ is sound if and only if:

- *For each state M, reachable from I, there exists a sequence of transitions leading from M to O:*

$$\forall M : I \xrightarrow{*} M \Rightarrow M \xrightarrow{*} O$$

- *State O is the only state reachable from state I with at least one token in O:*

$$\forall M : I \xrightarrow{*} M \wedge M \geq O \Rightarrow M = O$$

- *There are no dead transitions in PN:*

$$\forall t \in T \quad \exists M, M' : \; I \xrightarrow{*} M \Rightarrow M \xrightarrow{t} M'$$

Theorem 4.8. *A WF-net PN is sound iff the extended WF Net is live and bounded.*

Proof. Can be found in [vdA97]. □

It would be of interest to find an algorithm to check the soundness property for any possible workflow. It turns out that if we restrict ourselves to so-called *free-choice* Petri Nets, then there are algorithms [vdA97] to check soundness in polynomial time.

In Fig. 4.6 we see an example of how the concept of soundness may help in verification of workflows. Both workflows fulfill the static criteria of both number and types of input and output. The Petri Net of Fig. 4.6 (b) might be due to a point-and-click error during interactive specification.

4.4.3 Meta-analysis Workflows

The management of workflows in scientific processing and meta-analysis has not been discussed much. Using the ZOO framework [ILGP96], an object-oriented workflow management system was described in [A+98], with focus on reuse of database concepts in workflow managment. In [C+94b] the concept of computational proxy was introduced.

The scientific use of workflow technology poses special challenges not present in business applications, which is the usual area of application for workflow systems. Examples of challenges especially interesting for the BINS/NeuroGenerator project are:

- *Data mining and knowledge discovery:* It is clearly possible to automatically generate a large number of analyses in the linear model of section 2.3.1 to find untested hypotheses that may yield new results.

- *Lineage of results:* To be able to trace how an image was generated is very important. For example a too coarse subsampling during the processing chain will seriously impair the quality of end-results, something that can be traced afterwards in a debug process where suspicious results can be checked by inspection of the lineage.

- *Efficiency in processing:* If we have already computed a processing chain in a previous workflow, that is partially contained in a new workflow, we may save time by reusing results from the previous workflow.

As an example of reuse of previous workflows, consider the layout of the processing chain in Fig. 2.7 and assume that we have recently performed an analysis involving a set of subjects $S_1 = \{s_1, s_2, \ldots, s_n\}$ and the associated n anatomical and ncr functional images, c being the number of conditions applied, and r the number of repetitions. We then test a hypothesis in terms of the elements of a contrast vector v_1, using a design matrix X_1 as described in section 2.3.1.

We now wish to test a new hypothesis, given another contrast c_2, from a set of subjects S_2, containing a subset S_{11} such that $S_{11} \subseteq S_1$, $S_{11} \subseteq S_2, |S_{11}| = m, m \leq n$. The resulting processing chain for PET images corresponding to subjects S_2 will contain the processing chain of Fig. 4.6 (a) for each subject $s \in S_{11}$.

Let the time required to complete segmentation of one anatomical image be t_s, and for transformation and warping t_{tr} and t_w, respectively. The total time taken to complete the processing chain for S_{11} is thus

$$t_{tot}(S_{11}) = mt_s + mt_{tr} + mrct_w$$

Given the running times of processing of table 4.1 and if $m = 6$ and we choose an experiment with $n = 10$, $r = 8$ and $c = 3$ as in Fig. 2.7, $t_{tot} = 372$ minutes. Usually, processing will be performed on several experiments, so the total time saved during processing of e.g. 100 experiments is 37200 minutes.

Now, given a WF Net $PN = (P, T, F)$, and a set of previously computed workflows, how do we decide whether or not a place $p \in P$ has already been created?

Definition 4.9. The *lineage* of a place $p \in P$, $lin(p)_{PN}$, is the WF Net $PN' = (P', T', F')$, a subgraph of PN, containing all vertices $v \in P \cup T$ on a path from i to $\bullet p$, and all edges $e \in F$, where $e = (v_i \rightarrow v_j),\ v_i, v_j \in P \cup T$.

If a place p in a WF Net PN needs to be computed and is found to have the same lineage as a previously computed place p' in a WF Net $PN_i = (P_i, T_i, F_i)$, we say that we can reuse p' instead of calculating p.

To decide whether or not there exists such a p' in the set of previously computed WF Nets $\mathcal{W} = \{PN_1, PN_2, \ldots, PN_N\}$, we need to visit at most $\sum_{i=1}^{N} |P_i \cup T_i|$ vertices, that is, we may need to visit all vertices in the is the worst case. This could easily be improved by labelling vertices in a suitable manner, but since we traverse the graph in a sequential order when processing workflows, this will not be a problem.

As neat as this mechanism may seem for improving efficiency, the real advantage of being able to query an object's lineage is the traceability achieved. If an object (image) is suspected to be corrupt, its entire history can be investigated to find the source of the error, and correct it. Even more fundamentally,

to be able to compare aggregated objects from different processing chains, the lineage of the objects should be considered: which standard brain was used in the transformation, which thresholds were set for the hypothesis testing and so on.

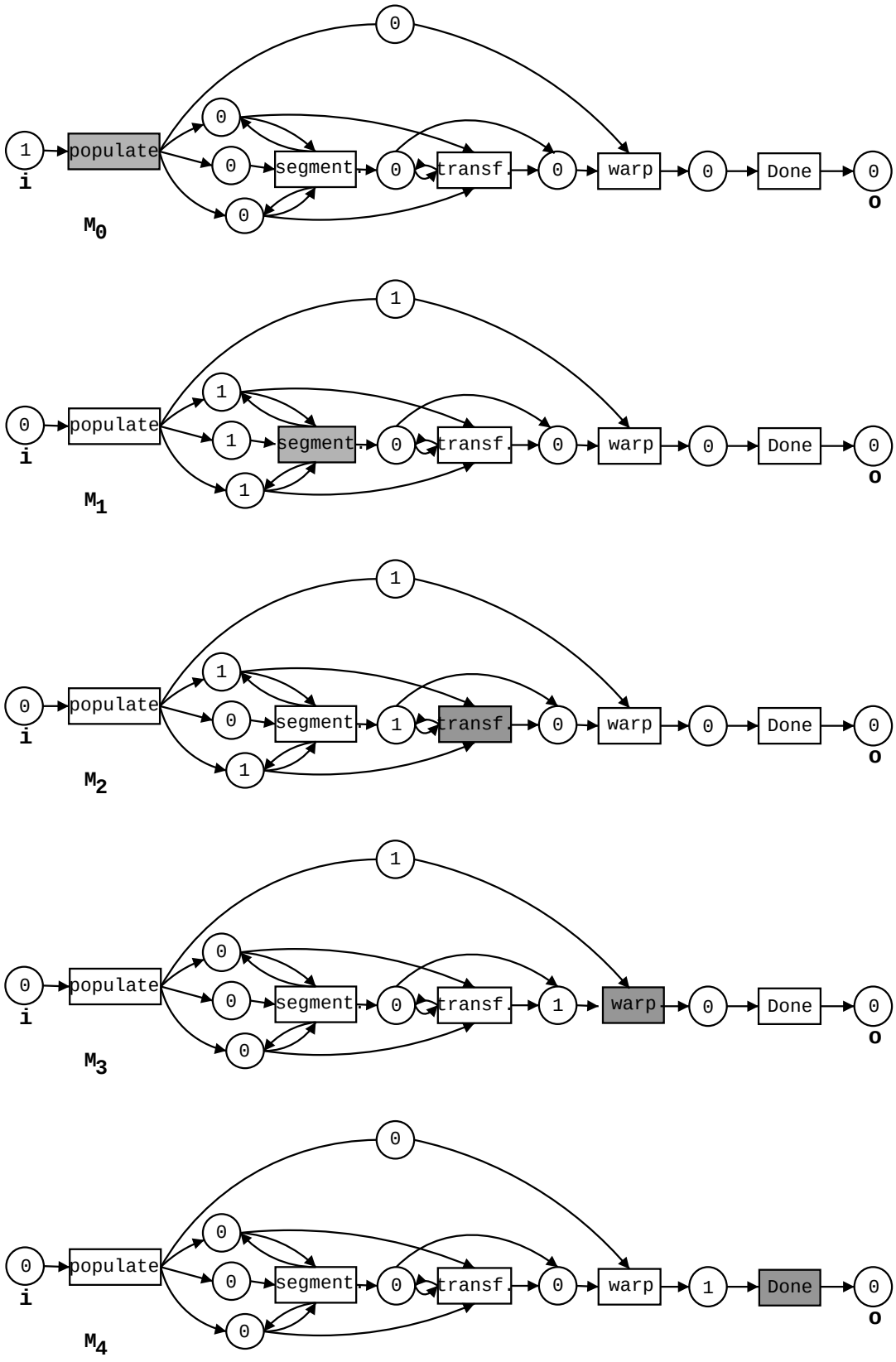


Figure 4.5. The firing sequence for a subset of the processing chain

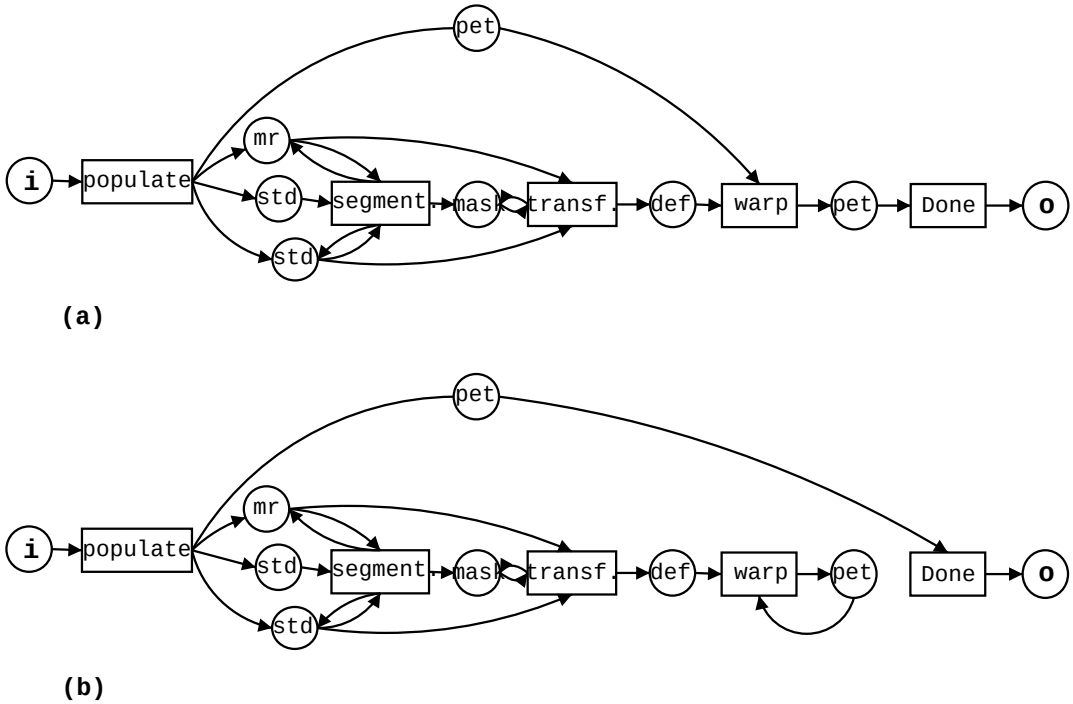


Figure 4.6. A sound workflow (a) and an unsound modification (b)

Chapter 5

Components and Frameworks For System Development

This chapter describes the tools that were used during the development of brain imaging meta-analysis systems:

- *The ECHBD system:* Built on top of the RasDaMan raster management system (section 5.2), which in turn uses the O_2 object-oriented DBMS [Deu92] for storage of so-called BLOBs (Binary Large OBjects).

- *The BINS/NG systems:*

  - *The software architecture:* builds on the mediator capabilities of the AMOS II mediator system (section 5.1) which is *extended* by use of the RasDaMan raster manager. As underlying DBMS, providing persistance and data security, the DB2 system was used.

  - *The workflow management system:* also uses the AMOS II system for active rules and object-oriented modelling.

  - *The workflow specification GUI:* uses the JHotDraw framework (section 5.3).

In the reuse categorization of chapter 3, usage of the AMOS II system and the RasDaMan raster manager belongs to the third category, system reuse, while the use of the JHotDraw framework belongs to both of the first two categories, design reuse and code-reuse.

5.1 The AMOS II Object-relational Database Management System

The AMOS II DBMS is a lightweight, main-memory, object-relational DBMS. It supports multi-database transparent communication (mediation) as well as the more basic client-server model of communication. The query language AMOSQL can be extended with so-called "foreign functions" in C, Java or Lisp. For more information on the AMOS II system, see [R+02a], [Jos99], [Ors96], [Skö94].

In [SM95], an object-relational DBMS (ORDBMS) is defined as capable of managing complex data *and* offering a query language. The former requirement separates ORDBMS from relational database systems, while the latter provides a distinction from object repositories such as ObjectStore.

For examples in this section, refer to the database schemas of Figs. 6.2 and 6.3.

5.1.1 AMOS II Data Model

The basic entities of the object-relational AMOS II type-system are shown in Fig 5.1. Notable in this hierarchy is the type *Function*. In contrast to, for example, Java, the AMOS II object model models functions (object *methods* in UML) as objects themselves!

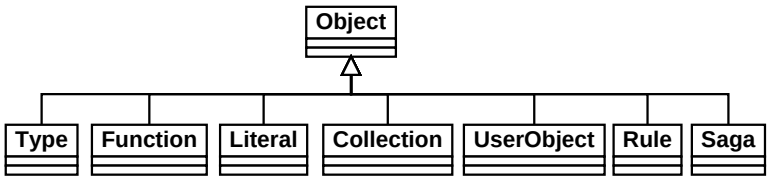


Figure 5.1. The AMOS II type system: A simplified one-level hierarchy

The data model of AMOS II is built from the functional DAPLEX model [Shi81] and Fig. 5.1 shows some of the basic entities in this model.

Here, the leftmost node is characteristically called *Type*, not *Class*, since a Type does not encompass methods on its instances (cf. Abstract Data Type, ADT). Subtypes of Type are *Stored Type*, *Derived Type* and *Mapped Type*, of which ordinary user-defined and built-in types are always stored - indicating a *local* storage, in contrast to the distributed derived types and mapped types, described in [Jos99]. An attribute, method or association to another Type, or *Literal*, is via an instance of *Function*. Literals are special (simple) objects, separated from Types by their fixed extent and the fact that

they are not associated with OIDs. *Integers*, *Reals*, *Charstrings* and *Booleans* are all examples of subtypes of Literal. A *Collection* is either a *Vector* or a multiset, *Bag*, that can contain any descendant of Object. All user-defined objects inherit from *UserObject*. *Sagas* and *Rules* are used for long-running transactions and ECA rules, respectively, and will be defined below.

The creation of a user-defined type, becoming a descendant of UserObject, along with some attributes, is described in example 5.1:

Example 5.1. Creation of a user-defined type.

```
CREATE TYPE Subject PROPERTIES(righthanded integer);
```

5.1.2 The AMOSQL Query Language

The AMOSQL query language originates from the OSQL [L+91] query language, resembling SQL99. It supports declarative queries of the form `SELECT...FROM...WHERE` as well as function definitions, overloading, foreign functions, multi-database integrative queries, ECA rules, long-running transactions and more [R+02a].

The declarative queries consist of the keywords `SELECT`, `FROM` and `WHERE` together with names of Types, Literals and Functions. An example query that finds lefthanded subjects, with a corresponding cluster image that overlaps at least 70% with cytoarchitectural area 17 is phrased in example 5.2:

Example 5.2. A declarative AMOSQL query.

```
SELECT s
FROM Subject s, ClusterImage cl
WHERE righthanded(s) = 0 AND
      is_subject_of(s, cl) AND
      image_overlap(cl, :a17) > 0.7;
```

The identifier `:a17` with a colon prefix is a globally defined AMOSQL variable, defined similar to example 5.3:

Example 5.3. A global declaration

```
DECLARE Contrast :contr;
CREATE Contrast :contr;
SET contr = ...;
```

The datatypes Subject and ClusterImage are defined in the database schema of Fig. 6.2. The functions `righthanded` and `is_subject_of` can easily be defined, the former a *stored* and the latter a *derived* function. The function `image_overlap` may be implemented as a *foreign function*, if cluster images

are stored on external files, blobs in a relational system, or in RasDaMan (5.2).

Functions can be either *derived*, *stored*, *procedural* or *foreign*. A derived function is nothing but a conserved query while a stored function is typically used to store an attribute with a type. A procedure supports more general constructs than the declarative derived function and the foreign function even allows calculations to be performed in an external language such as C, Java or Lisp. In example 5.4 below, we define the derived function `is_subject_of(...)`

Example 5.4. A derived AMOSQL function.

```
CREATE FUNCTION is_subject_of(Subject s, ClusterImage cl)
  -> boolean AS
    SELECT subj(exp(cl)) = s;
```

5.1.3 Mediation Capabilities

The AMOS II mediator system is capable of integration of data from heterogeneous datasources by the use of object-oriented views over *imported types* [Jos99].

The datasources may be e.g. specialized raster data management systems [FSR01], ordinary filesystems, XML documents [LRK00] or standard relational database management systems [FR97]. To perform data integration between an AMOS II database and a relational database, there exists an *OBDC translator* for AMOS II, that is capable of translating between AMOSQL and SQL. For other datasources, a translator has to be constructed

Assume we have stored image data in a DB2 relational database as blobs, while the experiment descriptions, the *metadata*, are stored in an AMOS II database. To integrate image data with metadata, we may for example store a unique identifier (`id`) at both datasources and then perform a join over two relations:

Example 5.5. A query accessing relational data.

```
/* set up ODBC connection to DB2 datasource and call it 'db2' */
/* then import relational table ClusterImage@db2 */

SELECT dbim
FROM ClusterImage@db2 dbim, ClusterImage im
WHERE id(im) = id(dbim) AND exp(im) = :exp;
```

In example 5.5 we fetch all ClusterImages from the DB2 database resulting from an experiment stored in the global AMOS variable :exp.

Into a translator may also be added optimization techniques based on capabilities of the translated datasource such as specialized index structures and optimization techniques. A translator for the RasDaMan system (see section 5.2) is being developed under the NeuroGenerator project.

5.1.4 Event-Condition-Action Rules in AMOS II

The Event-Condition-Action (ECA) rules of AMOS II [Skö94] are similar to triggers in SQL-99, providing an automatic constraint checking mechanism with corresponding actions.

A rule is also an object as shown in Fig. 5.1 and consists of:

- An event: either *updated*, *added* or *removed*
- A condition of the form: `when <predicate-expression>`
- An action to be taken if the condition is met

Example 5.6. A rule to check if a process terminated with improper exit signal (see section 6.3.2)

```
CREATE RULE notify_rule(CompProxy cp) AS
ON UPDATED(completed(cp))
WHEN completed(cp) = -1
DO notify_administrator(cp);
```

5.1.5 Long-Running Transactions in AMOS II

In AMOS II, it is possible to define a long-running transaction in the Saga [GMS87] framework. A Saga is defined as a set of transactions and a set of *compensating* transactions to be performed if the original transactions cannot be performed (i.e. if they are aborted). This is further described in section 4.3.2. In AMOS II, a Saga object can be created by the code snippet as below:

```
SET :s = create_saga();
Saga :s BEGIN
        <specify forward transactions>
      END;
      COMPENSATION BEGIN
        <specify compensating transactions>
      END;
```

The global variable ":s" now holds the Saga object, and to commit/abort this Saga, either of the following commands are issued:

```
commit_saga(:s);
```

or

```
abort_saga(:s);
```

It is further possible to construct hierarchical transactional structures through the use of *subsagas*.

5.2 The RasDaMan Database Management System

RasDaMan [BFRW97b] [BFRW97a] offers database management for Multidimensional Discrete Data (MDD), for example 3D brain images. The architecture is client-server oriented, with a server responsible for storage management and query processing and C++ or Java client libraries that communicate with the server using Remote Process Communication (RPC). The storage model is specialized for fast retrieval of subimages and the query language is built on an array algebra for manipulation and querying of array data of arbitrary dimension [Bau99].

5.2.1 Storage Model

The RasDaMan system uses an underlying base DBMS for its persistent storage of MDD:s in so-called BLOBs, objects with no semantical descriptions in the base DBMS, but modelled so that RasDaMan can interpret them.

The structure used for storage of data is designed to optimize access times of subimages, which is achieved by so-called *tiling* of subimages onto disk pages, thereby reducing the number of pages to read in order to access a subimage.

The tile-based storage (as opposed to conventional linear storage models) allows for fast subimage retrieval and is generalized to perform well for any number of dimensions. The sizes of the tiles can be preset by the user, and also tiles of different sizes and shapes can be used.

5.2.2 Query Language

RasQL is RasDaMans own query language for queries on MDD:s, optimizable both semantically and physically (i.e., queries can be rewritten based on an optimal algebraic execution order as well as with respect to the storage architecture). The physical optimization is achieved by keeping an index of the tiles of the images. The query language is reminiscent of SQL, as can be seen in example 5.7:

Example 5.7. "Extract all subimages defined by a certain subvolume, from the collection ClusterImage which contain at least one voxel with a value $> v$ within this subvolume"

```
SELECT cl[xmin,xmax:ymin,ymax:zmin,zmax]
FROM ClusterImage AS cl
WHERE some_cells(cl[xmin,xmax:ymin,ymax:zmin,zmax]) > v
```

5.2.3 Integration with underlying DBMS

RasDaMan stores and manages raster data and nothing else. Descriptive metadata have to be stored in a separate database in the underlying DBMS, not using RasDaMan. To be able to connect a raster data object with these metadata, its unique object identifier (OID) is needed. This OID is given to an image during insertion into the database, and must be propagated to the metadata before it is stored in the separate database for subsequent integration of metadata and rasterdata.

5.3 The JHotDraw Pattern Based Framework

The JHotDraw framework was originally developed for the Smalltalk language [Joh92], perhaps most known for its extensive usage of *patterns* (see section 3.3.1). The domain of applicability of the framework is the creation of tools for manipulation of semantically described graphical symbols, such as a UML editor or a Petri Net analysis tool. Its current incarnation in Java [JHo02] uses the same MVC architecture as the previous Smalltalk version, see Fig. 3.5.

The user manipulates a set of instances of subclasses to *Figure*, all of which is connected to a corresponding data model. All instances of Figure are contained in a *Drawing*, which has a view defined in the class *DrawingView*. User interaction is managed by *DrawWindow*.

The JHotDraw framework then corresponds to a model-view-controller in which the composite *model*, Drawing, contains several instances of Figure that

are in turn associated to display-independent information maintained in some class. Figures are in turn composable, in that an instance of the subclass *CompositeFigure* may contain other instances of Figure, delegating behaviour to its constituents, which corresponds to the Composite pattern, described in section 3.3.2.

Other patterns involved in the JHotDraw framework are *Strategy* and *State*, the former separating algorithms from the types the algorithm is defined for and the latter defines dynamic behaviour by adding a dimension of state to a context. Strategy is used for layout of Figures and State is used for e.g. popup menus.

To summarize, the JHotDraw framework provides reuse in terms of (cf. section 3.3.1):

- *Black-box reuse:* The behaviour of components of a CompositeFigure is completely defined by the components; the application programmer may not change component behaviour, other than by changing components and therefore knowledge of their implementation is superfluous.

- *White-box reuse:* A Figure may be subclassed, involving overloading of methods and addition to the data model. The programmer will need to understand the interface of the superclass he/she is specializing but not necessarily details on *how* the superclass implements the interface.

- *Frozen spots:* The MVC architecture requires adherence to the priciple of separation of presentation, model and control of the system. Moreover, to be able to use JHotDraw as a framework, one has to work by subclassing Figure and supplying a corresponding model.

- *Hot spots:* Extending the set of descendents of Figure, addition of State:s, usage of and addition to the set of Strategy:s and so on.

Chapter 6

Development of a Workflow System for Meta-analysis

For the Neurogenerator and BINS projects mentioned in section 1.2, one of the major tasks are processing raw image data from neuroimaging experiments. The complexity of this task was described in chapter 2 and some of the solutions were hinted in 4. The actual implementation of a system to satisfy these demands are described below. Focus has been on reuse of database concepts such as active rules (section 6.3.1) and transactional models (section 6.3.2) to speed up development and keep down costs.

6.1 Managing Processing Chains

The processing chain previously described (section 2.3.2) for transformation of raw PET image data into statisical cluster images is made up of processing modules as listed below. The list of processing modules are according to [Hal], from a forthcoming publication. There are some differences between this list of actual programs and the conceptual sketch described in section 2.3.2, for example the two separate processing stages *GLM* and *clustering* are performed by one program, and has not been reengineered into the more conceptually appealing two-step approach of section 2.3.2. Another difference is practicalities such as possible speed-enhancements by subsampling images to the effective scanner resultion instead of using an oversampled default voxel size.

The below set of processing modules will form a running example of the processing chain for PET image data, depicted conceptually in Fig. 2.7 and the actual implmentation in described in Fig. 6.6. Note that this is not the only possible processing chain but merely an initial example of how to process PET imaging data by the current state-of-the-art algorithms. Methods

and implementations can be expected to change in the future and the new possibilities for meta-analysis will benefit greatly from this flexible modelling. This is examplified by description of addition of a new processing module in section 6.3.3.

- *Segmentation:* The segmentation of bone and tissue other than brain substance from the image to be transformed [UL02].
  Input data:
  - *A standard brain, segmented*
  - *A standard brain, unsegmented*
  - *An anatomical image*

  Output data:
  - *A segmentation mask*

- *Transformation:* The calculation of a transformation from the individual brain to a standard brain space [WCM92]
  Input data:
  - *A standard brain, segmented*
  - *An anatomical image*
  - *A segmentation mask*

  Output data:
  - *A deformation field, describing the transformation*
  - *A transformed anatomical image*

- *Warping:* The application of a previously calculated deformation field to functional images.
  Input data:
  - *A deformation field*
  - *A functional image*

  Output data:
  - *A transformed functional image*

- *Subsampling:* For speeding up execution of other processing modules, we may wish to subsample images.
  Input data:

  - *A functional/anatomical image*

  Output data:

  - *A subsampled functional/anatomical image*

  Parameter:

  - *A subsample factor*

- *Masking:* Calculating a mask for removing so-called null-values from images prior to the statistical analysis:
  Input data:

  - *A set of functional images*

  Output data:

  - *A null-value mask*

- *Conversion:* Convert from scanner format into internal image format.
  Input data:

  - *A functional/anatomical image (scanner-specific)*

  Output data:

  - *A functional/anatomical image (internal format)*

- *Statistical Analysis:* A linear model with subsequent statistical inference resulting in a cluster image of activations [Led00].
  Input data:

  - *Contrast vector(s)*
  - *A designmatrix*
  - *A set of functional images*
  - *A null-value mask*

  Output data:

  - *Three cluster images corresponding to p-values: 0.05, 0.01 and 0.005*

In Fig. 6.6 the full PET processing chain is depicted, using the programs listed above. While the conceptual stages of processing were described in section 2.3.2 we are here concerned with the integration of the processing chain modules with the workflow management system. Some additions and differences from the conceptual description are, according to [Hal]:

- *Conversion:* A conversion module is added for converting scanner-specific raw data into internal format.
- *Subsampling:* The functional images are supersampled to match the resolution of the anatomical images. Therefore, subsequent to warping, the functional images are subsampled to their original resolution to speed up the Monte-Carlo simulations of the statistical analysis.
- *Mask for statistical analysis:* The functional scanner may not be able to deliver a value for the activation-dependent variable in a certain voxel, resulting in so-called *NaN*[1] values. The location of all NaN values are removed from statistical analysis by creating a mask that is sent to the module for statistical analysis.
- *Statistical analysis:* The statistical analysis processing module combines the general linear model and subsequent cluster analysis.

6.2 Architectural Overview

The workflow system is built on top of the object-oriented, main memory active mediator system (DBMS) AMOS II, see section 5.1. This approach demands a powerful modelling capability of the mediator system, such as the object-relational model of AMOS II. As discussed in [A+98], this approach has some advantages, especially for reducing implementation efforts through re-use of facilities such as Event-Condition-Action (ECA) rules and transaction managment. On the other hand, it may produce difficulties in re-use of database-*oblivious* concepts, such as a C++ API for Petri Nets.

The system is interfaced to an IBM parallel computer centre in Stockholm, the Center for Parallel Computers (PDC), offering 178 parallel nodes equipped with a large common secondary storage and automatic migrability to tape stations. To interface the workflow system with this architecture, a client-server computational model was employed, in which the central Workflow Managment System (WFMS) server distributes the accumulated jobs onto client nodes according to a cost-based model of different jobs. Details on this are presented in 6.4.

[1] The familiar "Not a Number" symbol

The architecture can, from a client-server perspective, be said to consist of a server part, *WFServer*, and several light-weight clients distributed on the processing nodes of the parallel computer. Both the server and the clients are implemented to run within AMOS II, thereby utilizing built-in communication layers, support for long-running transactions and ECA rules present in the AMOS II DBMS.

The server can be described by a five-layer architecture as in Fig. 6.2. The *communication layer* listens to incoming requests from clients through the TCP/IP layer of AMOS II. The *workflow logic* is basically to check if there are any computations that can be performed (has all its indata present), and if so, schedules it to be performed on a client. This checking is performed by automatically querying the *representation layer* through ECA rules. If something goes wrong in the client execution, exit signals from the external UNIX process is registrered and sent to the server, which takes action according to the *transaction layer*, using so-called Sagas that are implemented in AMOS II. The representation layer maintains an object-oriented view of the processing chain; its objects and connections are represented as a bipartite graph as described in section 6.3.

A prototypical graphical user interface (GUI) is being implemented in Java, using the JHotDraw framework (see section 5.3) and the same client-server mechanism as described above to connect to the WFServer.

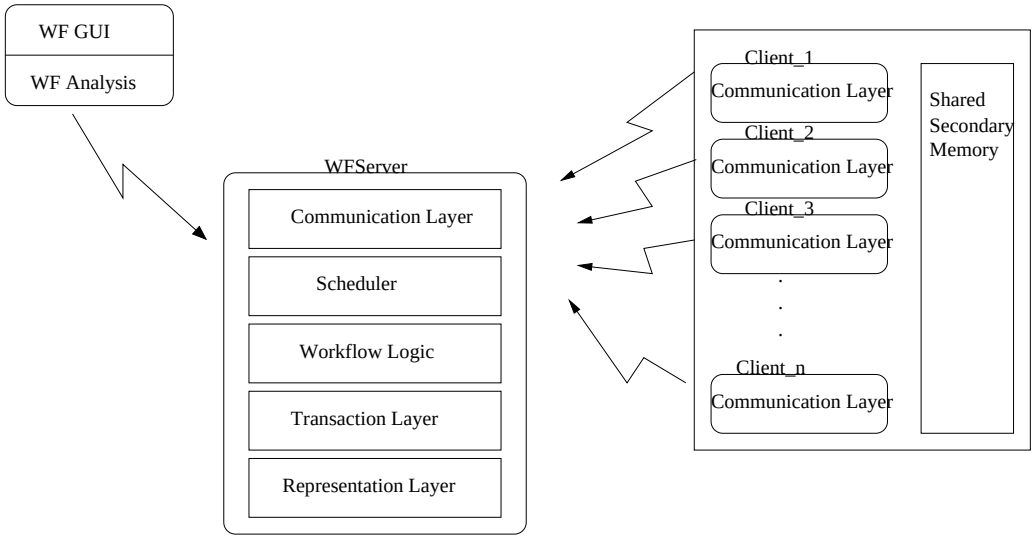


Figure 6.1. The workflow client-server architecture

To provide robust handling of client crashes and for special processing requirements, a transactional model is employed. More on this can be found

in 6.3.2.

6.3 Workflow Model Implementation

The workflow model is implemented in an object-oriented fashion, somewhat similar to a Petri Net model, as described in section 4.4, but also borrows from the long-running transactional model of section 4.3.

To discuss the workflow management system we start by displaying a much simplified version of the database schema in Fig. 6.2. All attributes and class methods were omitted for simplicity.

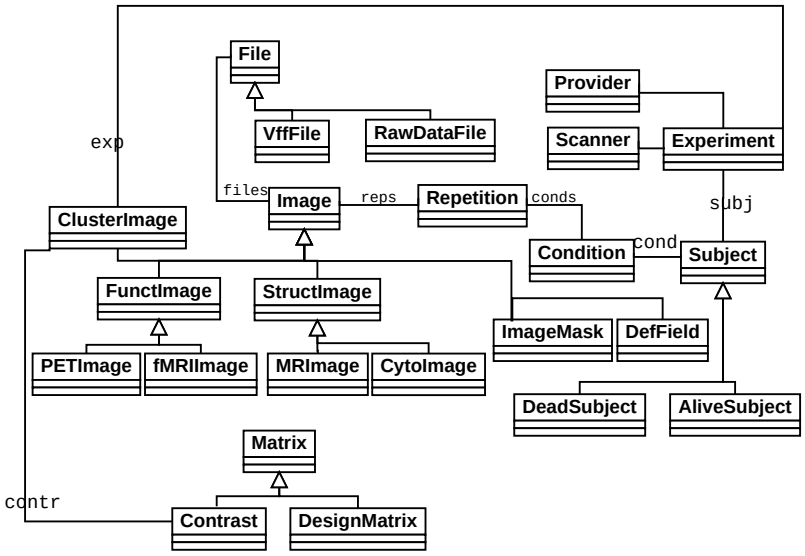


Figure 6.2. A simplified class diagram of experiments

The database schema for the workflow system can be found in Fig. 6.3. The workflow model is simple, building on a couple of patterns, 3.3.2:

- *Type-Object*: A program is modeled by the class *PCModule*. All instances of this program *running* on a parallel node, is modeled by a *CompProxy*. In the same manner, a *DataSource* represents in- or outdata to a CompProxy. DataSource is coupled to the baseclass of AMOSII, called *Object*. Many DataSources, but not all, will be coupled to subclasses of *Image*, presented in the schema describing experiment data, 6.2.

- *Composite*: A CompProxy is either *composite* (un-ordered multiset), *sequential* (ordered multiset), or simple. A *SimpleCP* represents the last category by linking to a PCModule, containing information on how to run this program. Note that the class *Workflow* is defined as a subclass of a *CompositeCP*.

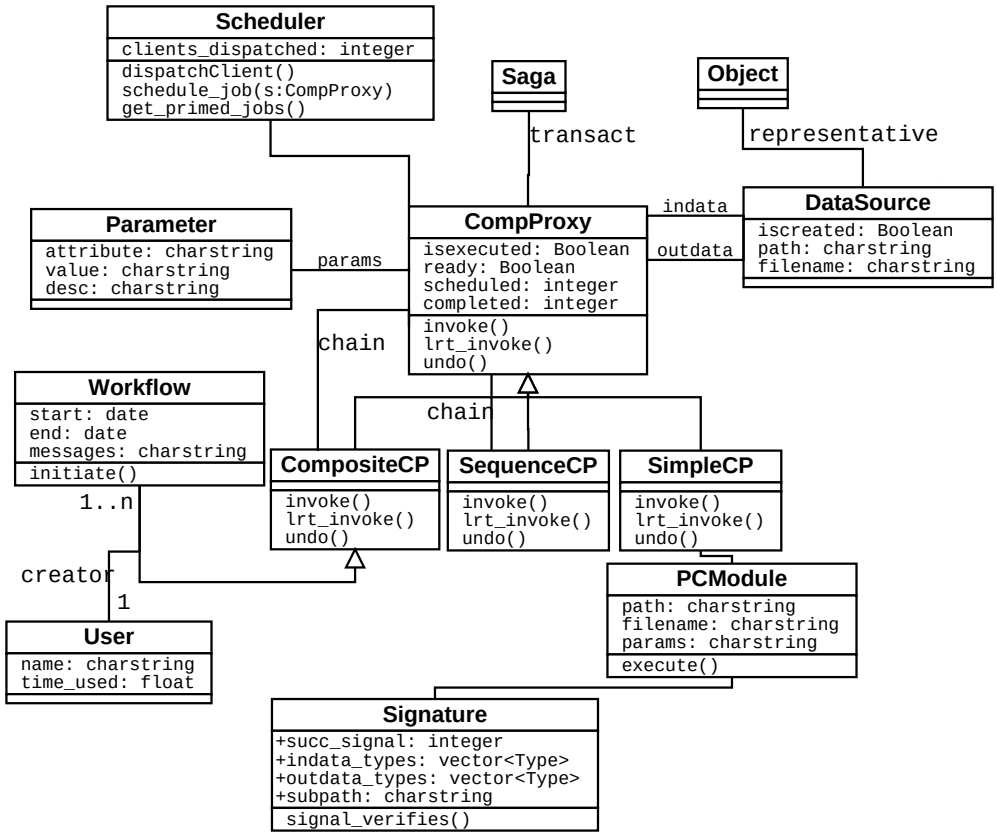


Figure 6.3. The workflow schema

The *Scheduler* manages the contact with the supercomputer scheduler, stores submitted CompProxy:s until a certain total execution cost is reached. The execution cost for each CompProxy is modeled in the PCModule and when CompProxys of a certain total cost is reached, then the corresponding jobs are scheduled on the supercomputer.

Stored by each CompProxy is also a Saga object, which enables transaction management during workflow.

The *Signature* class models input and output and remembers which operative system level signal that is expected from a PCModule. If that signal is not received, compensations (see section 6.3.2) will fire and clean up after the faulty operation.

6.3.1 Active Rules for Triggering Events in Workflows

A common model for supporting dynamic behaviour in databases is to use *Event-Condition-Action* (ECA) rules [DHW94]. ECA rules makes it possible to specify constraints on the contents of the database, (e.g. to place a *condition* on an attribute of an object, in the *event* of an update of that attribute) and matching *actions* to take when the conditions are met.

In [GD93], the task of detecting events in a database system was solved using Petri Nets. We are going to use the opposite approach and follow [DHL90] in using ECA rules to implement a Petri Net-like structure for managing workflows.

The active rules in AMOS II, described in section 5.1.4 are used by the WFServer to detect the event of an update of the workflow structure, namely when a DataSource is created, and the condition that all input data to a CompProxy is present. This should be compared with definiton 4.5, the firing rule for a Petri Net. Using the simple ECA rule below, we implement the appropriate firing rule for our workflow structure:

```
create rule execute_cp(CompProxy cp) as
  from DataSource ds
  on updated(created(ds))
  when ds = indata(cp) and ready(cp) = 1 and
       scheduled(cp) = 0 and completed(cp) != -1
  do lrt_invoke(cp);
```

Take the example of Fig. 6.6, where we start out with scanner-specific images that already exist when the workflow is defined. To initiate processing we set the attribute `created` of each scanner-specific image to 1. The update event is now caught by the ECA rule, which tests that all input data are present through the function `ready`, that this process has not already been scheduled and also that the process has not already been aborted (`completed = -1`).

The action part of this rule-instance now executes the `lrt_invoke(CompProxy cp)` function, that starts up a process under the Saga transactional framework as described in section 5.1.5. The process is started as described in section 6.4 and upon successful termination updates the database

by setting the attribute `created` of all output data from the CompProxy to 1, which may enable segmentation of the converted images.

We may add more rules to this simple rule that takes care of the initiation of processing. For example, if a process has aborted, then depending of the way the process terminated, we either automatically re-submit the processing or notify the administrator. An automatic re-submit would be suitable if the alloted time on the supercomputer is out and the processing can continue from where it was stopped in a subsequent re-submission of the processing.

The number of rules in a workflow system should probably be kept small, since the increased complexity of state transitions that rules infer may lead to anomalies such as infinite loops.

6.3.2 Transactional Model

In section 6.3.1, the rule-execution revealed many of the possible states of the transactional model depicted in Fig. 6.4 (a). We may not always want to execute a process within a transaction and then we employ the non-transactional model of Fig. 6.4 (b). In the figure, italicized words denote state transitions while normal font indicate states (we borrow the notation from [KS95]) and, further, underlined transitions denote controllable transitions. That is, if a process is terminated abnormally on the *operative system-level*, in the Saga transactional model, the appropriate *compensations* (see section 5.1.5) are programmed to be executed. The other controlled transition of Fig. 6.4 (a), also present in (b), is the reset of states to be performed before a new transaction can be initiated. This stage may be necessary to perform manually.

The compensating transactions $\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ should be designed to perform the inverse of the forward transactions $\{\tau_1, \tau_2, \ldots, \tau_n\}$, for example to clean the working directory for intermediate results, if any.

6.3.3 Interfacing Existing Analysis Programs to the WFMS

The WFMS executes external programs by issuing operative system-level calls from the WFClients of Fig. 6.2. To be able to automate this, all executables have to be wrapped in a script assuring that the invocation of the executable is on the form:

```
<executable> -i <indata_list> -o <outdata_list> -p <parameter_list>
```

where `indata_list`, `outdata_list` and `parameter_list` are blankspace delimited lists of indata, outdata and parameters, respectively.

An example of a modified processing chain could be to stop after the warped PET images, the end-result of Fig. 4.6 (a) and add a correlation module operating on a set of warped PET images.

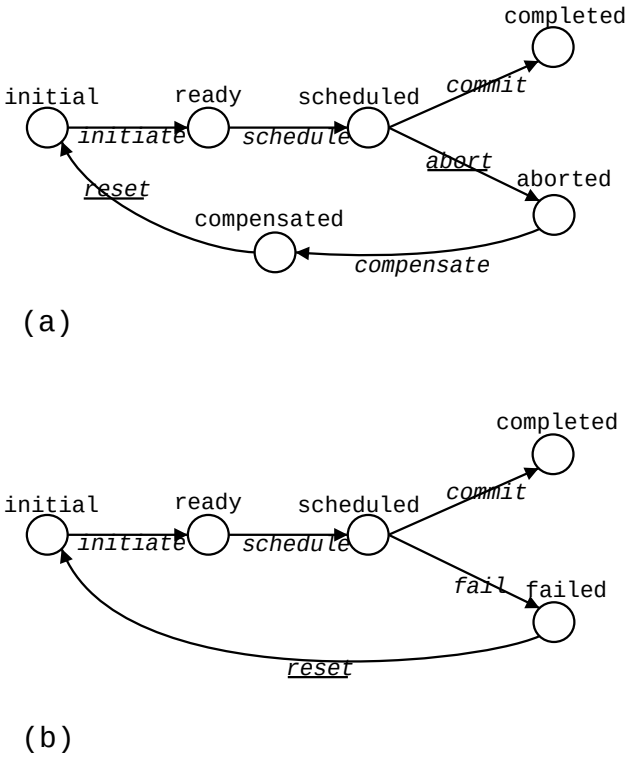


Figure 6.4. The Saga transactional model (a) and the non-transactional model (b) employed in the WFMS

To add a new correlation module operating on a set of warped PET data one needs to:

- *Wrap the existing executable:* Add a script `correlation.s` that is executed by the following sequence:

  ```
  correlation.s -i pet_1.vff pet_2.vff ... pet_n.vff -o corr.vff
  ```

  The script calls the original correlation executable and produces the output data `corr.vff`[2].

- *Create a PCModule in the WFMS:* An AMOSQL constructor `insert_pcmodule( ... )` is called with appropriate input data.

- *Create an AMOSQL function for creation of CompProxy objects:* The PCModule can be used on its own, but providing a function that produces CompProxy:s corresponding to this PCModule is handy.

[2] vff is the suffix of the internal file format

- *Plug in the new type of CompProxy to the appropriate place in a workflow:* See Fig. 6.7. The AMOSQL code needed to produce this will be very similar to that described in section 6.4.1, remove the last lines inserting CompProxy:s for the mask and the statistical analysis and place the correlation module before the preceeding `END` to be executed for functional images corresponding to each subject.

6.4 Distributing Image Processing: A Simple Client-Server Approach

The method employed for distributing processes across the parallel computer is based on the client-server communication between instances of AMOS II databases. The *Workflow Server* maintains a full database, containing the raw, unprocessed data as well as a representation of the processing chain (see section 2.3.2) and its intermediate data as described in section 6.3 When an ECA rule is triggered, as described in section 6.3, and thus a process is ready to start, it is put in a *processing queue*.

Each process has a cost associated with it, based on prior knowledge of the time it takes to finish, and when a certain total cost has been reached in a queue, the scheduler of the parallel computer[3] receives a request to start a job on a certain kind of node, see table 6.4 with an allowed CPU time somewhat higher than the total cost of the processes currently in the queue.

Type	MHz	RAM (MB)	# of processors	# of nodes
T	160	256	1	122
W	160	512	1	8
Z	160	1024	1	2
M	332	1024	4	21
N	222	4096	8	7
H	222	16384	8	1

Table 6.1. Types of parallel computer nodes in the IBM Strindberg computer

This job starts up an empty AMOS II database on the alloted processing node, which immediately connects to the workflow server, identifies itself and asks for the processes in its processing queue. When a process finishes, the database at the server is updated and when all processes are finished, the workflow client terminates. The updates to the server database might trigger new processes as described in section 6.3.1 and thus new processes are queued.

[3]not to be confused with the WFMS class Scheduler of Fig. 6.3

The processing chain currently work on files, not database objects, which removes the problem of distribution of data, since the parallel computer system is equipped with a fast secondary memory shared between all the nodes.

Comparing this with the evolutionary development devised in section 3.4, we note that in the final stage of the architecture we will need an automatic database population program for images that should go into the specialized raster database management system RasDaMan (section 5.2), a program that could be added to the processing chain of Fig. 6.6.

6.4.1 AMOSQL Interface for Managing Workflows

The WFMS is implemented in the AMOSQL language with some extensions in the form of Java foreign functions as described in section 5.1.1.

For each PCModule that may be executed in a workflow there is an AMOSQL function that inserts a CompProxy of that type at the correct place in the processing chain. These functions can then combined in the below AMOSQL pseudo-code database procedure for specifying a workflow for one PET experiment, (cf. Fig. 6.6):

```
CREATE FUNCTION insert_workflow(PETExperiment exp) -> Workflow wf
AS BEGIN
  FOR EACH MRImage mr, FunctionalSubject fs
  WHERE mr = anatomical_image(fs) AND fs = subjects(exp)
  BEGIN
       /* Conversion of anatomical images */

       /* Segmentation of anatomical images */

       /* Transformation of anatomical images */

       FOR EACH PETImage pet
       WHERE pet = functional_images(sessions(fs)) AND
             fs = subjects(exp)
       BEGIN
            /* Conversion of anatomical images */

            /* warping of PET images */

            /* subsampling of PET images */
        END;
    END;
```

```
    /* Generate mask for imagemodel */

    /* Run image_model (statistical analysis) */

END;
```

Once the workflow has been specified, either through AMOSQL as sketched above, or with the help of a graphical user interface, see section 6.5, the workflow may be initiated, paused, and resumed through purposefully defined AMOSQL functions. When a process is triggered as described in section 6.3.1, the following function is executed:

```
CREATE FUNCTION lrt_invoke(CompProxy cp) -> CompProxy AS BEGIN
  DECLARE Saga s;

  SET s = transact(cp);

  Saga s BEGIN
      invoke(cp);
  END;
  COMPENSATION BEGIN
      undo(cp);
  END;

   RESULT cp;
END;
```

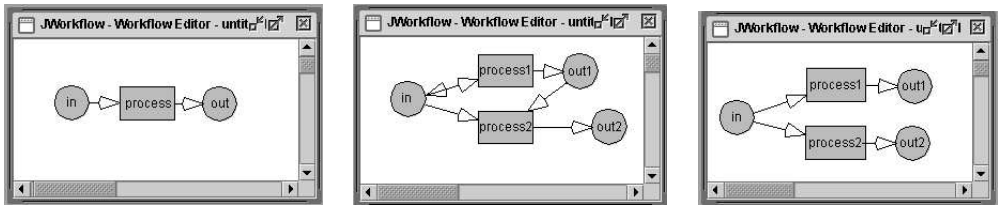
The function `invoke(CompProxy cp)` called from within the Saga schedules the CompProxy and sets the state acordingly, see figure 6.4.

For each PCModule, wherever possible, there is a compensating `undo`-function that is executed if, during execution of the corresponding CompProxy, an unsuccessful exit signal[4] of the operative system-level process is encountered.

6.5 A Graphical Language for Workflow Specification

Workflow interaction can be accomplished through the AMOSQL interface to the WFMS, but a graphical user interface (GUI) will provide a more intuitive interaction at least for new users. Further, the definition of a graphical

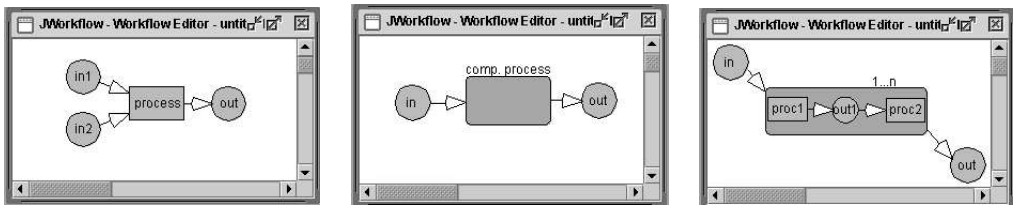
[4] often a signal $\neq 0$



(a) Serial processing

(b) A non-consuming process (process1)

(c) Parallel split of processing



(d) Parallel join of processing

(e) Composite process

(f) Multiple parallel instances

Figure 6.5. Graphical language constructs in the WFMS GUI

language for workflow specification will add to a consistent description of workflow dynamics and provides a nice environment for incorporation of Petri Net based analysis, as described in section 4.4. Last but not least, the use of the JHotDraw environment for constructing the GUI provides an excellent example of reuse of frameworks building on patterns.

6.5.1 Workflow Constructs

The workflow constructs so far present in the WFMS are depicted in Fig. 6.5 and are based on the needs of the processing chain as described in section 6.1. Since the processing chain is massively parallel in structure, the construct of Fig. 6.5(f) proved especially valuable, as can be seen in Fig. 6.6 where the PET processing chain depicted conceptually in Fig. 2.7 is inserted into the WFMS GUI. The workflow contains 3 conditions and 8 repetitions for which the functional images are converted and subsampled in parallel. Moreover, the number of subjects are 10 and thus conversion of anatomical images, segmentation and transformation are performed in parallel *as well as* the aforementioned 3×8 instances of conversion, warping and subsampling of functional images, resulting in e.g. 240 conversions of functional images to internal format.

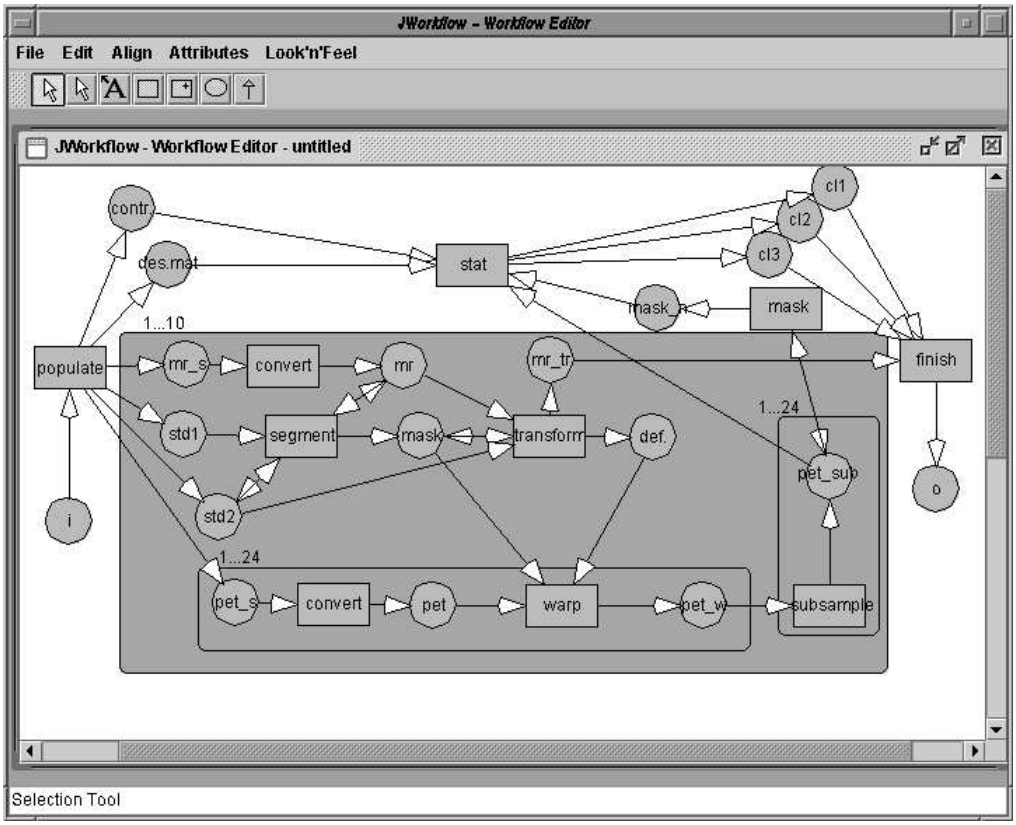


Figure 6.6. The full PET processing chain for 10 subjects and 8×3 functional images

6.5.2 GUI Implementation

The WFMS GUI is being constructed using the JHotDraw framework defined in section 5.3 and the AMOS II Java API for connecting to the WFServer as depicted in Fig. 6.2. The lower part of Fig. 6.8 shows the JHotDraw classes reused in the implementation and the upper part are the WF GUI classes. The main idea of building on the JHotDraw framework is that all user interaction and presentation can be reused directly through the inheritance of *LineConnection*, *EllipseFigure*, *RectangleFigure* and *GraphicalCompositeFigure* from the main GUI presentation classes *ArcConnection*, *PlaceFigure*, *TransitionFigure* and *CompTransFigure*, respectively.

Of these, PlaceFigure and TransitionFigure share a number of features that were put into the *PNFigure* interface. Their presentation-independent data are maintained by the abstract superclass *PNVertex*.

The *WorkflowFactory* connects the WFMS GUI to the WFServer through usage of the Java callin package of AMOS II (see section 5.1), using the class

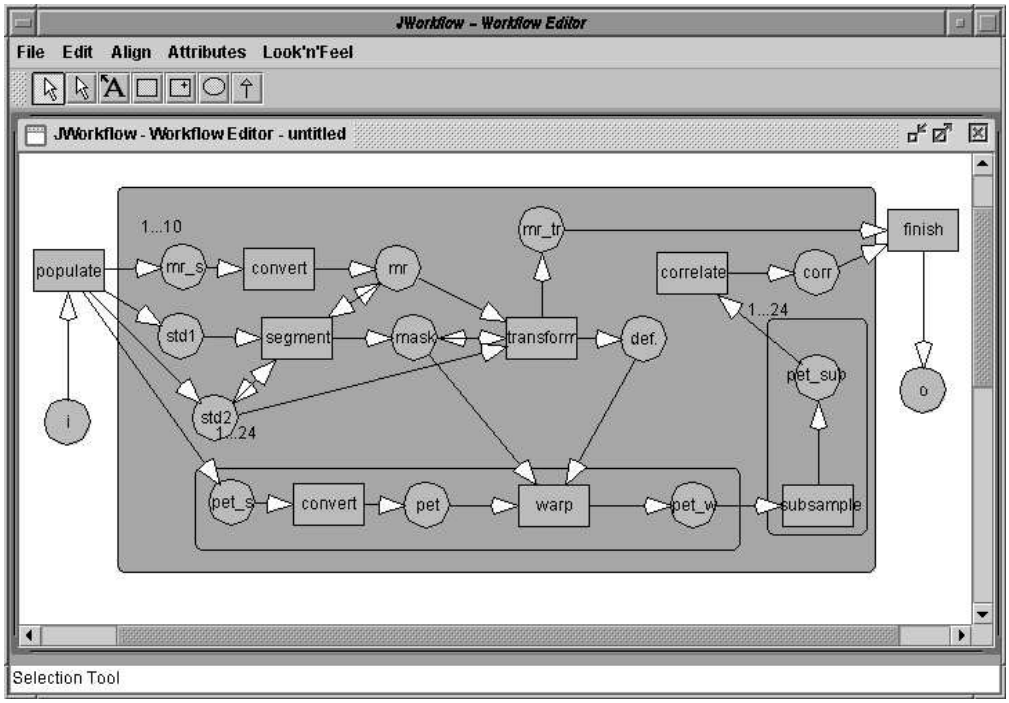


Figure 6.7. A modified processing chain using a (hypothetical) correlation module

Connection for opening a client connection to the WFServer. By calling the class Signature (see Fig. 6.3), keeping track of datatypes and cardinality of input and output data corresponding to a PCModule, in the WFMS database, constraints on how to connect the processing chain can be enforced. The Petri Net analysis methods described in section 4.4 are plugged in via the *AnalysisStrategy* class and can be used to e.g. verify that a workflow is sound.

At present the WorkflowFactory and the AnalysisStrategy classes are not implemented, but they are still included in the UML diagram of Fig. 6.8 to indicate how these concepts can be plugged in to the system. A connection to the WFMS will aong other things enable the user to monitor and manipulate the status of running workflows, to for example obtain estimates on when the workflow will be finished or pause and resume workflows.

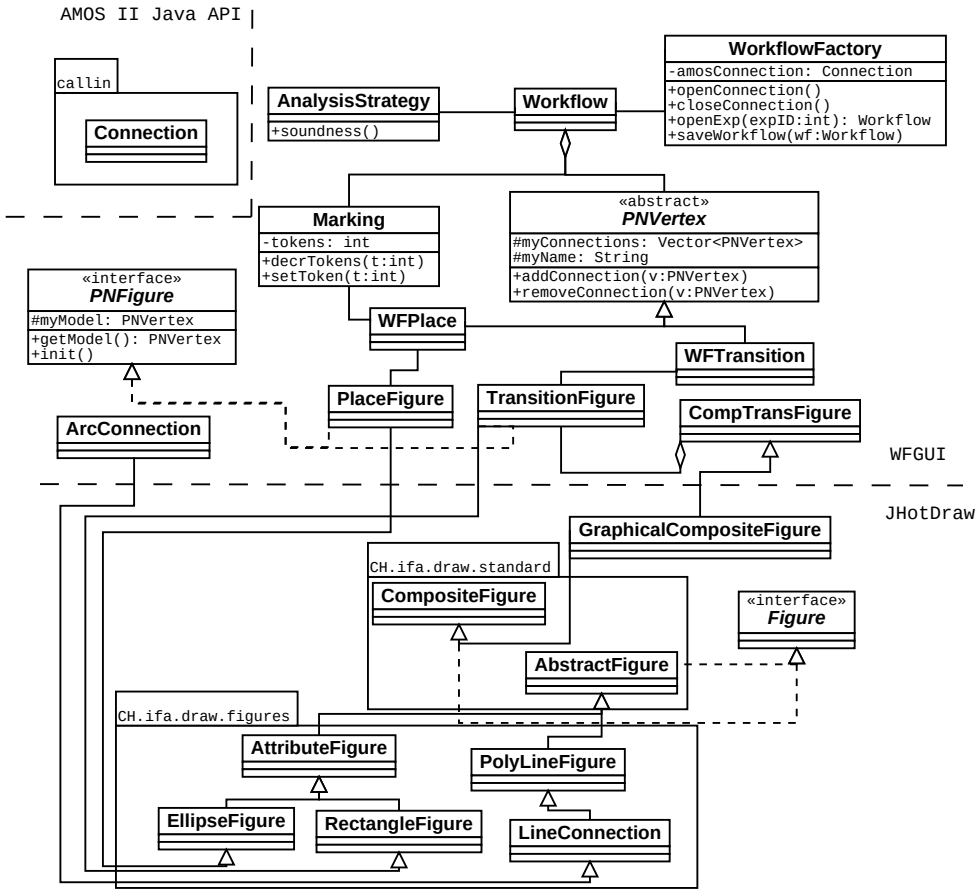


Figure 6.8. A classdiagram of the JWorkflow GUI

Chapter 7

Conclusions

We relate experiences in designing and developing meta-analysis systems for brain research. The methodology for system construction described in section 3.4 is new and we present a novel view on how to design a workflow environment for brain imaging meta-analysis in chapters 4 and 6 as an example of the methodology. The development of the workflow system was cost-effective due to the reuse of database concepts as described in chapter 6.

Although databases for managing brain imaging experiments are already manifold, as presented in chapter 2, the aspects related in this thesis, such as evolutionary development and image processing management for the domain of brain imaging research, have not been previously discussed. We believe that the system being designed will help participating researchers in formulating new hypotheses about brain function, as discussed in section 2.4.2. We also believe that other systems for brain imaging experiments, as well as other complex systems, may benefit from the ideas about evolutionary development and workflow management related in chapters 3 and 4, 6, respectively.

In the field of scientific workflow management, we believe the concept of traceability have not been given a formal definition as in section 4.4.3

Future work includes a more formal definition of the methodology described in section 3.4 and research on applicability outside the domain of construction of systems for brain imaging research and the organizational aspects of the methodology (such as defining types of organizations that may profit from the decentralized approach). In the field of workflow management, the benefits of a Petri Net based analysis for scientific workflows and the usages of workflow management in data mining and meta-analysis should be investigated.

For the BINS and NeuroGenerator projects, the most interesting issues are related to aiding researchers in formulating new hypotheses on brain function, such as providing tools for meta-analysis and data mining.

Acknowledgments

First of all I thank my supervisor, Per Svensson, for all his support and help during the work presented here. The project groups of BINS and NeuroGenerator has been of much help, Hjörleifur Halldorsson has made a great effort in organizing the processing chain, without which the workflow system would not have been easy to finish. Lars Forsberg always has interesting suggestion and working with the group from the parallel computer centre (PDC) also consisting of Gert Svensson, Christian Engström and Anders Selander has been fruitful; many ideas have come out of these meetings. I want to thank Per Roland from the Karolinska Institute Human Brain Research group, and all the members of the lab, for their help and support. I also want to thank Tore Risch of Uppsala University for helping me a lot with the AMOS II system. Many thanks to Karim Oukbir, Johannes Keukelaar and Klas Wallenius at NADA, KTH, for interesting discussions, sometimes even work-related.

I finally want to thank Josefin Herolf especially for help with the brain images of chapter 2 and for putting up with me during the last few weeks.

Bibliography

[A+96] M. Arya et al. *Multimedia Database Systems; Issues & Research Directions*, chapter Design and implementation of QBISM, a 3-D medical image database system, pages 79–100. Springer Verlag, 1996.

[A+98] A. Ailamaki et al. Scientific workflow management by database management. In M. Rafanelli and M. Jarke, editors, *Statistical and Scientific Database Management*, pages 190–199, Capri, Italy, July 1998. IEEE Computer Society.

[Adl91] R. J. Adler. *The Geometry of Random Fields*. John Wiley & Sons, 1991.

[AIS+77] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language - Towns, Buildings, Construction*. Oxford University Press, 1977.

[Ale79] C. Alexander. *The Timeless Way of Building*. Oxford University Press, New York, 1979.

[All97] R. J. Allen. *A Formal Approach to Software Architecture*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, May 1997.

[Bau99] P. Baumann. A database array algebra for spatio-temporal data and beyond. In *Fourth International Workshop on Next Generation Information Technologies and Systems*, Lecture Notes in Computer Science, pages 76–93. Springer-Verlag, 1999.

[BFG93] S. C. Bandinelli, A. Fuggetta, and C. Ghezzi. Software process model evolution in the SPADE environment. *IEEE Transactions on Software Engineering*, 19(12), December 1993.

[BFRW97a] P. Baumann, P. Furtado, R. Ritsch, and N. Widmann. Geo/environmental and medical data management in the rasdaman system. In *Proceedings of the 23th VLDB Conference*, pages 548–552, 1997.

[BFRW97b] P. Baumann, P. Furtado, R. Ritsch, and N. Widmann. The rasdaman approach to multidimensional database management. In *Proceedings ACM Symposium on Applied Computing*, pages 166–173, San Jose, CA, USA, 1997. ACM Press.

[BH95] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *J. of the Royal Statistical Society B*, 57:289–300, 1995.

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architectue. A System of Patterns.* John Wiley & Sons, West Sussex, England, 1996.

[Boe88] B. W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, 21(2):61–72, May 1988.

[Bow96] F. D. J. Bowden. Modelling time in petri nets. In *Proceedings of the second Australia-Japan Workshop on Stochastic Models in Engineering, Technology and Management, Gold Coast, Australia*, July 1996.

[BW01] L. C. Briand and J. Wust. Modeling development effort in object-oriented systems using design properties. *IEEE Transactions on Software Engineering*, 27(11):963–86, November 2001.

[C+94a] J. V. Carlis et al. A zoomable DBMS for brain structure, function and behavior. In *Applications of Databases*, pages 299–316, 1994.

[C+94b] J. B. Cushing et al. Computational proxies: Modeling scientific applications in object databases. In J. C. French and H. Hinterberger, editors, *Statistical and Scientific Database Management*, pages 196–206, Charlottesville, Virginia, USA, July 1994. IEEE Computer Society.

[CKKE97] C. A. Cocosco, V. Kollokian, R. K-S. Kwan, and A. C. Evans. Brainweb: online interface to a 3-D MRI simulated brain database. *Neuroimage*, 5(4), 1997.

[Cop99] J. O. Coplien. Reevaluating the architectural metaphor: Toward piecemeal growth. *IEEE Software*, pages 40–44, September/October 1999.

[CS95] J. O. Coplien and D. C. Schmidt, editors. *Pattern Languages of Program Design.* Addison-Wesley, 1995.

[CS96] P. Churchland and T. Sejnowski. *The Computational Brain.* MIT Press, Cambridge, MA, USA, 1996.

[CSAK99] C. Tegeler C, S. C. Strother, J. R. Anderson, and S. G. Kim. Reproducibility of BOLD-based functional MRI obtained at 4 t. *Human Brain Mapping*, 7(4):267–83, 1999.

[D+97] A. E. Dashti et al. Database challenges and solutions in neuroscientific applications. *Neuroimage*, 5(2):97–115, February 1997.

[Deu92] O. Deux[1]. *The Story of O_2, Building an Object-Oriented Database system.* Morgan Kauffman publishers, 1992.

[DHL90] U. Dayal, M. Hsu, and R. Ladin. Organizing long-running activities with triggers and transactions. In *Proceedings SIGMOD Conference*, pages 204–214, Atlantic City, NJ, USA, 1990. ACM Press.

[DHW94] U. Dayal, E. N. Hanson, and J. Widom. *Modern Database Systems: The Object Model, Interoperability and Beyond*, chapter Active Database Systems. Addison-Wesley, Reading, MA, USA, September 1994.

[DI01] P. Donzelli and G. Iazeolla. A hybrid software process simulation model. *Software Process: Improvement and Practice*, 6(2), 2001.

[DKH+02] K. Druschky, M. Kaltenhauser, C. Hummel, A. Druschky, E. Pauli, WJ. Huk WJ, H. Stefan, and B. Neundorfer. Somatotopic organization of the ventral and dorsal finger surface representations in human primary sensory cortex evaluated by magnetoencephalography. *Neuroimage*, 15(1):182–189, January 2002.

[1] O. Deux is a pseudonym for the persons who particicated in the design or implementation of the O_2 Database System

[Elm92] Ahmed K. Elmagarmid, editor. *Database Transaction Models For Advanced Applications*. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1992.

[FFF+97] R. Frackowiak, K. Friston, C. Firth, R. Dolan, and J. Mazziotta. *Human Brain Function*. Academic Press, London, England, 1997.

[FJL+97] S. Floyd, V. Jacobson, C-G Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE ACM Transactions on Networking*, 5(6):784–803, 1997.

[FR97] G. Fahl and T. Risch. Query processing over object views of relational data. *VLDB Journal*, November 1997.

[Fre99] J. Fredriksson. Design of an internet accessible visual human brain database system. In *Proc. Int. Conference on Multimedia Computing and Systems*, pages 469–474, Florence, Italy, 1999. IEEE Computer Science.

[FRS99] J. Fredriksson, P. Roland, and P. Svensson. Design and rationale of the european computerized human brain database. In *Proc. 11th Int. Conference on Scientific and Statistical Database Management*, pages 148–157, Cleveland, Ohio, USA, 1999. IEEE Computer Science.

[FS97] M. Fayad and D. Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38, October 1997.

[FS01] J. Fredriksson and P. Svensson. Evolutionary design and development of image meta-analysis environments based on object-relational database mediator technology. In *Proc. 13th Int. Scientific and Statistical Database Management 2001*, pages 190–199, Fairfax, Virginia, USA, 2001. IEEE Computer Science.

[FSR01] J. Fredriksson, P. Svensson, and T. Risch. Mediator-based evolutionary design and development of image meta-analysis environments. *J. Intelligent Information Systems*, 17(2/3):301–322, December 2001.

[FvE91] D. J. Felleman and D. C. van Essen. Distributed hierarchical processing in primate cerebral cortex. *Cerebral Cortex*, 1:1–47, 1991.

[G+96] H. Gomaa et al. A knowledge-based software engineering environment for reusable software requirements and architectures. *Journal of Automated Software Engineering*, 3(3/4), August 1996.

[GD93] S. Gatziu and K. R. Dittrich. Events in an active object-oriented database system. In Norman W. Paton and M. Howard Williams, editors, *Proceedings of the 1st International Workshop on Rules in Database Systems (RIDS)*, pages 23–39, Edinburgh, Scotland, 1993. Springer Verlag.

[GHJV93] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: Abstraction of reuse of object-oriented design. In *ECOOP '93 Conference Proceedings*, Lecture Notes in Computer Science, pages 406–431. Springer-Verlag, 1993.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, USA, 1995.

[GHR94] E. Gallopoulos, E. Houstis, and J. R. Rice. Problem-solving environments for computational science. *IEEE Computational Science and Engineering*, 1(2):11–23, 1994.

[GHS95] D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, April 1995.

[Gib94] W. W. Gibbs. Software's chronic crisis. *Scientific American*, pages 72–81, September 1994.

[GKF00] H. Gomaa, L. Kerschberg, and G. K. Farrukh. Domain modeling of software process models. In *IEEE International Conference on Engineering of Complex Computer Systems*, Tokyo, Japan, September 2000. IEEE Computer Society.

[GLM00] A. Gupta, B. Ludäscher, and M. E. Martone. Knowledge-based integration of neuroscience data sources. In *12th Int. Conf. on Statistical and Scientific Database Management*, Berlin, Germany, July 2000. IEEE Computer Society.

[GLN01] C. R. Genovese, N. A. Lazar, and T. Nichols. Thresholding of statistical maps in functional neuroimaging using the false

discovery rate. Technical Report 735, Carnegie-Mellon Dept. of Statistics, 2001.

[GMS87] H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of the ACM Conference of Management of Data*, pages 249–259, May 1987.

[GR83] A. Goldberg and D. Robson. *Smalltalk-80: The language and its implementation.* Addison Wesley, 1983.

[Gra76] F. Graybill. *Theory and Application of the Linear Model.* Duxbury Press, 1976.

[H+01] J. D. Van Horn et al. The functional magnetic resonance imaging data center (fMRIDC): the challenges and rewards of large-scale databasing of neuroimaging studies. *Philos Trans R Soc Lond B Biol Sci.*, 356(1412):1323–39, 2001.

[Hal] H. Halldorsson. Personal communication. The NeuroGenerator processing chain composition.

[HDJM+00] F. W. Howell, J. Dyrhfjeld-Johnsen, R. Maex, N. Goddard, and E. De Schutter. A large scale model of the cerebellar cortex using pgenesis. *Neurocomputing*, 32:1041–1036, 2000.

[HM92] J. D. Van Horn and I.C. McManus. Ventricular enlargement in schizophrenia. a meta-analysis of studies of the ventricle:brain ratio (vbr). *Br J Psychiatry*, 160:687–97, 1992.

[Hol96] A. P. Holmes. Nonparametric analysis of statistical images from functional mapping experiments. *Journal of Cerebral Blood Flow Metabolism*, 16(1):7–22, Januari 1996.

[HW68] D.H. Hubel and T.N Wiesel. Receptive fields and functional architecture or the monkey striate cortex. *Journal of Physiology*, 195:215–243, 1968.

[ILGP96] Y. Ioannidis, M. Livny, S. Gupta, and N. Ponnekanti. Zoo: A desktop experiment management environment. In *Proceedings of 22nd International VLDB Conference*, pages 274–285, Bombay, India, September 1996.

[JCJv92] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach.* ACM Press/Addison-Wesley, Reading, MA, USA, 1992.

[JHo02] JHotDraw, April 2002. `http://jhotdraw.sourceforge.net`.

[Joh92] R. E. Johnson. Documenting frameworks using patterns. In Andreas Paepcke, editor, *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, volume 27, pages 63–72, New York, NY, USA, 1992. ACM Press.

[Joh96] R. E. Johnson. The type object pattern. In *EuroPLoP 1996 Conference Proceedings*, July 1996.

[Jos99] V. Josifovski. *Design, Implementation and Evaluation of a Distributed Mediator System for Data Integration.* Ph.d. dissertation, Dept. of Computer and Information Science, Linköping Institute of Technology, 1999.

[KP88] G. E. Krasner and S. T. Pope. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object Oriented Programming*, 1(3):26–49, August/September 1988.

[Kre92] C. Kreuger. Software reuse,. *ACM Computing Surveys*, 24(2):131–183, June 1992.

[KS94] N. Krishnakumar and A. Sheth. Specification of workflows with heterogeneous tasks in METEOR. Technical Report TM-24198, Bellcore, 1994.

[KS95] N. Krishnakumar and A. Sheth. Managing heterogeneous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3(2):155–186, April 1995.

[KSJ99] E. Kandel, J. Schwarz, and T. Jessell. *Priciples of Neural Science.* Elsevier, October 1999.

[L+91] P. Lyngbaek et al. OSQL: A language for object databases. Technical Report HPL-DTD-91-4, HP Labs, 1991.

[LAG+02] J. Larsson, K. Amunts, B. Gulyás, A. Malikovic, K. Zilles, and P. Roland. Perceptual segregation of overlapping shapes activates posterior extrastriate visual cortex in man. *Experimental Brain Research*, 143(1):1–10, March 2002.

[LDBo] Jason Leigh, Thomas A. DeFanti, Brad B. Blumenthal, and others. The GENESIS simulation-based neural modeling database.

[Led00] A. Ledberg. Robust estimation of the probabilities of 3d clusters in functional brain images: Application to pet data. *Human Brain Mapping*, 13:185–198, 2000.

[Led01] A. Ledberg. *Measuring Brain Functions: Statistical Tests for Neuroimaging Data.* Phd thesis, Karolinska Institute, Stockholm, Sweden, September 2001.

[LFDM94] J. L. Lancaster, P. T. Fox, G. Davis, and S. Mikiten. BrainMap: A database of human functional brain mapping. In *The Fifth International Conference: Peace through Mind/Brain Science*, Hammamatsu, Japan, February 1994.

[LKA+95] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann. Specification and analysis of system architecture using Rapide. *IEEE Transactions on Software Engineering*, 21(4):336–355, 1995.

[LLM98] N. Lévy, F. Losavio, and A. Matteo. Comparing architectural styles:broker specializes mediator. In *Proceedings of the Third International Workshop on Software Architecture*, pages 93–96. ACM Press, 1998.

[LRK00] H. Lin, T. Risch, and T. Katchaounov. Object-oriented mediator queries to XML data on web information systems engineering. In *Proc. 1st Intl. Conf.*, pages 38–45, Hong Kong, China, June 2000.

[Man01] Dragos-Anton Manolescu. An extensible workflow architecture with objects and patterns. In *To appear in TOOLSEE 2001 Proceedings*, 2001.

[MB99] C. Moonen and P. Bandettini, editors. *Functional MRI.* Springer, Berlin, 1999.

[MDH99] Vasileios Megalooikonomou, Christos Davatzikos, and Edward Herskovits. Mining lesion-deficit associations in a brain image database. In *Knowledge Discovery and Data Mining*, pages 347–351, 1999.

[MJ98] Dragos-Anton Manolescu and Ralph E. Johnson. Patterns of workflow management facility. Available on the web, November 1998. http://micro-workflow.com/PDF/PWFMF.pdf.

[MS00] M. Ma and G. M. Shepherd. Functional mosaic organization of mouse olfactory receptor neurons. *Proceedings of the National Academy of Sciences*, 97(23):12869–12874, November 2000.

[NH02] F. Å. Nielsen and L. K. Hansen. Modeling of activation data in the BrainMap database: Detection of outliers. *Human Brain Mapping*, 15(3):146–56, March 2002.

[OfHBM01] OHBM Organization for Human Brain Mapping. Neuroimaging databases. *Science*, 292(5522):1673–6, June 2001.

[OMG02] Workflow management facility specification v1.2. Available on the web, May 2002. `http://www.omg.org/cgi-bin/doc?formal/2000-05-02`.

[OR96] K. Orsborn and T. Risch. *Advances in Computational Structures Technology*, chapter Next Generation of O-O Database Techniques in Finite Element Analysis, pages 121–136. Civil-Comp Press, 1996.

[Ors96] K. Orsborn. *On Extensible and Object-Relational Database Technology for Finite Element Analysis Application.* Ph.d. dissertation, Dept. of Computer and Information Science, Linköping Institute of Technology, 1996.

[Ouk01] K. Oukbir. *A Database Query Language for Uncertain Spatial Data.* Ph. lic. dissertation, Royal Institute of Technology, Dept. of Numerical Analysis and Computing Science, Stockholm, Sweden, 2001.

[PLGL00] R. Peyron, B. Laurent, and L. Garcia-Larrea. Functional imaging of brain responses to pain. a review and meta-analysis. *Neurophysiologie Clinique*, 30(5):263–88., October 2000.

[PMHJ98] S. G. Parker, M. Miller, C. D. Hansen, and C. R. Johnson. An integrated problem solving environment: The SCIRun computational steering system. In *31st Hawaii International Conference on System Sciences*, pages 147–156, January 1998.

[Pre95] W. Pree. *Design Patterns for Object-Oriented Software Development.* Addison-Wesley, 1995.

[R+02a] Tore Risch et al. Amos II user's guide, March 2002. `http://www.dis.uu.se/~udbl/amos/doc/amos_users_guide.html`.

[R+02b] P. Roland et al. The NeuroGenerator project, May 2002. `http://www.neurogenerator.org`.

[RB96] J. R. Rice and R. F Boisvert. From scientific software libraries to problem-solving environments. *IEEE Computational Science and Engineering*, 3(4):44–53, 1996.

[RJ96] D. Roberts and R. Johnson. Evolving frameworks: A pattern language for developing object-oriented frameworks. In *Proceedings of Pattern Languages of Programs*, Allerton Park, Illinois, September 1996.

[Rol93] P. Roland. *Brain Activation*. John Wiley, New York, NY, USA, 1993.

[Rol02] P. E. Roland. Dynamic depolarization fields in the cerebral cortex. *Trends in Neurosciences*, 25(4):183–190, April 2002.

[RS97] M. Roth and P. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In *Proc. VLDB Conference*, 1997.

[RSL+01] P. Roland, G. Svensson, T. Lindeberg, T. Risch, P. Baumann, A. Dehmel, J. Fredriksson, H. Halldorsson, L. Forsberg, Y. Young, and K. Zilles. A database generator for human brain imaging. *Trends in Neurosciences*, 24(10):562–564, 2001.

[RZ96] P. Roland and K. Zilles. The developing european computerized human brain database for all imaging modalities. *Neuroimage*, 4(3):39–47, December 1996.

[S+01] K. E. Stephan et al. Advanced database methodology for the collation of connectivity data on the macaque brain (CoCoMac). *Philos Trans R Soc Lond B Biol Sci.*, 356(1412):1159–86, August 2001.

[SB92] D. D. Stark and W. G. Bradley. *Magnetic resonance imaging. Second Edition*. Mosby Year Book, Inc, 1992.

[SF02] G. Svensson and L. Forsberg. The neurogenerator database submission procedure and user interface (abstract). In *To appear in Proc. of Hum. Brain Mapp. Conference*, Sendai, Japan, 2002.

[She98] G. M. Shepherd, editor. *The Synaptic Organization of the Brain.* Oxford University Press, fourth edition, 1998.

[Shi81] D. Shipman. The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems*, 6(1), 1981.

[SK91] S. A. Siegelbaum and E. R. Kandel. Learning-related synaptic plasticity: LTP and LTD. *Curr Opin Neurobiol.*, 1(1):113–20, June 1991. Review.

[Skö94] M. Sköld. *Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques.* Ph. lic. dissertation, Dept. of Computer and Information Science, Linköping Institute of Technology, 1994.

[SM95] M. Stonebraker and D. Moore. *Object-Relational DBMSs: The Next Great Wave.* Morgan Kaufmann, 1995.

[Som01] I. Sommerville. *Software Engineering.* Addison Wesley, sixth edition, 2001.

[Str02] G. F. Stridter. Brain homology and function: An uneasy alliance. *Brain Res Bulletin*, 57(3-4):239–42, February 2002.

[Tha02] R. H. Thayer. Software system engineering: A tutorial. *IEEE Computer*, 35(4):68–73, April 2002.

[THF+97] H. Topcuoglu, S. Hariri, W. Furmanski, J. Valente, and I. Ra. The software architecture of a virtual distributed computing environment. In *Proc. 6th Int. Symp. on High Performance Distributed Computing*, pages 40–49. IEEE Computer Society Press, August 1997.

[UL02] C. Undeman and T. Lindeberg. Automatic brain segmentation using probablistic diffusion and watershed analysis. in preparation, 2002.

[vdA94] W. M. P. van der Aalst. Modelling and analysing workflow using a petri-net based approach. In *Proc. 2 nd Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pages 31–50, 1994.

[vdA97] W. M. P. van der Aalst. Verification of workflow nets. In P. Azema and G. Balbo, editors, *Application and Theory of Petri Nets*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, Germany, 1997.

[vdA98] W. M. P. van der Aalst. The application of Petri Nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[vdA99] W. M. P. van der Aalst. Generic workflow models: How to handle dynamic change and capture management information. In M. Lenzerini and U. Dayal, editors, *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems (CoopIS'99)*, pages 115–126, Edinburgh, Scotland, September 1999. IEEE Computer Society Press.

[vdABtHK00] W. M. P. van der Aalst, A. P. Barros, A. H. M. ter Hofstede, and B. Kiepuszewski. Advanced workflow patterns. In *Conference on Cooperative Information Systems*, pages 18–29, 2000.

[Vin97] S. Vinoski. CORBA: integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14(2), 1997.

[W+96] K. E. Ward et al. Meta-analysis of brain and cranial size in schizophrenia. *Schizophr Res.*, 22(3):197–213, December 1996.

[W+00] I. C. Wright et al. Meta-analysis of regional brain volumes in schizophrenia. *Am J Psychiatry*, 157(1):16–25, Jan 2000.

[WCM92] R. P. Woods, S. R. Cherry, and J. C. Mazziotta. Rapid automated algorithm for aligning and reslicing pet images. *Journal of Computer Assisted Tomography*, 16:620–33, 1992.

[Wie92] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25:38–49, 1992.

[WRL+00] D. Walker, O. Rana, M. Li, M. Shields, and Y. Huang. The software architecture of a distributed problem solving environment. *Concurrency, Practice and Experience*, 12(15):1455–1480, 2000.

[Zim94] W. Zimmer. Relationships between design patterns. In D. Schmidt J. Coplien, editor, *Pattern Languages Of Program Design 1*, pages 345–364, 1994.

[ZPHJ00] K. K. Zakzanis, P. Poulin, K. T. Hansen, and D. Jolic. Searching the schizophrenic brain for temporal lobe deficits: a systematic review and meta-analysis. *Psychol. Med.*, 30(3):491–504, May 2000.